



BrainStem Reference Manual

Release 2.12.3

Acroname, Inc.

May 19, 2026

1	Devices	1
1.1	USBHub3p	2
1.1.1	Quick Start Guide	2
1.1.2	Basic Example	3
1.1.3	Indicators and Connections	5
1.1.4	Programming Interface	6
1.1.5	USBHub3+ Module Entities	7
1.2	USBHub3c	24
1.2.1	Quick Start Guide	24
1.2.2	Basic Example	26
1.2.3	Indicators and Connections	27
1.2.4	Programming Interface	30
1.2.5	USBHub3c Module Entities	35
1.2.6	USBHub3c Software Features	60
1.3	USBHub2x4	78
1.3.1	Quick Start Guide	78
1.3.2	Basic Example	79
1.3.3	Indicators and Connections	82
1.3.4	Programming Interface	83
1.3.5	USBHub2x4 Module Entities	84
1.4	USB-C-Switch Pro	99
1.4.1	Quick Start Guide	99
1.4.2	Functionality	100
1.4.3	Indicators and Connections	105
1.4.4	Software Control	107
1.4.5	USB-C-Switch Pro Software Features	110
1.4.6	USB-C-Switch Pro Module Entities	116
1.4.7	USB-C-Switch Pro Cabling Guide	140
1.5	USB-C-Switch	144
1.5.1	Quick Start Guide	144
1.5.2	Basic Example	144
1.5.3	Indicators and Connections	147
1.5.4	Programming Interface	148
1.5.5	USB-C-Switch Module Entities	152
1.6	USBExt3c	165
1.6.1	Quick Start Guide	165
1.6.2	Functionality	166

1.6.3	Indicators and Connections	170
1.6.4	USBExt3c Software Features	172
1.6.5	USBExt3c Module Entities	184
1.7	MTM Products	213
1.7.1	MTM-Relay	213
1.7.2	MTM-DAQ-2	225
1.7.3	MTM-PM-1	239
1.7.4	MTM-Load-1	254
1.7.5	MTM-IO-Serial	269
1.7.6	MTM-EtherStem	293
1.7.7	MTM-USBStem	309
2	Software	327
2.1	HubTool	327
2.1.1	What can it do?	328
2.1.2	Which Acraname devices work with HubTool?	328
2.1.3	Host system requirements	329
2.1.4	Installation	329
2.1.5	Usage	329
2.1.6	Connecting devices	330
2.1.7	Control devices on remote hosts	331
2.1.8	Updating device firmware	332
2.1.9	Device-specific interfaces	332
2.2	BrainD	430
2.2.1	Installation	430
2.2.2	Quick Start Guide	432
2.2.3	Usage	434
2.2.4	Security	436
2.2.5	Configuration	437
2.2.6	Logging	441
2.2.7	Platform Specific Considerations	441
2.2.8	Functionality and Features	443
2.2.9	Audio-Video Conferencing Solutions	443
2.2.10	Test and Measurement Applications	443
2.3	ControlRoom	444
2.3.1	Installation	444
2.3.2	Usage	446
2.3.3	Advanced	456
2.3.4	Features and Functions	464
2.4	Q-Sys	464
2.4.1	Installation	465
2.4.2	Quick Start Guide	465
2.4.3	Usage	466
2.4.4	Features and Functions	466
2.5	DFU Automator	467
2.5.1	DFU Automator Features	467
3	API Reference	475
3.1	BrainStem Entities	476
3.1.1	Analog Entity	476
3.1.2	App Entity	477
3.1.3	Clock Entity	478
3.1.4	Digital Entity	480
3.1.5	Ethernet Entity	482

3.1.6	Equalizer Entity	486
3.1.7	HDBaseT Entity	487
3.1.8	I ² C Entity	488
3.1.9	Mux Entity	490
3.1.10	Pointer Entity	492
3.1.11	PoE Entity	494
3.1.12	Port Entity	495
3.1.13	Power Delivery Entity	504
3.1.14	Rail Entity	511
3.1.15	RCServo Entity	514
3.1.16	Relay Entity	516
3.1.17	Signal Entity	518
3.1.18	Store Entity	520
3.1.19	System Entity	522
3.1.20	Temperature Entity	530
3.1.21	Timer Entity	531
3.1.22	UART Entity	532
3.1.23	USB Entity	538
3.1.24	USB System Entity	545
3.2	Python API Reference	549
3.2.1	Getting (Quickly) Started	549
3.2.2	Acroname Modules	552
3.2.3	Package Structure	572
3.2.4	Definitions	574
3.2.5	Discovery	574
3.2.6	Entity	576
3.2.7	Link	580
3.2.8	Module	582
3.2.9	PDChannelLogger	586
3.2.10	Results	588
3.2.11	Version	589
3.2.12	Analog	592
3.2.13	App	596
3.2.14	Clock	597
3.2.15	Digital	600
3.2.16	Equalizer	603
3.2.17	Ethernet	604
3.2.18	HDBaseT	610
3.2.19	I2C	613
3.2.20	Mux	615
3.2.21	PoE	617
3.2.22	Pointer	624
3.2.23	Port	627
3.2.24	PowerDelivery	647
3.2.25	RCServo	662
3.2.26	Rail	664
3.2.27	Relay	671
3.2.28	Signal	672
3.2.29	Store	675
3.2.30	System	677
3.2.31	Temperature	690
3.2.32	Timer	691
3.2.33	UART	692
3.2.34	USB	697

	3.2.35	USBSystem	710
3.3		C++ API Reference	719
	3.3.1	Acroname Modules	719
	3.3.2	Entity Class	754
	3.3.3	Analog Class	759
	3.3.4	App Class	763
	3.3.5	Clock Class	764
	3.3.6	Digital Class	767
	3.3.7	Equalizer Class	769
	3.3.8	Ethernet Class	770
	3.3.9	HDBaseT Class	775
	3.3.10	I2C Class	777
	3.3.11	Mux Class	779
	3.3.12	PoE Class	781
	3.3.13	Pointer Class	786
	3.3.14	Port Class	789
	3.3.15	PowerDelivery Class	803
	3.3.16	RCServo Class	814
	3.3.17	Rail Class	815
	3.3.18	Relay Class	821
	3.3.19	Signal Class	822
	3.3.20	Store Class	824
	3.3.21	System Class	826
	3.3.22	Temperature Class	835
	3.3.23	Timer Class	836
	3.3.24	UART Class	837
	3.3.25	USB Class	841
	3.3.26	USBSystem Class	850
	3.3.27	Link Class	856
	3.3.28	Module Class	868
	3.3.29	PDChannelLogger	873
3.4		C API Reference	874
	3.4.1	aDefs.h	874
	3.4.2	aDiscovery.h	875
	3.4.3	Error Codes	879
	3.4.4	aFile.h	881
	3.4.5	aLink.h	885
	3.4.6	aMutex.h	888
	3.4.7	aPacket.h	890
	3.4.8	aProtocoldefs.h	892
	3.4.9	aStream.h	960
	3.4.10	aTime.h	971
	3.4.11	aUEI.h	972
	3.4.12	aVersion.h	974
	3.4.13	PortMapping.h	976
	3.4.14	bs_pd_packet.h	978
3.5		RESTful API Reference	980
	3.5.1	API Version v1	980
3.6		.NET API Reference	1175
	3.6.1	BrainStem2 CLI Types	1175
	3.6.2	Port Mapping	1177
	3.6.3	Module Class	1179
	3.6.4	Analog Class	1185
	3.6.5	App Class	1188

3.6.6	Clock Class	1190
3.6.7	Digital Class	1192
3.6.8	Equalizer Class	1195
3.6.9	Ethernet Class	1196
3.6.10	HDBaseT Class	1201
3.6.11	I2C Class	1204
3.6.12	Mux Class	1205
3.6.13	PoE Class	1208
3.6.14	Pointer Class	1212
3.6.15	Port Class	1215
3.6.16	PowerDelivery Class	1230
3.6.17	RCServo Class	1241
3.6.18	Rail Class	1242
3.6.19	Relay Class	1248
3.6.20	Signal Class	1249
3.6.21	Store Class	1251
3.6.22	System Class	1254
3.6.23	Temperature Class	1263
3.6.24	Timer Class	1264
3.6.25	UART Class	1265
3.6.26	USB Class	1269
3.6.27	USBSystem Class	1279
3.7	CCA API Reference	1285
3.7.1	Analog Entity	1285
3.7.2	App Entity	1290
3.7.3	Clock Entity	1291
3.7.4	Digital Entity	1294
3.7.5	Equalizer Entity	1298
3.7.6	Ethernet Entity	1299
3.7.7	HDBaseT Entity	1306
3.7.8	I2C Entity	1310
3.7.9	Mux Entity	1312
3.7.10	PoE Entity	1315
3.7.11	Pointer Entity	1322
3.7.12	Port Entity	1326
3.7.13	PowerDelivery Entity	1350
3.7.14	RCServo Entity	1365
3.7.15	Rail Entity	1367
3.7.16	Relay Entity	1376
3.7.17	Signal Entity	1377
3.7.18	Store Entity	1380
3.7.19	System Entity	1383
3.7.20	Temperature Entity	1398
3.7.21	Timer Entity	1399
3.7.22	UART Entity	1400
3.7.23	USB Entity	1406
3.7.24	USBSystem Entity	1422
3.7.25	Module Entity	1431
3.7.26	Link Entity	1434
3.7.27	PDChannelLogger	1436
3.7.28	PortMapping	1439
3.7.29	Version	1440
3.8	LabVIEW API Reference	1443
3.9	Q-Sys API Reference	1444

3.9.1	Product Plugins	1444
4	BrainStem	1451
4.1	What is BrainStem	1451
4.1.1	Scalable	1451
4.1.2	Usable	1452
4.1.3	Next Steps	1452
4.2	aEther	1452
4.2.1	Overview	1452
4.2.2	Communication Models	1452
4.3	Getting Started	1459
4.3.1	Do I need Drivers?	1459
4.3.2	Connecting to a BrainStem Device	1460
4.3.3	Launch HubTool	1461
4.3.4	Toggling the LED	1461
4.4	Firmware Management	1461
4.4.1	Updating Firmware via HubTool	1461
4.4.2	Updating Firmware via CLI	1467
4.4.3	Updating Software Licenses	1472
4.4.4	Updating Firmware without an Internet Connection	1474
4.4.5	Updating a Brainstem Module via the Brainstem Network.	1476
4.4.6	Recovering BrainStem Firmware via CLI	1479
4.4.7	Recovering MTM Module Firmware via CLI	1481
4.5	Terminology	1485
4.5.1	BrainStem® Network	1485
4.5.2	BrainStem® Bus	1486
4.5.3	Routing	1486
4.5.4	Module	1487
4.5.5	Host	1487
4.5.6	Entity	1487
4.5.7	Discovery	1487
4.6	USB Drivers	1487
4.6.1	Mac OS X	1488
4.6.2	Linux Ubuntu	1488
4.6.3	Windows 7 USB Driverless Installation	1488
4.7	Appendix	1488
4.7.1	Appendix I: BrainStem Universal Entity Interface (UEI)	1489
4.7.2	Appendix II: BrainStem Communication Protocol	1497
4.7.3	Appendix III: BrainStem Networking	1499
4.7.4	Appendix IV: Updater File Structure	1507
Index		1511

Here you can find software API documentation organized around specific Acroname devices. This documentation also includes specific behaviors, features, and configuration values that are unique to each device.

1.1 USBHub3p



The USBHub3+ gives engineers advanced flexibility and configurability over USB ports in testing and development applications for both USB 3.0 and USB 2.0 devices. The USBHub3+ hub architecture consists of two layers of internal hubs to achieve 8 fully controllable downstream ports.

To get up to speed with the USBHub3+ and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [capabilities](#) of the USBHub3+ for a more in depth view.

1.1.1 Quick Start Guide

Power

- Using the provided universal power supply connect the barrel jack into the hub.
- Connect the other end into a 120/240V AC outlet.

Data

- With the provided USB 3.0 A-B cable connect the A side to your host computer and the B side to the connection labeled “Up0”.

Download

- Download the [BrainStem Development Kit \(BDK\)](#)¹ for your particular operating system and architecture.
- Download [HubTool](#)² for your particular operating system and architecture.

Play

- Open HubTool
- On the bottom right side of the application select the USBHub3p device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the USBHub3p. For more information please take a look at our [Getting Started Guide](#)

1.1.2 Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

static const uint8_t PORT = 5;

int main(int argc, const char * argv[]) {

    //Create an instance of the USBHub3p
    aUSBHub3p hub;

    //Connect to USBHub3p
    aErr err = hub.discoverAndConnect(USB);
    if(err == aErrNone) {
        printf("Connected\r\n");
    }
    else {
        printf("Unable to discover device\r\n");
        return 1;
    }

    //Disable PORT
    hub.usb.setPortEnable(PORT);
}
```

(continues on next page)

¹ <https://acroname.com/api>

² <https://acroname.com/hubtool>

(continued from previous page)

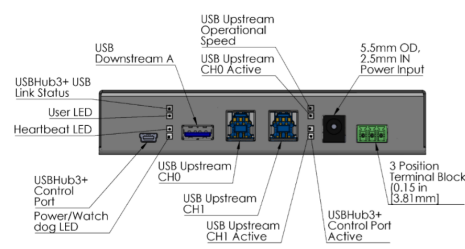
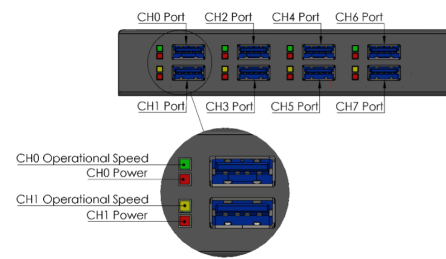
```
//////////  
//Do Stuff  
//////////  
  
//Enable PORT  
hub.usb.setPortDisable(PORT);  
  
//Disconnect  
hub.disconnect();  
  
return 0  
}
```

Python

```
import brainstem  
from brainstem.result import Result  
import sys  
  
PORT = 5  
  
#Create an instance of the USBHub3p  
hub = brainstem.stem.USBHub3p()  
  
#Connect to USBHub3p  
result = hub.discoverAndConnect(brainstem.link.Spec.USB)  
  
if result == Result.NO_ERROR:  
    print("Connected\r\n");  
else:  
    print("Unable to discover device\r\n");  
    sys.exit(1)  
  
hub.usb.setPortEnable(PORT)  
  
#####  
#Do Stuff  
#####  
  
hub.usb.setPortDisable(PORT)  
  
#Close the connection  
hub.disconnect()
```

1.1.3 Indicators and Connections

Connections



LEDs

LED Name	Color	Description
Link Status LED	Yellow	On once a host device has enumerated the BrainStem controller
User LED	Blue	Can be manipulated through any of the available APIs
Heartbeat LED	Green	Indicates active BrainStem connection; pulses at a rate determined by the system heartbeat rate
Power/Watchdog LED	Red and flashing blue	Solid red indicates the system is powered. Flashing blue is indication the internal watchdog is running and the USBHub3+ firmware is healthy
Upstream Operational Speed LED	Yellow or green	Upstream enumeration speed to host: green for SuperSpeed; yellow for Hi-Speed or lower USB 2.0 speeds.
Upstream 0 LED	Green	Indicates an active connection on upstream port
Upstream 1 LED	Green	
Control Port LED	Yellow	
Downstream Operational Speed LED	Yellow or green	Downstream device enumeration speed: green for SuperSpeed; yellow for Hi-Speed or lower USB 2.0 speeds; off when no device is enumerated
Downstream Power LED	Red	LED is on when downstream Vbus is enabled

1.1.4 Programming Interface

The USBHub3+ is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the USBHub3+.

A complete list of all entities and functions can be found in the Module *Module Entities* page.

Software Control

Software control of the features of the USBHub3+ is done with the BrainStem API via a BrainStem link. BrainStem links are done over USB and can be established via upstream port 0 (Up0), upstream port 1 (Up1), or the Control Port. After one or more of these ports is connected to a host machine, a user can connect to it via software API:

```
stem.link.discoverAndConnect(USB)
```

When multiple Acroname devices are connected to a host, connecting to a specific hub can be done by providing the hub serial number. Further, all connected devices can be found using

```
brainstem.discover.findAllModules(USB) (Python)
Acroname::BrainStem::Link::sDiscover() (C++)
```

BrainStem Control Port

The USBHub3+ also has a dedicated control channel on the USB mini-B connector. This is a full-speed USB 2.0 connection for BrainStem interface only. No USB hub traffic can flow on this connection. When a cable is connected to the mini-B connector, the BrainStem link can only be established through the Control Port, independent of the selected upstream port. The USB 3.0 type-B connectors are then used only for USB hub traffic to connect downstream USB devices. When the Control Port is not used, the BrainStem link will share the active upstream USB connection. Using the Control Port provides the ability to completely disconnect both USB upstream host connections while maintaining software control of the hub.

Using Multiple Hosts with USBHub3+

The two upstream-facing host ports can be connected to two different host computers. The control port can be attached to no computer, one of the same computers attached to the upstream ports, or a third host computer. Due to limitations of USB specification, only one host computer can access downstream USB ports at any time. Through the BrainStem API, the upstream port used can be controlled, or the system can automatically select the upstream port (see USB Hub Upstream Mode). When automatically selecting the upstream port, the USBHub3+ will favor using Up0 if it is connected.

Device Drivers

The USBHub3+ leverages operating system user space interfaces that do not require custom drivers for operation on modern operating systems.

Some older operating systems may require the installation of a BrainStem USB driver to enable software control. Installation details on installing USB drivers can be found within the BrainStem Development Kit under the “drivers” folder. For example, Windows 7 requires the supplied INF to communicate with BrainStem USB devices.

1.1.5 USBHub3+ Module Entities

Temperature

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Certain modules have a temperature measurement available. The temperature entity gives access to these measurements. Check your module datasheet to see if your module has a temperature entity.

System Temperature

The temperature of the USBHub3+ can be measured with:

```
stem.temperature[0].getValue(μC) \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

where temperature is in micro-degrees Celcius.

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every USBHub3+ is assigned a unique serial number at the factory. This facilitates an arbitrary number of USBHub3+ devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) [cpp] [python] [NET] [CCA] [REST]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for USBHub3+ is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) [cpp] [python] [NET] [CCA] [REST]
```

Hardware Version

Get the module's hardware version information.

```
stem.system.getHardwareVersion(hardwareVersion) [cpp] [python] [NET] [CCA] [REST]
```

The hardware version is a 32-bit value that encodes hardware variant information. The format is:

Table 1: Hardware Version Bit Breakdown

Bits	Field	Description
24-31	Revision	Reserved (always 0)
16-23	Reserved	Reserved (always 0)
8-15	Variant	Hardware variant identifier
0-7	Reserved	Reserved (always 0)

Table 2: Hardware Variant Values

Value	Description
'2'	2 Hub Chip variant
'3'	3 Hub Chip variant

The hardware variant is determined by reading hardware strapping pins during initialization to identify the specific USB hub controller configuration.

System Error Status Mapping

Error states for the device as a whole is bit-packed in a 32-bit word available from:

```
stem.system.getErrors() [cpp] [python] [NET] [CCA] [REST]
```

Calling this function will clear the current errors. If the error persists it will be set again.

Details about the system error status 32-bit word are as follows:

Bit	Port Error Status (channel) Result Bitwise Description
0	Thermal Protection Mode
1	Power Limit Exceeded
2:31	Reserved

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the USBHub3+ away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Pressing the reset button two times within 5 seconds will return all settings to factory defaults: all ports' data (HS and SS) and power enabled, CDP mode, enumeration delay of 0, 4095mA current limit.

Savable Items	
Software Offset	I2C Rate
Router Address	Port Enumeration Delay
Boot Slot	Downstream Boost
Port Mode (SDP, CDP) – each port	Current Limit – per port
Upstream Boost	Port state (data and power)
Upstream Port	

USB

API Documentation: [cpp] [python] [NET] [CCA] [REST]

The USB Entity provides the software control interface for USB related features. This entity is supported by BrainStem products which have programmatically controlled USB features.

USB Downstream Channel Control

Downstream USB channels can be manipulated through the `usb` entity command to enable and disable USB data and Vbus lines, measure current, measure Vbus voltage, boost data line signals, and measure temperature.

Manipulating Hi-Speed data, SuperSpeed data, and Vbus lines simultaneously for a single port can be done by calling the following methods with channel in [0-7] being the port index:

```
stem.usb.setPortEnable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

Manipulating Hi-Speed data and SuperSpeed data lines while not affecting the Vbus lines simultaneously for a single port can be done by calling the following method with channel [0-7]:

```
stem.usb.setDataEnable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setDataDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB 2.0 Hi-Speed data lines for a single port can be done by calling the following method with channel [0-7]:

```
stem.usb.setHiSpeedDataEnable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setHiSpeedDataDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB 3.1 SuperSpeed data lines for a single port can be done by calling the following method with channel [0-7]:

```
stem.usb.setSuperSpeedDataEnable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setSuperSpeedDataDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB Vbus line for a single port can be done by calling the following method with channel [0-7]:

```
stem.usb.setPowerEnable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPowerDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

To affect multiple ports and lines simultaneously, see `usb.setHubMode()` later in this section.

USB Downstream Measurements

The USB Vbus voltage, as well as the current consumed on Vbus, can be read for each channel by calling the following methods with channel [0-7], where the second variable passed into the method is the location for the measurement result:

```
stem.usb.getPortVoltage(channel, μV) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getPortCurrent(channel, μA) [cpp] [python] [NET] [CCA] [REST]
```


USB Downstream Current Limiting

Current-limit trip point settings can be accessed for each port by calling the following methods with channel [0-7], where the second variable passed into the method is either the set value or the write location of the result:

```
stem.usb.getPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

The USBHub3+ current limit behavior follows the USB BC1.2 defined “trip off” behavior. When a downstream device consumes more current than the set current limit, the Vbus voltage will immediately turn off and latch off until the port is re-enabled. An overcurrent error flag is set in getPortState() bitfield. The voltage-current behavior is detailed in Figure 6.

USB Downstream Enumeration Speed

The enumeration state and speed of each downstream port can be read with

```
stem.usb.getDownstreamDataSpeed [cpp] [python] [NET] [CCA] [REST]
```

Value	Hub Downstream Speed Descriptions
0	No device enumerated
1	Hi-Speed device enumerated
2	SuperSpeed device enumerated

USB Downstream Operational Mode

The USB port operational mode controls the behavior of each downstream port’s charging behavior. Each port can be setup to support different modes in the USB Battery Charge Specification 1.2 (BC1.2). Standard Downstream Port (SDP) mode will cause BC1.2 compliant or older USB devices to consume 500mA or less. Configuring a port as a Charging Downstream Port (CDP) will cause the hub signal to downstream devices that devices may consume up to 5A, the maximum allowed by BC1.2. If there is no upstream USB host connected to the hub, downstream ports set to CDP will behave as Dedicated Charging Ports (DCP).

The actual current consumed by the device is controlled by the downstream device and not the USBHub3+. Devices which are not compliant with BC1.2 or the previous USB power specifications may draw more current than specified above.

The operational mode is set or read by calling the methods:

```
stem.usb.getPortMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

Value	Hub Port Mode Descriptions
0	Standard downstream port (SDP)
1	Charging downstream port (CDP)

Note: A system.save() and system.reset() is required before the new setting will take affect.

USB Downstream Enumeration Delay

Once a USB device is detected by the USBHub3+ it is possible to delay its connection to an upstream host computer and subsequent enumeration on the USB bus. The enumeration delay can mitigate or eliminate host kernel instabilities by forcing devices to enumerate in slow succession, allowing a focus on validation of drivers and software. The enumeration delay is configured in milliseconds, representing the time delay between enabling each successive downstream port from 0 to 7. Enumeration delay is applied when the hub powers on or when a new upstream connection is made.

```
stem.usb.setEnumerationDelay(delay) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getEnumerationDelay(delay) [cpp] [python] [NET] [CCA] [REST]
```

USB Boost Mode

Boost mode increases the drive strength of the USB 2.0 Hi-Speed data signals (SuperSpeed data and power signals are not changed). Boosting the data signal drive strength may help to overcome connectivity issues when using long cables or connecting through relays, “pogo” pins or other adverse conditions. This setting is applied after a `system.save()` call and reset or power cycle of the hub. The system setting is persistent until changed or the hub is hard reset. After a hard reset, the default value of 0% boost is restored. A hard reset is done by pressing the “Reset” button on the back of the hub while the hub is powered.

Boost mode can be applied to both the upstream and downstream USB ports with the follow methods:

```
stem.usb.getDownstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setDownstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getUpstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setUpstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```

The setting parameter is an integer that correlates to the following:

Value	Hub Boost Mode Descriptions
0	Normal drive strength
1	4% increase in drive strength
2	8% increase in drive strength
3	12% increase in drive strength

USB Hub Upstream Channels

The USBHub3+ is perfect for environments where multiple devices need to be shared or switched between two host computers using two host (upstream) connections via USB standard-B connectors. The upstream connection can be automatically detected or specifically selected using the following methods:

```
stem.usb.getUpstreamMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setUpstreamMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

The mode parameter can be defined as the following:

Value	Definitions	Hub Upstream Mode Descriptions
0	usbUpstreamModePort0	Force upstream port 0 to be selected
1	usbUpstreamModePort1	Force upstream port 1 to be selected
2	usbUpstreamModeAuto	Automatically detect upstream port
255	usbUpstreamModeNone	Disconnect both upstream ports

Predefined C++ macros for these can be found in `aProtocoldef.h`, and Python's built-in help interface.

The default operational mode is to auto detect which upstream USB port is selected. Automatic detection uses the presence of Vbus on the USB type-B upstream connector to determine presence of a host. If only one upstream port is connected to a host, it will be used for upstream USB. If both upstream ports are connected, the hub will use upstream port 0.

If the Hub Upstream Mode is set to disconnect both upstream ports (or the only active upstream port), the only path available to establish a BrainStem link to the USBHub3+ will be via a host connected to the BrainStem Control Port. See Figure 9 for more details.

USB Hub Upstream State

The USBHub3+ can provide status information on which upstream port is actively selected as data path to the downstream ports:

```
stem.usb.getUpstreamState(mode) [cpp] [python] [NET] [CCA] [REST]
```

This command returns a 32-bit value which indicates:

Value	Definitions	Hub Upstream Mode Descriptions
0	usbUpstreamStatePort0	Upstream port 0 is actively selected
1	usbUpstreamStatePort1	Upstream port 1 is actively selected
2	usbUpstreamStateNone	No upstream port is selected

USB Hub Operational Mode

In addition to targeting individual downstream USB ports, a bit-mapped hub mode interface is also available. This interface allows the reading or setting of all USB downstream ports in one functional call.

Auto VBus Toggle

By default the USBHub3+ will toggle its downstream ports anytime the host connection is lost, changed or disconnected. Disabling (setting the bit) will cause the hub to not cycle downstream power on upstream changes. This behavior can be helpful for certain host controllers and devices. Enumeration delay will override this setting.

```
stem.usb.getHubMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setHubMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

Bit	Hub Operational Mode Word Definition
0	USB Ch 0 USB Hi-Speed Data Enabled

continues on next page

Table 3 – continued from previous page

Bit	Hub Operational Mode Word Definition
1	USB Ch 0 USB Vbus Enabled
2	USB Ch 1 USB Hi-Speed Data Enabled
3	USB Ch 1 USB Vbus Enabled
4	USB Ch 2 USB Hi-Speed Data Enabled
5	USB Ch 2 USB Vbus Enabled
6	USB Ch 3 USB Hi-Speed Data Enabled
7	USB Ch 3 USB Vbus Enabled
8	USB Ch 4 USB Hi-Speed Data Enabled
9	USB Ch 4 USB Vbus Enabled
10	USB Ch 5 USB Hi-Speed Data Enabled
11	USB Ch 5 USB Vbus Enabled
12	USB Ch 6 USB Hi-Speed Data Enabled
13	USB Ch 6 USB Vbus Enabled
14	USB Ch 7 USB Hi-Speed Data Enabled
15	USB Ch 7 USB Vbus Enabled
16	USB Ch 0 USB SuperSpeed Data Enabled
17	Reserved
18	USB Ch 1 USB SuperSpeed Data Enabled
19	Reserved
20	USB Ch 2 USB SuperSpeed Data Enabled
21	Reserved
22	USB Ch 3 USB SuperSpeed Data Enabled
23	Reserved
24	USB Ch 4 USB SuperSpeed Data Enabled
25	Reserved
26	USB Ch 5 USB Super Speed Data Enabled
27	Reserved
28	USB Ch 6 USB SuperSpeed Data Enabled
29	Reserved
30	USB Ch 7 USB SuperSpeed Data Enabled
31	Auto VBus Toggle Disable

USB Port State

Each downstream port reports information regarding its operating state represented in bit-packed results from:

```
stem.usb.getPortState(state) [cpp] [python] [NET] [CCA] [REST]
```

where channel can be [0-7], and the value status is 32-bit word, defined as the following:

Bit	Port State: Result Bitwise Description
0	USB Vbus Enabled
1	USB2 Data Enabled
2	Reserved
3	USB3 Data Enabled
4:10	Reserved
11	USB2 Device Attached
12	USB3 Device Attached
13:18	Reserved
19	USB Error Flag
20	USB2 Boost Enabled
21:22	Reserved
23	Device Attached
24:31:00	Reserved

USB Port Error Status Mapping

Error states for all downstream ports are bit-packed in 32-bit words available from:

```
stem.usb.getPortError(channel) [cpp] [python] [NET] [CCA] [REST]
```

where channel is [0-7].

Errors can be cleared on each individual channel by calling the following method:

```
stem.usb.clearPortErrorStatus(channel) [cpp] [python] [NET] [CCA] [REST]
```

Calling this command clears the port-related error bit flags (see Table 7) in the port error state. Global bits for hub errors cannot be cleared by this command.

Details about the port error status 32-bit word are as follows:

Bit	Port Error Status (channel) Result Bitwise Description
0	USB port current limit exceeded
1	USB port back-drive condition detected
2	Hub external power not present
3	Hub overtemperature condition
4	USB port short-circuit condition
5:31	Reserved

Port

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Port Entity provides control over the most basic items related to a USB Port. This includes actions ranging from a complete port enable and disable to the individual interface control. Voltage and current measurements are also included for devices which support the Port Entity.

Port Control

The USBHub3p has a Port Entity for every port on the device; however, not all ports have the same capabilities. These ports can be referenced by their instance (port[x]) index.

Port Label	Index (port[x])
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
Down-A	8
Up0	9
Up1	10
Control	11

One of the most powerful features of the USBHub3p is its ability to turn ports on and off which is available on Ports 0-7.

```
stem.hub.port[x].setEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB Vbus line for a single port can be done by calling the following method on Ports 0-7.

```
stem.hub.port[x].setPowerEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating Hi-Speed data and SuperSpeed data lines while not affecting the Vbus lines simultaneously for a single port can be done by calling the following method for Ports 0-7.

```
stem.hub.port[x].setDataEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB 2.0 Hi-Speed data lines for a single port can be done by calling the following for Ports 0-7.

```
stem.hub.port[x].setDataHSEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataHSEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB 3.1 SuperSpeed data lines for a single port can be done by calling the following method for Ports 0-7.

```
stem.hub.port[x].setDataSSEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataSSEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Voltage and Current Measurements

The USBHub3p provides Voltage and Current measurements for Vbus. These values can be acquired for all 8 ports through the following APIs

```
stem.hub.port[x].getVbusVoltage() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getVbusCurrent() [cpp] [python] [NET] [CCA] [REST]
```

Power Modes

The ports of the USBHub3p are capable of providing power in multiple formats. The default is Charging Downstream Port (CDP), but that can be changed to things like: Standard Downstream Port (SDP), Charging Downstream Port (CDP) / Dedicated Charging Port (DCP). These modes can be set through:

```
stem.hub.port[x].setPowerMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerMode() [cpp] [python] [NET] [CCA] [REST]
```

Power Mode	Value	Define
None	0	portPowerMode_none_Value
SDP	1	portPowerMode_sdp_Value
CDP/DCP	2	portPowerMode_cdp_dcp_Value

Note: The Power Modes can only be changed when the port power is disabled.

Port Mode

As outlined in the “Port Control” section the USBHub3p can individually manipulate almost every pin on the connector; however, depending on your application that might require multiple function calls in order to configure the port how you want it. Port Mode on the other hand is a one stop shop that allows you to pick and choose which lines you want enabled or disabled through a single call. Additionally, it has a few other features tucked away inside of it.

```
stem.hub.port[x].setMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getMode() [cpp] [python] [NET] [CCA] [REST]
```

Port Mode Item	Bit	Value	Define
Power Enable	0	0/1	portPortMode_powerEnabled_Bit
HS 1 Enable	1	0/1	portPortMode_HS1Enabled_Bit
SS 1 Enable	3	0/1	portPortMode_SS1Enabled_Bit
Power Mode: Offset		16	portPortMode_portPowerMode_Offset
Power Mode: Mask		0x7	portPortMode_portPowerMode_Mask
Power Mode: None	16-18	0	portPortMode_portPowerMode_none_Value
Power Mode: SDP	16-18	1	portPortMode_portPowerMode_sdp_Value
Power Mode: CDP/DCP	16-18	2	portPortMode_cdp_dcp_Value

Data Role

The data role describes the current configuration of the port in regards to its data direction. In most cases this evaluates to an Upstream Facing Port (UFP) or a Downstream Facing Port (DFP). Upstream in this case means the host side of the port and Downstream refers to the device side. The Data Role can be aquired through:

```
stem.hub.port[x].getDataRole() [cpp] [python] [NET] [CCA] [REST]
```

Data Role	Value	Define
Disabled	0	portDataRole_Disabled_Value
Upstream	1	portDataRole_Upstream_Value
Downstream	2	portDataRole_Downstream_Value
Control	3	portDataRole_Control_Value

Port Limits and Modes

At the Port level the user has the ability to define current limit.

```
stem.hub.port[x].setCurrentLimit() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].getCurrentLimit() [cpp] [python] [NET] [CCA] [REST]
```

Downstream Data Speed

The USBHub3p can detect if a device has been enumerated. Additionally, it can detect at what speed a device has enumerated at.

```
stem.hub.port[x].getDataSpeed() [cpp] [python] [NET] [CCA] [REST]
```


Data Speed	Bit	Value	Define
1.5 Mbit/s	0	0/1	portDataSpeed_ls_1p5M_Bit
12 Mbit/s	1	0/1	portDataSpeed_fs_12M_Bit
480 Mbit/s	2	0/1	portDataSpeed_hs_480M_Bit
5 Gbit/s	3	0/1	portDataSpeed_ss_5G_Bit
10 Gbit/s	4	0/1	portDataSpeed_ss_10G_Bit
USB 2.0	6	0/1	portDataSpeed_Connected_2p0_Bit
USB 3.0	7	0/1	portDataSpeed_Connected_3p0_Bit

USB System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USBSystem class provides high level control of the lower level *Port Entity*

Upstream Control

The USBHub3p has the ability to designate one of the upstream ports (9-10) as the upstream connection. This is very useful for moving devices between hosts.

```
stem.hub.setUpstream() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.hub.getUpstream() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Enumeration Delay

Once a USB device is detected by the USBHub3p it is possible to delay its connection to an upstream host computer and subsequent enumeration on the USB bus. The enumeration delay can mitigate or eliminate host kernel instabilities by forcing devices to enumerate in slow succession, allowing a focus on validation of drivers and software. The enumeration delay is configured in milliseconds, representing the time delay between enabling each successive port. Enumeration delay is applied when the hub powers on or when a new upstream connection is made.

```
stem.hub.setEnumerationDelay() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.hub.getEnumerationDelay() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Data Behavior

The USBHub3p is capable of a few different behaviors for how it switches upstream port connections. It can auto switch based on port priority or have a fixed upstream port. The method in which these events are handled is referred to as data behavior.

List of Available Data Behaviors for USBHub3p

Behavior	Value	Define
Hard Coded	0	usbsystemDataBehavior_HardCoded
Reserved	1	usbsystemDataBehavior_Reserved
Port Priority	2	usbsystemDataBehavior_PortPriority

Hard Coded (Default Configuration)

The Hard Coded data behavior is used to fix the Upstream port to a single port and not allow it to move except for a command through the [Set Upstream](#) API.

Port Priority

The Port Priority data behavior prioritizes making the Upstream port the lowest numbered port on the USB-Hub3p that is capable of being an Upstream port.

Relevant API's

```
stem.hub.setDataRoleBehavior() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getDataRoleBehavior() [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
App[0-3]	execute	
	return	
system[0]	getModel	
	getHardwareVersion	
	getModule	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getVersion	
	getSerialNumber	

continues on next page

Table 4 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	save	
	reset	
	getInputVoltage	
	getInputCurrent	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	getUptime	
	getMaximumTemperature	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
timer[0-8]	getExpiration	
	setExpiration	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	
	getTransferStore	
	setTransferStore	
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	
	setInt	
temperature[0]	getTemperature	
usb[0]	setPortEnable	Channels 0-7
	setPortDisable	Channels 0-7
	setDataEnable	Channels 0-7
	setDataDisable	Channels 0-7
	setHiSpeedDataEnable	Channels 0-7
	setHiSpeedDataDisable	Channels 0-7
	setSuperSpeedDataEnable	Channels 0-7
	setSuperSpeedDataDisable	Channels 0-7
	setPowerEnable	Channels 0-7
	setPowerDisable	Channels 0-7
	getPortVoltage	Channels 0-7
	getPortCurrent	Channels 0-7
	getPortCurrentLimit	Channels 0-7
	setPortCurrentLimit	Channels 0-7

continues on next page

Table 4 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
port[0-7]	setPortMode	Channels 0-7
	getPortMode	Channels 0-7
	getDownstreamDataSpeed	Channels 0-7
	getHubMode	
	setHubMode	
	getPortState	Channels 0-7
	getPortError	
	getEnumerationDelay	
	setEnumerationDelay	
	clearPortErrorStatus	
	getUpstreamMode	
	setUpstreamMode	
	getUpstreamState	
	getUpstreamBoostMode	
	setUpstreamBoostMode	
	getDownstreamBoostMode	
	setDownstreamBoostMode	
	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getDataSSEnabled	
	setDataSSEnabled	
	getPowerEnabled	
	setPowerEnabled	
	getHSBoost	
	setHSBoost	
	getMode	
	setMode	
	getCurrentLimit	
	setCurrentLimit	
	getVbusVoltage	
	getVbusCurrent	
	getState	
	getName	
	setName	
	getPowerMode	
	setPowerMode	
	getDataRole	
	getDataSpeed	
	getErrors	
port[8]	getHSBoost	
	setHSBoost	
	getName	
	setName	
	getPowerMode	
	setPowerMode	
	getDataRole	

continues on next page

Table 4 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
port[9-10]	getHSBoost	
	setHSBoost	
	getName	
	setName	
	getDataRole	
port[11]	getName	
	setName	
	getDataRole	
USBSystem [0]	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
	getEnumerationDelay	
	setEnumerationDelay	

1.2 USBHub3c



The USBHub3c gives engineers advanced flexibility and configurability over USB ports in testing and development applications to validate, control, and test the limits of devices built on the Power Delivery (USB-PD) and USB specification.

To get up to speed with the USBHub3c and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [functionality](#) of the USBHub3c for a more in depth view.

1.2.1 Quick Start Guide

1. Download The Development Kit & HubTool

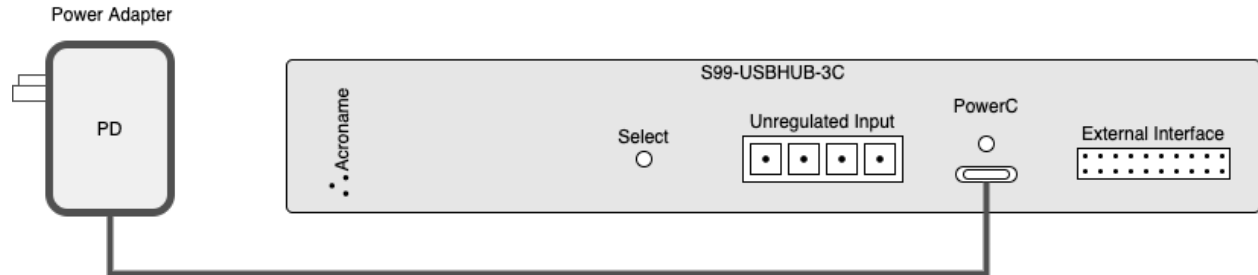
- Download the [BrainStem Development Kit \(BDK\)](#)³ for your particular operating system and architecture.
- Download [HubTool](#)⁴ for your particular operating system and architecture.

³ <https://acroname.com/api>

⁴ <https://acroname.com/hubtool>

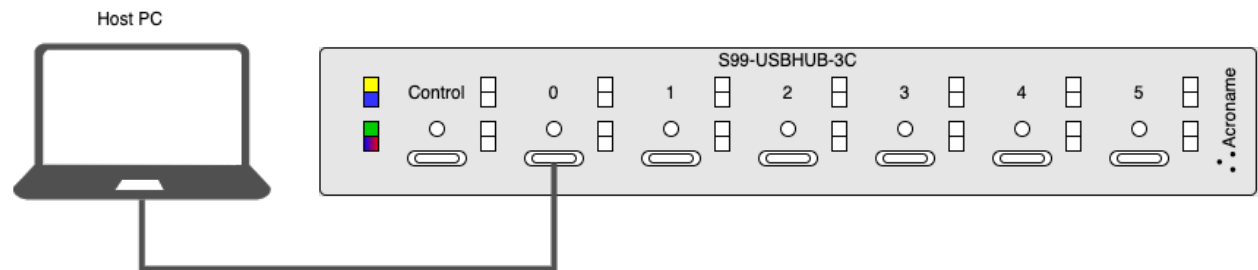
2. Connect Power

- Using the provided Power Delivery (PD) brick/wallwort and the USBU 3.0 C-C cable make a connection between it and the Power-C port located on the back of the USBHub3c.
- Plug the PD Brick into a 120/240V AC outlet.



3. Connect Data

- With a second USB 3.0 Type-C cable make a connection between Port “0” and the your host computer.



4. Play

- Open HubTool
- On the bottom right side of the application select the USBHub3c device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the USBHub3c. For more information please take a look at our [Getting Started Guide](#)

1.2.2 Basic Example

This simple example shows briefly how instantiate, connect to, disable and re-enable a port on the USBHub3c. There are multiple examples shipped with the [BrainStem Development Kit \(BDK\)](#)⁵ download.

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

static const uint8_t PORT = 5;

int main(int argc, const char * argv[]) {

    //Create an instance of the USBHub3c
    aUSBHub3c device;

    //Connect to USBHub3c
    aErr err = device.discoverAndConnect(USB);
    if(err == aErrNone) {
        printf("Connected\r\n");
    }
    else {
        printf("Unable to discover device\r\n");
        return 1;
    }

    //Disable PORT
    device.hub.port[PORT].setEnabled(0);

    ////////////
    //Do Stuff
    ////////////

    //Enable PORT
    device.hub.port[PORT].setEnabled(1);

    //Disconnect
    device.disconnect();

    return 0
}
```

Python

```
import brainstem
from brainstem.result import Result
import sys

PORT = 5

#Create an instance of the USBHub3c
device = brainstem.stem.USBHub3c()

#Connect to USBHub3c
result = device.discoverAndConnect(brainstem.link.Spec.USB)
```

(continues on next page)

⁵ <https://acroname.com/api>

(continued from previous page)

```

if result == Result.NO_ERROR:
    print("Connected\r\n");
else:
    print("Unable to discover device\r\n");
    sys.exit(1)

# Disable Port
device.hub.port[PORT].setEnabled(0)

#####
# Do Stuff
#####

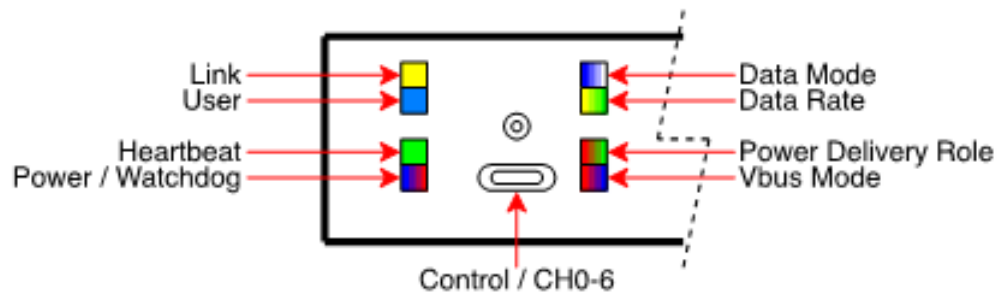
# Enable Port
device.hub.port[PORT].setEnabled(1)

#Close the connection
device.disconnect()

```

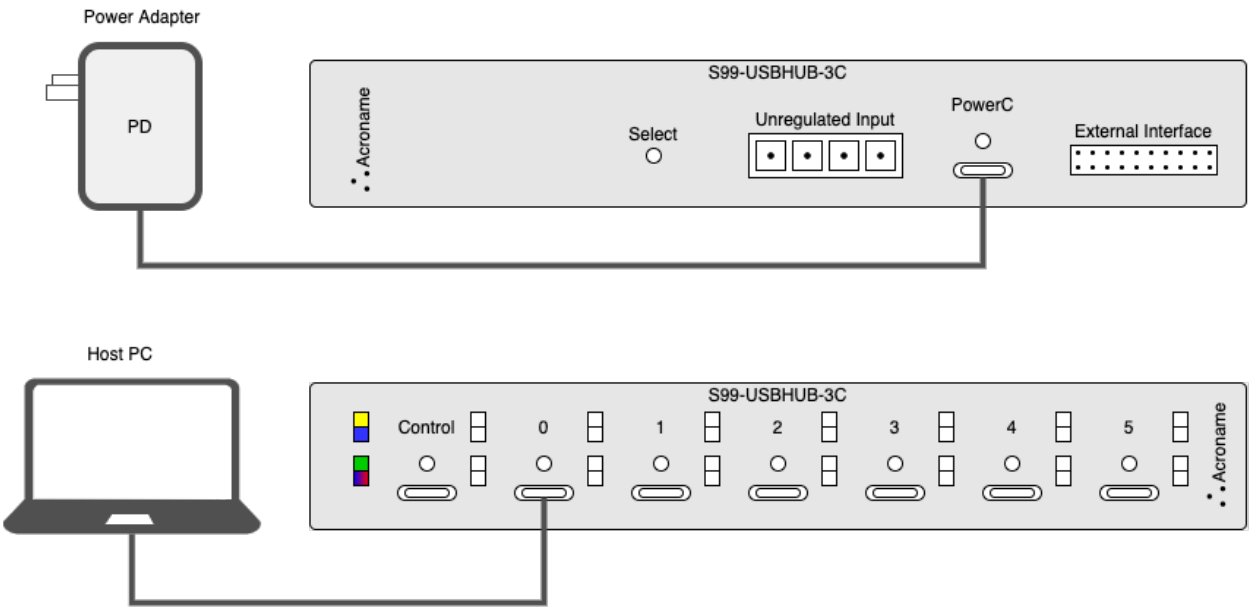
1.2.3 Indicators and Connections

LEDs

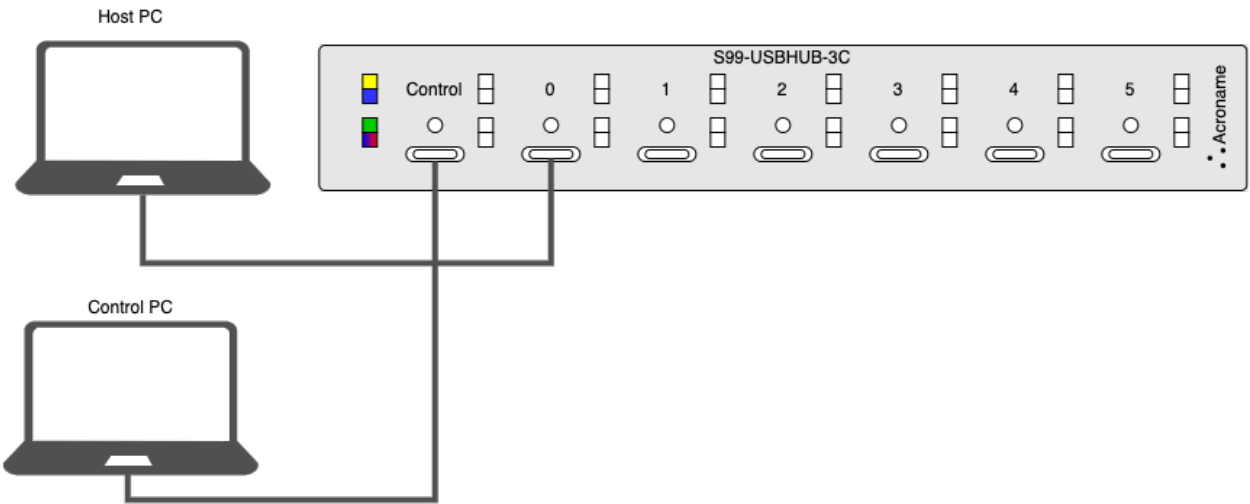


LED Name	Color	Description
Link Status LED	Yellow	On once a host device has enumerated the BrainStem controller
User LED	Blue	Can be manipulated through any of the available APIs
Heartbeat LED	Green	Indicates active BrainStem connection; pulses at a rate determined by the system heartbeat rate
Power/Watchdog LED	Red and flashing blue	Solid red indicates the system is powered. Flashing blue is indication the internal watchdog is running and the USBHub3c firmware is healthy
Data Mode	Green	Upstream Port
	Red	Downstream Port
	White	Control Port
Data Rate	Yellow	Downstream enumeration of USB 2.0 speeds.
	Green	Downstream enumeration of SuperSpeed (5Gbps)
	Blue	Downstream enumeration of SuperSpeed+ (10Gbps)
Power Role	Red	Connected: Port is Sourcing Power; Not Connected: Source Only
	Green	Connected: Port is Sinking Power; Not Connected: Sink Only
	Blue	Connected: N/A; Not Connected: Port is Dual Role Power Capable
Mode Mode	Red	SDP, CDP or DCP modes
	Blue	Power Delivery Mode
	Green	Quick Charge™ Mode
	White	Programable Power Supply Mode

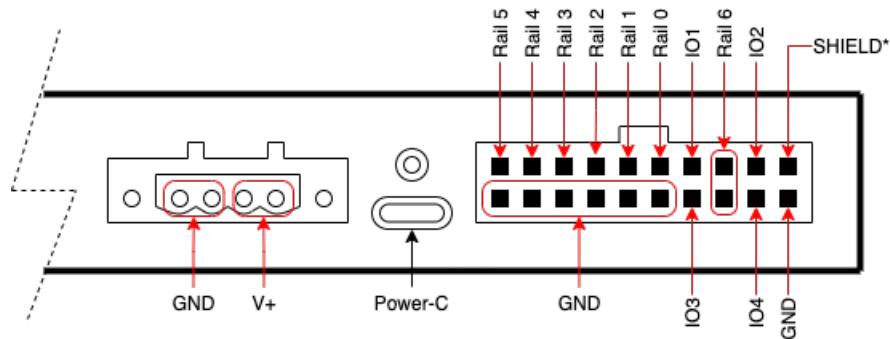
Connections



Separate Control Computer



External Connector



1.2.4 Programming Interface

Overview

The USBHub3c consists of 6 USB Type-C ports connected to a USB 3.2 Gen 2x1 Hub, USB Power Delivery, and Qualcomm QuickCharge hardware. The software control of this device manipulates various capabilities of each of these ports. The USBHub3c is one of a family of Acroname devices, and shares some of its feature set and interface with other Acroname devices.

Acroname provides software APIs for working with the USBHub3c in a number of different languages. We strive to keep much of the syntax and structure of the API similar across these languages. The core of this common protocol, API software structure, and nomenclature we call BrainStem (for more information see [BrainStem](#)).

Whether C++ or Python or another language interface is the target, all of our APIs start with the concept of a connection to the USBHub3c device. This is usually represented by a class, an instance of which represents a connection to a specific USBHub3c. We call these classes “Modules” or “Module classes.” Each class then contains a set of subclass instances which represent interface functionality for a specific piece of hardware functionality. We call these sub objects “Entities.” The following code block represents the “Module” class definition for the USBHub3c.

Module class

C++

```
class aUSBHub3c : public Acroname::BrainStem::Module
{
public:

    aUSBHub3c(const uint8_t module = aUSBHUB3P_MODULE,
              bool bAutoNetworking = true,
              const uint8_t model = aMODULE_TYPE_USBHub3c) :
        Acroname::BrainStem::Module(module, bAutoNetworking, model)
    {
        for(int x = 0; x < aUSBHUB3C_NUM_APPS; x++) {
            app[x].init(this, x);
        }
    }
};
```

(continues on next page)

(continued from previous page)

```

    }

    for(int x = 0; x < aUSBHUB3C_NUM_PD_PORTS; x++) {
        pd[x].init(this, x);
    }

    for(int x = 0; x < aUSBHUB3C_NUM_POINTERS; x++) {
        pointer[x].init(this, x);
    }

    store[aUSBHUB3C_STORE_INTERNAL_INDEX].init(this, storeInternalStore);
    store[aUSBHUB3C_STORE_RAM_INDEX].init(this, storeRAMStore);
    store[aUSBHUB3C_STORE_EEPROM_INDEX].init(this, storeEEPROMStore);

    system.init(this, 0);

    for(int x = 0; x < aUSBHUB3C_NUM_TEMPERATURES; x++) {
        temperature[x].init(this, x);
    }

    for(int x = 0; x < aUSBHUB3C_NUM_TIMERS; x++) {
        timer[x].init(this, x);
    }

    hub.init(this, 0);

    for(int x = 0; x < aUSBHUB3C_NUM_RAILS; x++) {
        rail[x].init(this, x);
    }
}

HubClass hub; /**< Hub Class */
Acroname::BrainStem::AppClass app[aUSBHUB3C_NUM_APPS]; /**< App Class */
Acroname::BrainStem::PointerClass pointer[aUSBHUB3C_NUM_POINTERS]; /**< Pointer_
↪Class */
Acroname::BrainStem::PowerDeliveryClass pd[aUSBHUB3C_NUM_USB_PORTS]; /**< Power_
↪Delivery Class */
Acroname::BrainStem::RailClass rail[aUSBHUB3C_NUM_RAILS]; /**< Rail Class */
Acroname::BrainStem::StoreClass store[aUSBHUB3C_NUM_STORES]; /**< Store Class */
Acroname::BrainStem::SystemClass system; /**< System Class */
Acroname::BrainStem::TemperatureClass temperature[aUSBHUB3C_NUM_TEMPERATURES]; /**
↪< Temperature Class */
Acroname::BrainStem::TimerClass timer[aUSBHUB3C_NUM_TIMERS]; /**< Timer Class */

/** Port ID */
typedef enum PORT_ID : uint8_t {
    kPORT_ID_0 = 0,
    kPORT_ID_1,
    kPORT_ID_2,
    kPORT_ID_3,
    kPORT_ID_4,
    kPORT_ID_5,
    kPORT_ID_CONTROL,
    kPORT_ID_POWER_C
} PORT_ID_t;
};

```

Python

```

class USBHub3c(Module):

    BASE_ADDRESS = 6
    NUMBER_OF_STORES = 3
    NUMBER_OF_INTERNAL_SLOTS = 12
    NUMBER_OF_RAM_SLOTS = 1
    NUMBER_OF_TEMPERATURES = 3
    NUMBER_OF_TIMERS = 8
    NUMBER_OF_APPS = 4
    NUMBER_OF_POINTERS = 4
    NUMBER_OF_USB_PORTS = 8
    NUMBER_OF_RAILS = 7
    NUMBER_OF_POWER_DELIVERY_PORTS = 8
    STORE_INTERNAL_INDEX = 0
    STORE_RAM_INDEX = 1
    STORE_EEPROM_INDEX = 2

    def __init__(self, address=BASE_ADDRESS, enable_auto_networking=True, model=defs.
↳MODEL_USBHUB_3C):
        super(USBHub3c, self).__init__(address, enable_auto_networking, model)
        self.system = System(self, 0)
        self.app = [App(self, i) for i in range(0, USBHub3c.NUMBER_OF_APPS)]
        self.pointer = [Pointer(self, i) for i in range(0, USBHub3c.NUMBER_OF_
↳POINTERS)]
        self.store = [Store(self, Store.INTERNAL_STORE),
                        Store(self, Store.RAM_STORE),
                        Store(self, Store.EEPROM_STORE)]
        self.temperature = [Temperature(self, i) for i in range(0, USBHub3c.NUMBER_OF_
↳TEMPERATURES)]
        self.timer = [Timer(self, i) for i in range(0, USBHub3c.NUMBER_OF_TIMERS)]
        self.hub = USBHub3c.Hub(self, 0)
        self.rail = [Rail(self, i) for i in range(0, USBHub3c.NUMBER_OF_RAILS)]
        self.pd = [PowerDelivery(self, i) for i in range(0, USBHub3c.NUMBER_OF_POWER_
↳DELIVERY_PORTS)]

    def connect(self, serial_number, **kwargs):
        return super(USBHub3c, self).connect(Spec.USB, serial_number)

    class Hub(USBSystem):
        def __init__(self, module, index):
            super(USBHub3c.Hub, self).__init__(module, index)
            self.port = [Port(module, i) for i in range(0, USBHub3c.NUMBER_OF_USB_
↳PORTS)]

```

In the code example above notice that the member variables app, pointer, pd, rail, store, system, temperature, and timer are all instances of “entity” classes. Once the USBHub3c is instantiated and the device connected the hardware functionality can be controlled via the entity interface.

C++

```

aUSBHub3c device;
device.discoverAndConnect(USB);
device.hub.port[0].setEnabled(1);

```

Python

```
device = aUSBHub3c();
device.discoverAndConnect(brainstem.link.Spec.USB);
device.hub.port[0].setEnabled(True);
```

Supported Entites

See the [Module Entities](#) section of the this document for a complete list of the entities supported by the USB-Hub3c.

Ports

Each of the 6 regular Type-C ports of the USBHub3c implement separate and independently switched USB2/3 data lines, CC, Vconn and current-limited Vbus lines. USB power, data and SS data can be independently disconnected for advanced USB testing applications. Control of various aspects of USB for each of the port is effected through two entities combined into a **Hub** member and Power Delivery control is effected through the **Power Delivery** entity.

USBHub3c Hub

There is a single “hub” instance within the module that controls the functionality primarily of the USBHub portion of the USBHub3c. It is made up of two separate entities. The [USB System](#) controls USB hub functionality as a whole, such as switching the upstream port setting enumeration delays and and controlling and monitoring multiple ports at a time. The [Port](#) entity provides fine grained control and monitoring of each port within the hub.

USBHub3c Power Delivery

There is a [Power Delivery](#) entity for each port on the USBHub3c. The [Power Delivery](#) entity controls much of the Power Delivery functionality of each of the Type-C ports.

USBHub3c System and Supporting entities

- [System](#): Device level functionality.
- [Rail](#): External rail controls supporting Loading on the Type-C ports.
- [Temperature](#): External rail controls supporting Loading on the Type-C ports.
- [Store](#): Access to non-volatile storage on the device.

Connections

Each USBHub3c is uniquely addressable and controllable from a host PC via the selected upstream port (0 by default) or through a dedicated Control Port. Acroname’s BrainStem™ link is then established over the USB input and allows a connection to the on-board controller in the USBHub3c. USBHub3c can be controlled via a host running BrainStem APIs

The USBHub3c is capable of many features. These features are organized into groups called [entities](#). Through these entities we can access the vast features of the USBHub3c.

A complete list of all entities and functions can be found at the bottom of the [page](#)

Establishing Software Control

Software control of the features of the USBHub3c is done with the BrainStem API via a BrainStem link. BrainStem links are done over USB and can be established via the currently selected upstream port (0-6) or the Control Port. After one or more of these ports is connected to a host machine, a user can connect to it via software API:

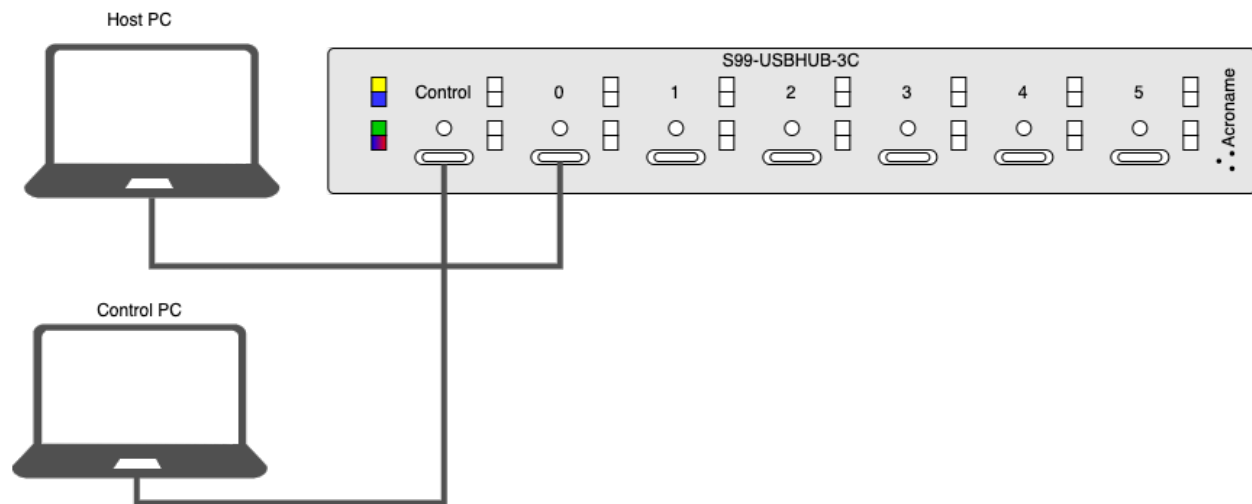
```
device.link.discoverAndConnect(USB)
```

When multiple Acroname devices are connected to a host, connecting to a specific hub can be done by providing the hub serial number. Further, all connected devices can be found using;

```
brainstem.discover.findAllModules(USB)` ` (Python)
Acroname::BrainStem::Link::sDiscover()` ` (C++)
```

BrainStem Control Port

The USBHub3c also has a dedicated control channel on the Type C connector labeled “Control”. This is a full-speed USB 2.0 connection for BrainStem interface only. No USB hub traffic can flow on this connection. When a cable is connected to the Type C connector, the BrainStem link can only be established through the Control Port, independent of the selected upstream port. Ports 0-6 are then used only for USB hub traffic to connect upstream and downstream USB devices. When the Control Port is not used, the BrainStem link will share the active upstream USB connection. Using the Control Port provides the ability to completely disconnect USB upstream host connections while maintaining software control of the hub.



Using Multiple Hosts with USBHub3c

The USBHub3c supports Acroname’s AnyPort™ technology. Any of the ports 0 - 6 can be selected as the hub’s upstream facing port. This functionality is accessed via the [USBSystem](#) entity. In order to seamlessly switch upstream ports, it is recommended that you use the device’s dedicated control port to connect to and control the USBHub3c.

Device Drivers

The USBHub3c leverages common operating system drivers that do not require custom installations on modern operating systems.

Some older operating systems may require the installation of a BrainStem USB driver information files to enable software control. Installation details on installing USB drivers can be found within the BrainStem Development Kit under the “drivers” folder. For example, Windows 7 requires the supplied INF to communicate with BrainStem USB devices.

1.2.5 USBHub3c Module Entities

I2C

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

I2C Control

The USBHub3c has the ability to send and receive I2C messages through the back expansion connector.

The I2C interface is controlled through the following APIs:

```
stem.i2c[x].read() \[cpp\] \[python\] \[NET\] \[CCA\]  
stem.i2c[x].write() \[cpp\] \[python\] \[NET\] \[CCA\]  
stem.i2c[x].setPullup() \[cpp\] \[python\] \[NET\] \[CCA\]  
stem.i2c[x].setSpeed() \[cpp\] \[python\] \[NET\] \[CCA\]  
stem.i2c[x].getSpeed() \[cpp\] \[python\] \[NET\] \[CCA\]
```

Port

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Port Entity provides control over the most basic items related to a USB Port. This includes actions ranging from a complete port enable and disable to the individual interface control. Voltage and current measurements are also included for devices which support the Port Entity.

Port Control

The USBHub3c has a Port Entity for every Type C port on the device; however, not all ports have the same capabilities. These ports can be referenced by their instance (port[x]) index.

Port Label	Index (port[x])
0	0
1	1
2	2
3	3
4	4
5	5
Control	6
Power C	7

One of the most powerful features of the USBHub3c is its ability to turn ports on and off which is available on Ports 0-5.

```
stem.hub.port[x].setEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB Vbus line for a single port can be done by calling the following method on Ports 0-5.

```
stem.hub.port[x].setPowerEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating Hi-Speed data and SuperSpeed data lines while not affecting the Vbus lines simultaneously for a single port can be done by calling the following method for Ports 0-5.

```
stem.hub.port[x].setDataEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB 2.0 Hi-Speed data lines for a single port can be done by calling the following for Ports 0-5.

```
stem.hub.port[x].setDataHSEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataHSEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Even further granularity can be achieved through Hi-Speed 1 and Hi-Speed 2 control methods for Ports 0-5.

```
stem.hub.port[x].setDataHS1Enabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataHS1Enabled() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].setDataHS2Enabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataHS2Enabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB 3.1 SuperSpeed data lines for a single port can be done by calling the following method for Ports 0-5.

```
stem.hub.port[x].setDataSSEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataSSEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Just as with the Hi-Speed lines the USBHub3c also has granular control of the SuperSpeed 1 and SuperSpeed 2 lines.

```
stem.hub.port[x].setDataSS1Enabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataSS1Enabled() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].setDataSS2Enabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataSS2Enabled() [cpp] [python] [NET] [CCA] [REST]
```

The CC lines can also be individually controlled.

```
stem.hub.port[x].setCCEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getCCEnabled() [cpp] [python] [NET] [CCA] [REST]
```

As you would expect at this point granular control is also provided.

```
stem.hub.port[x].setCC1Enabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getCC1Enabled() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].setCC2Enabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getCC2Enabled() [cpp] [python] [NET] [CCA] [REST]
```

Finally we come to Vconn control; however, this one overlaps with CC control because Vconn is what the unused CC line becomes (if needed). i.e. if CC1 is used for orientation and/or PD communication then CC2 will become Vconn (Vconn2) if it is enabled. If you are unaware of which pin is being used for Vconn you can simply call:

```
stem.hub.port[x].setVconnEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getVconnEnabled() [cpp] [python] [NET] [CCA] [REST]
```

and the USBHub3c will take care of the guess work for you. If you are aware of which line is being used for Vconn you can use the granular control just as we have outlined above.

```
stem.hub.port[x].setVconn1Enabled() [cpp] [python] [NET]:[CCA] [REST]
stem.hub.port[x].getVconn1Enabled() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].setVconn2Enabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getVconn2Enabled() [cpp] [python] [NET] [CCA] [REST]
```

Voltage and Current Measurements

The USBHub3c provides Voltage and Current measurements for both the Vbus and Vconn lines. These values can be acquired for all 8 ports through the following APIs

```
stem.hub.port[x].getVbusVoltage() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getVbusCurrent() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].getVconnVoltage() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getVconnCurrent() [cpp] [python] [NET] [CCA] [REST]
```

Power Modes

The ports of the USBHub3c are capable of providing power in multiple formats. The default is Power Delivery (PD), but that can be changed to things like: Standard Downstream Port (SDP), Charging Downstream Port (CDP) / Dedicated Charging Port (DCP), or even Qualcomm Quick Charge (QC) 3 and 4. These modes can be set through:

```
stem.hub.port[x].setPowerMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerMode() [cpp] [python] [NET] [CCA] [REST]
```

Power Mode	Value	Define
None	0	portPowerMode_none_Value
SDP	1	portPowerMode_sdp_Value
CDP/DCP	2	portPowerMode_cdp_dcp_Value
QC	3	portPowerMode_qc_Value
PD	4	portPowerMode_pd_Value
PS	5	portPowerMode_ps_Value
USB-C	6	portPowerMode_usbc_Value

Note: The Power Modes can only be changed when the port power is disabled.

Port Mode

As outlined in the “Port Control” section the USBHub3c can individually manipulate almost every pin on the Type-C connector; however, depending on your application that might require multiple function calls in order to configure the port how you want it. Port Mode on the other hand is a one stop shop that allows you to pick and choose which lines you want enabled or disabled through a single call. Additionally, it has a few other features tucked away inside of it.

```
stem.hub.port[x].setMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getMode() [cpp] [python] [NET] [CCA] [REST]
```

Port Mode Item	Bit	Value	Define
Power Enable	0	0/1	portPortMode_powerEnabled_Bit
HS 1 Enable	1	0/1	portPortMode_HS1Enabled_Bit
HS 2 Enable	2	0/1	portPortMode_HS2Enabled_Bit
SS 1 Enable	3	0/1	portPortMode_SS1Enabled_Bit
SS 2 Enable	4	0/1	portPortMode_SS2Enabled_Bit
CC 1 Enable	5	0/1	portPortMode_CC1Enabled_Bit
CC 2 Enable	6	0/1	portPortMode_CC2Enabled_Bit
Vconn 1 Enable	7	0/1	portPortMode_Vconn1Enabled_Bit
Vconn 2 Enable	8	0/1	portPortMode_Vconn2Enabled_Bit
Power Mode: Offset		16	portPortMode_portPowerMode_Offset
Power Mode: Mask		0x7	portPortMode_portPowerMode_Mask
Power Mode: None	16-18	0	portPortMode_portPowerMode_none_Value
Power Mode: SDP	16-18	1	portPortMode_portPowerMode_sdp_Value
Power Mode: CDP/DCP	16-18	2	portPortMode_cdp_dcp_Value
Power Mode: QC	16-18	3	portPortMode_portPowerMode_qc_Value
Power Mode: PD	16-18	4	portPortMode_portPowerMode_pd_Value
Power Mode: PS	16-18	5	portPortMode_portPowerMode_ps_Value
Power Mode: USB-C	16-18	6	portPortMode_portPowerMode_usbc_Value

Data Role

The data role describes the current configuration of the port in regards to its data direction. In most cases this evaluates to an Upstream Facing Port (UFP) or a Downstream Facing Port (DFP). Upstream in this case means the host side of the port and Downstream refers to the device side. The Data Role can be acquired through:

```
stem.hub.port[x].getDataRole() [cpp] [python] [NET] [CCA] [REST]
```

Data Role	Value	Define
Disabled	0	portDataRole_Disabled_Value
Upstream	1	portDataRole_Upstream_Value
Downstream	2	portDataRole_Downstream_Value
Control	3	portDataRole_Control_Value

Port Limits and Modes

At the Port level the user has the ability to define current limit and/or a power limit.

```
stem.hub.port[x].setCurrentLimit() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getCurrentLimit() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].setPowerLimit() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerLimit() [cpp] [python] [NET] [CCA] [REST]
```

If either of these values are exceeded then the USBHub3c will then apply on of the following modes

```
stem.hub.port[x].setCurrentLimitMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getCurrentLimitMode() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.port[x].setPowerLimitMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerLimitMode() [cpp] [python] [NET] [CCA] [REST]
```

Available Power

One of the unique features of the USBHub3c is its ability to manage input and output power. Because of smart charging technologies like PD we know exactly how much power we have access too. That input power must then be shared across all of the ports. The following function allows the user to request the amount of power that is currently allocated to the port in question.

```
stem.hub.port[x].getAvailablePower() [cpp] [python] [NET] [CCA] [REST]
```

Accumulated Power

The USBHub3c is capable of monitoring the accumulated power (energy) it has sank or sourced on both the VBus and VConn lines of each port. This value is presented as mWh.

```
stem.hub.port[x].getVbusAccumulatedPower() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getVconnAccumulatedPower() [cpp] [python] [NET] [CCA] [REST]
```

The accumulated power is set to zero and the accumulation period is restarted with these commands.

```
stem.hub.port[x].resetVbusAccumulatedPower() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].resetVconnAccumulatedPower() [cpp] [python] [NET] [CCA] [REST]
```

Downstream Data Speed

The USBHub3c can detect if a device has been enumerated. Additionally, it can detect at what speed a device has enumerated at.

```
stem.hub.port[x].getDataSpeed() [cpp] [python] [NET] [CCA] [REST]
```

Data Speed	Bit	Value	Define
1.5 Mbit/s	0	0/1	portDataSpeed_ls_1p5M_Bit
12 Mbit/s	1	0/1	portDataSpeed_fs_12M_Bit
480 Mbit/s	2	0/1	portDataSpeed_hs_480M_Bit
5 Gbit/s	3	0/1	portDataSpeed_ss_5G_Bit
10 Gbit/s	4	0/1	portDataSpeed_ss_10G_Bit
USB 2.0	6	0/1	portDataSpeed_Connected_2p0_Bit
USB 3.0	7	0/1	portDataSpeed_Connected_3p0_Bit

Power Delivery

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface. This Entity will allow you to directly access the vast landscape of PD.

When the capabilities of a PD system are fully realized everything in the system is “smart”. That includes the device, the host and even the cable. All of these elements contain electronics that identify themselves and what they are capable of doing. Because of this complexity it is important to align on a few terms that will be used throughout this Entity.

Partner This refers to the side of the PD connection in question. The possible options for this parameter are.

- **Local** Indicates the context/perspective of the Acraname device you are communicating with through a BrainStem connection.
- **Remote** The context/perspective of anything other than the Acraname device.

Partner Type	Value	Define
Local	0	powerdeliveryPartnerLocal
Remote	1	powerdeliveryPartnerRemote

Power Role Indicates the direction of power. This value is typically used in the context of a “Partner”. i.e. The remote partner is sinking, which would mean the local partner is sourcing. The possible options for this context are:

- **Sink** Indicates that the partner is taking power in/from.
- **Source** Indicates that the partner is providing power out/to.

Power Roles are also used in the context of what a port is capable of doing.

- **Sink** Device is capable of consuming power.

- **Source** Device is capable of producing power.
- **Sink/Source** Device is capable of both consuming or producing power. Dual Role Port (DRP)

Power Role	Value	Define
Disabled	0	powerdeliveryPowerRoleDisabled
Source	1	powerdeliveryPowerRoleSource
Sink	2	powerdeliveryPowerRoleSink
Source/Sink	3	powerdeliveryPowerRoleSourceSink

Power Data Objects (PDO)

- PDO's define what a device is capable of doing in the world of Power Delivery. PDO's are bit packed integers defined by the PD Specification which vary in meaning based on the type of PDO.

Request Data Objects (RDO)

- RDO's are the final agreement after successful Power Delivery negotiations. This RDO is always sent by the sinking device and is the result of the sources advertised PDO's and the needs/requirements of the sinking device. Only one RDO exists per valid connection.
-

Manipulating PDO's and RDO's

The Power Delivery specification defines a large number of Data Objects and the USBHub3c is capable of manipulating and modifying most of them in some fashion. The most common are PDO's and RDO's which were defined above. Direct manipulation is quite a complex process and is a feature of the Pro version only. It is highly recommended that users of these features first experiment with the Power Rule Editor within HubTool. Once you know the values you want to use manipulation can be done through:

```
stem.pd[x].setPowerDataObject() [cpp] [python] [NET] [CCA] [REST]
stem.pd[x].getPowerDataObject() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.pd[x].setRequestDataObject() [cpp] [python] [NET] [CCA] [REST]
stem.pd[x].getRequestDataObject() [cpp] [python] [NET] [CCA] [REST]
```

Power Roles Preferred

Preferred Power Role	Value	Define
None	0	pdPowerRolePreferred_None
Source	1	pdPowerRolePreferred_Source
Sink	2	pdPowerRolePreferred_Sink

Connection State

Connection State	Value	Define
None	0	pdConnectionState_None
Source	1	pdConnectionState_Source
Sink	2	pdConnectionState_Sink
Powered Cable	3	pdConnectionState_PoweredCable
Powered Cable with Sink	4	pdConnectionState_PoweredCableWithSink

Requests

Given the nature of Power Delivery there are only so many things that are within the direct control of the local USBHub3c. Many of the items on the remote side of the USBHub3c are merely request. In other words items in this category not guaranteed to happen.

```
stem.pd[x].request() [cpp] [python] [NET] [CCA] [REST]
```

Request	Value	Define
Hard Reset	0	pdRequestHardReset
Soft Reset	1	pdRequestSoftReset
Data Reset	2	pdRequestDataReset
Power Role Swap	3	pdRequestPowerRoleSwap
Power Fast Role Swap	4	pdRequestPowerFastRoleSwap
Data Role Swap	5	pdRequestDataRoleSwap
Vconn Swap	6	pdRequestVconnSwap
Sink Go to Min	7	pdRequestSinkGoToMinimum
Remote Source PDOs	8	pdRequestRemoteSourcePowerDataObjects
Remote Sink PDOs	9	pdRequestRemoteSinkPowerDataObjects
Remote Source Extended Capabilities	10	pdRequestRemoteSourceExtendedCapabilities
Remote Sink Extended Capabilities	11	pdRequestRemoteSinkExtendedCapabilities
Status	12	pdRequestStatus
PPS Status	13	pdRequestPPSStatus
Battery Capabilities	14	pdRequestBatteryCapabilities
Battery Status	15	pdRequestBatteryStatus
Manufacturer Info Sop	16	pdRequestManufacturerInfoSop
Manufacturer Info Sopp	17	pdRequestManufacturerInfoSopp
Manufacturer Inof Soppp	18	pdRequestManufacturerInfoSoppp
Discover Identity Sop	19	pdRequestDiscoverIdentitySop
Discover Identity Sopp	20	pdRequestDiscoverIdentitySopp
Discover Identity Soppp	21	pdRequestDiscoverIdentitySoppp
Revision	22	pdRequestRevision
Source Info	23	pdRequestSourceInfo
Country Code	24	pdRequestCountryCode
Country Info	25	pdRequestCountryInfo

Errors return from this function call only indicate the success of sending the request and do not reflect the suc-

cess of the actual request. To find the status of the request you can investigate the outcome of the connection or check the most recent status of the PD stack.

```
stem.pd[x].requestStatus() [cpp] [python] [NET] [CCA] [REST]
```

Cable Orientation

Although the Type C connector has no visible orientation the connector does have electrical orientation which directly correlates to the Communications Chanel (CC) strapping internal to the cable. The orientation be be obtained via:

```
stem.pd[x].getCableOrientation() [cpp] [python] [NET] [CCA] [REST]
```

Orientation	Value	Define
Invalid	0	pdCableOrientation_Invalid
CC1/A Side	1	pdCableOrientation_CC1
CC2/B Side	2	pdCableOrientation_CC2

Cable Type

Cable Type	Value	Define
Invalid	0	pdCableType_Invalid
Passive	1	pdCableType_Passive
Active	2	pdCableType_Active

Override

The USB C connector by default follows rules around maximum cable current and budgetted power. In some test applications, including ones with a Universal Orientation Cable, the port should ignore those rules which is why we have exposed override bits to allow for disabling of specific behavior. Below are the get/set routines for overrides and the bit definitions.

```
stem.pd[x].getOverride() [cpp] [python] [NET] [CCA] [REST]  
stem.pd[x].setOverride() [cpp] [python] [NET] [CCA] [REST]
```

Name	Bit	Definition
Cable Current	0	overrides the cable current limiting to 3A unless it's an emarked cable
Port Power	1	Overrides the port power budgetting and just allows full power always
Auto Discovery	2	Overrides the auto discovery feature. With override true the hub will only establish a basic power connection and wont ask for additional vendor information.
Data path disable	3	Override normal disabling of the data path when the PD Data Role and port configured data role differ

Rail

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Rail entity provides power control to connected devices on some modules. Check the module datasheet to determine if the module has this capability.

the Rail entity controls power provided to downstream devices, it has the ability to enable and disable power, can read voltage on the rail, and provides current consumption information on some modules. There are additional capabilities that certain modules provide which enhance basic power delivery through Kelvin sensing, or by bringing online separate power management functionality.

Certain modules may provide more than one power rail. These are independently controlled and can be accessed via the entity index.

Rail Control

The USBHub3c has the ability to redirect power from ports 0-5 to an external connection. This applies to both sinking and sourcing and allows load testing of connected devices. Additionally there is a 5V rail that can be used as a trigger or even powering external devices.

Rail	Index: rail[x]
Port 0	0
Port 1	1
Port 2	2
Port 3	3
Port 4	4
Port 5	5
5 Volt	6

Rails are controlled through the following APIs:

```
stem.rail[x].setEnabled() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
stem.rail[x].getEnable() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every USBHub3c is assigned a unique serial number at the factory. This facilitates an arbitrary number of USBHub3c devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Hardware Version

Get the module's hardware version information.

```
stem.system.getHardwareVersion(hardwareVersion) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The hardware version is a 32-bit value that encodes hardware revision and variant information. The format is:

Table 5: Hardware Version Bit Breakdown

Bits	Field	Description
24-31	Revision	Hardware revision identifier
16-23	Reserved	Reserved (always 0)
8-15	Variant	Hardware variant identifier
0-7	Reserved	Reserved (always 0)

Table 6: Hardware Revision Values

Value	Description
'B'	Revision B
'C'	Revision C
'D'	Revision D
'E'	Revision E

Table 7: Hardware Variant Values

Value	Description
'P'	PRO
'L'	LAB

The hardware revision and variant are determined by reading hardware strapping pins during initialization to identify the specific board revision and model configuration.

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the USBHub3+ away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

System Power

The USBHub3c is designed to accept power from either a Type-C PD port or from the unregulated power connector.

In the case of a Type-C PD source all power can be accounted for through Power Delivery (PD) negotiations. For instance, if a PD source advertises 100 Watts; that means we can provide 100 Watts of power to the connected sinking devices. The same can be said for a 60 Watt PD source. This value represents the maximum amount of power the USBHub3c can budget for all of its sinking devices. This value can be acquired through:

```
stem.system.getPowerLimit() [cpp] [python] [NET] [CCA] [REST]
```

It is possible that the limit of your PD source may not match that of what is being advertised by the function above. The contributing factors can be determined by checking the state of the power limit.

```
stem.system.getPowerLimitState() [cpp] [python] [NET] [CCA] [REST]
```

In the case of the unregulated power connector the power budgeting gets a bit more tricky because there is no way of knowing the supplies maximum power. To resolve this the user is allowed to define a power limit maximum that matches that of the connected power supply.

```
stem.system.setPowerLimitMax() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.getPowerLimitMax() [cpp] [python] [NET] [CCA] [REST]
```

Note: Failure to correctly configure this value can result in undefined behavior such as resets.

These values only apply when the input power source is that of the unregulated power connector.

```
stem.system.getInputPowerSource() [cpp] [python] [NET] [CCA] [REST]
```

Power Budgeting and Behavior

As we alluded to in the System power section the USBHub3c has to be a bit clever about accounting for all input and output power. The method by which an input power source is selected and used is referred to as Input Power Behavior. It can be configured through

```
stem.system.setInputPowerBehavior() [cpp] [python] [NET] [CCA] [REST]
stem.system.getInputPowerBehavior() [cpp] [python] [NET] [CCA] [REST]
```

Some behaviors require additional configuration. In most cases this is a list of port numbers that define which ports should be prioritized for input power.

```
stem.system.setInputPowerBehaviorConfig() [cpp] [python] [NET] [CCA] [REST]
stem.system.getInputPowerBehaviorConfig() [cpp] [python] [NET] [CCA] [REST]
```

To determine which power source is currently active and providing power to the system, use:

```
stem.system.getInputPowerSource() [cpp] [python] [NET] [CCA] [REST]
```

This command returns the port index of the active input power source. Port indices 0-7 correspond to the USB-C ports (ports 0-7), while an index of 8 indicates the unregulated power connector (VUNREG) input.

Voltage and Current Monitoring

The USBHub3c provides monitoring capabilities for the system's input voltage and current. These measurements allow users to track the power consumption and characteristics of the input power source.

Input Voltage

The module's input voltage can be retrieved using:

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

This command returns the input supply voltage of the USBHub3c in microvolts (μV). The voltage corresponds to the active input power source, which can be determined using `getInputPowerSource()` (see *Power Budgeting and Behavior*).

Input Current

The module's input current can be retrieved using:

```
stem.system.getInputCurrent() [cpp] [python] [NET] [CCA] [REST]
```

This command returns the input current being drawn from the power source in microamps (μA). The current corresponds to the active input power source, which can be determined using `getInputPowerSource()` (see *Power Budgeting and Behavior*).

Temperature

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Certain modules have a temperature measurement available. The temperature entity gives access to these measurements. Check your module datasheet to see if your module has a temperature entity.

System Temperature

The temperature of the USBHub3c can be measured with:

```
stem.temperature[0].getValue( $\mu\text{C}$ ) [cpp] [python] [NET] [CCA] [REST]
```

where temperature is in micro-degrees Celcius.

The USBHub3c has 3 temperature sensors:

Index	Sensor
0	Ambient temperature sensor
1	Buck side temperature sensor
2	Boost side temperature sensor

UART

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The UART entity is a class which allows a user to configure and control an arbitrary stream of serial data. These streams can represent a number of different transports, including a RS232 external interface, a virtual COM port, and onboard UART interfaces from system components. This entity allows each transport to be configured as either an endpoint, or as a passthrough between transports, similar to a switchboard of a telephone operator.

Extron Control

The USBHub3c has the ability to be controlled through an RS232 interface on the external expansion connector of the USBHub3c. The USBHub3c also supports VCOM (Virtual COM port) channels for USB-based serial communication. For additional information about the serial protocol, reference *USBHub3c Serial Communication Feature*

UART Entity Indices

The USBHub3c provides the following UART entity indices:

Table 8: UART Entity Indices

Index	Description
0	Hardware RS232 (External Expansion Connector)
1	VCOM_0 (Virtual COM port channel)

For additional information about the serial protocol, reference *USBHub3c Serial Communication Feature*

UART APIs

The USBHub3c has four protocol values enumerated.

Value	Description
0	Disabled/Undefined
1	Extron Compatible Protocol
2	Brainstem Transport
6	Loopback

Uarts are controlled through the following APIs:

```
stem.uart[x].setEnabled() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]  
stem.uart[x].getEnable() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]  
stem.uart[x].setBaudRate() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]  
stem.uart[x].getBaudRate() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]  
stem.uart[x].setProtocol() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```



```
stem.uart[x].getProtocol() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].setLinkChannel() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].getLinkChannel() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].setStopBits() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].getStopBits() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].setParity() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].getParity() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].setDataBits() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].getDataBits() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].getCapableProtocols() [cpp] [python] [NET] [CCA] [REST]
stem.uart[x].getAvailableProtocols() [cpp] [python] [NET] [CCA] [REST]
```

USB System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USBSystem class provides high level control of the lower level *Port Entity*

Upstream Control

The USBHub3c has the unique ability to designate any of its full featured (0-5) ports as the upstream connection. This is very useful for moving devices between hosts or testing dual role port functionality.

```
stem.hub.setUpstream() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getUpstream() [cpp] [python] [NET] [CCA] [REST]
```

Enumeration Delay

Once a USB device is detected by the USBHub3c it is possible to delay its connection to an upstream host computer and subsequent enumeration on the USB bus. The enumeration delay can mitigate or eliminate host kernel instabilities by forcing devices to enumerate in slow succession, allowing a focus on validation of drivers and software. The enumeration delay is configured in milliseconds, representing the time delay between enabling each successive port. Enumeration delay is applied when the hub powers on or when a new upstream connection is made.

```
stem.hub.setEnumerationDelay() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getEnumerationDelay() [cpp] [python] [NET] [CCA] [REST]
```

Power Behavior

Ports 0-5 of the USBHub3c are all capable of sourcing 100 watts pending the system has access to that amount of power. In most cases 500 Watts is not available and therefore the system has to be clever about how it allocates power. The method in which power is allocated across these ports is called the power behavior and it can be configured through

```
stem.hub.setPowerBehavior() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getPowerBehavior() [cpp] [python] [NET] [CCA] [REST]
```

Some behaviors require additional configuration. In most cases this is a list of port numbers that define which ports should be prioritized for power allocation.

```
stem.hub.setPowerBehaviorConfig() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getPowerBehaviorConfig() [cpp] [python] [NET] [CCA] [REST]
```

Data Behavior

Many devices are now capable of being Dual Role Ports (DRP) meaning that they can be both a device (downstream) and a host (upstream). These devices can request to become a host at anytime which may or may not contradict the users desired upstream setting. The method in which these events are handled is referred to as data behavior. Just as power behavior it can be configured with a similar set of APIs

List of Available Data Behaviors for USBHub3c

Behavior	Value	Define
Hard Coded	0	usbsystemDataBehavior_HardCoded
Reserved	1	usbsystemDataBehavior_Reserved
Port Priority	2	usbsystemDataBehavior_PortPriority

Hard Coded (Default Configuration)

The Hard Coded data behavior is used to fix the Upstream port to a single port and not allow it to move except for a command through the [Set Upstream](#) API or via the *Serial Communication Feature*.

Port Priority

The Port Priority data behavior prioritizes making the Upstream port the lowest numbered port on the front of the USBHub3c that is capable of being an Upstream port. This means a USB-C connection that's a sink or a USB PD connection that has described itself as USB Coms Capable and can act as a Host.

Relevant API's

```
stem.hub.setDataRoleBehavior() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getDataRoleBehavior() [cpp] [python] [NET] [CCA] [REST]
```

Some behaviors require additional configuration. In most cases this is a list of port numbers that define which ports should be prioritised for power allocation.

```
stem.hub.setDataRoleBehaviorConfig() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getDataRoleBehaviorConfig() [cpp] [python] [NET] [CCA] [REST]
```

Override

The USB system by default follows compliant behavior around Vbus toggling and upstream detection. In some test applications you may want to override specific behaviors which is why we have exposed override bits. Below are the get/set routines for overrides and the bit definitions.

```
stem.hub.getOverride() [cpp] [python] [NET] [CCA] [REST]
stem.hub.setOverride() [cpp] [python] [NET] [CCA] [REST]
```

Name	Bit	Definition
Auto Vbus Toggle Disable	0	Disables the auto vbus toggle behavior on re-enumeration of the upstream port
Vbus Detect Disable	1	Disables the requirement of an upstream connection for enabling the hub chip

High Level Control of the Port Entity

The USBSystem Entity and the *Port Entity* are capable of doing many of the same things its merely their perspective. The PortClass acts on individual elements where as the USBSystemClass acts on all of the ports. For instance if you wanted to enable all of the ports of the USBHub3c you would need to loop through each index and individually enable each port. With the USBSystem class you can do the exact same thing, but with a single API call with each bit representing a given port.

```
//USBSystem Entity method.
stem.hub.setEnabledList(0x3F); //0b0011 1111 bits 0-5 set high = ports 0-5

//Port Entity method.
for(int x = 0; x <= 5; x++) {
    stem.hub.port[x].setEnabled(true);
}
```

```
stem.hub.setEnabledList() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.setEnabledList() [cpp] [python] [NET] [CCA] [REST]
```

The same logic can also be applied to Data Role, Mode and State elements, but with slightly different interfaces depending on the size of the data.

```
stem.hub.setModeList() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.getModeList() [cpp] [python] [NET] [CCA] [REST]
```

Data Role and State are slightly different in that there are only get calls for these functions.

```
stem.hub.getDataRoleList() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.getStateList() [cpp] [python] [NET] [CCA] [REST]
```

Legacy Support

Previous Acroname USB products made use of the *USB Entity* for measurements, configuration and control; however, the USBHub3c aims to organize these capabilities in a cleaner fashion through the use of the all new *Port* and *USBSystem* Entities.

In efforts to ease this transition we have added Legacy support to the USBHub3c by implementing limited USB Entity functionality. The following functions of the *USB Entity* will still behave as they did in previous Acroname USB products.

Port

```
stem.usb.setPortEnable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getPortState(channel) [cpp] [python] [NET] [CCA] [REST]
```

Port State Item	Bit	Value
VBus is enabled	6	0/1
USB2A is enabled	4	0/1
USB2B is enabled	5	0/1
SuperSpeedA is enabled	7	0/1
SuperSpeedB is enabled	8	0/1
CC1 is enabled	12	0/1
CC2 is enabled	13	0/1

Power

```
stem.usb.setPowerEnable(channel) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setPowerDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

Voltage

```
stem.usb.getPortVoltage(channel,  $\mu$ V) [cpp] [python] [NET] [CCA] [REST]
```

Current

```
stem.usb.getPortCurrent(channel,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
stem.usb.getPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

Data

```
stem.usb.setDataEnable(channel) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setDataDisable(channel) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setHiSpeedDataEnable(channel) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setHiSpeedDataDisable(channel) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setSuperSpeedDataEnable(channel) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setSuperSpeedDataDisable(channel) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setCC1Enable(channel, enable) [cpp] [python] [NET] [CCA] [REST]
stem.usb.getCC1Enable(channel, enable) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setCC2Enable(channel, enable) [cpp] [python] [NET] [CCA] [REST]
stem.usb.getCC2Enable(channel, enable) [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
store[0-2]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	getSlotCapacity	
	getSlotSize	
	setSlotLocked	
	getSlotLocked	
system[0]	getModule	

continues on next page

Table 9 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getModuleBaseAddress	
	setRouter	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	getVersion	
	getModel	
	getHardwareVersion	
	getBuild	
	getLinkInterface	
	getSerialNumber	
	save	
	reset	
	logEvents	
	getUptime	
	getTemperature	
	getMinimumTemperature	
	getMaximumTemperature	
	getInputPowerSource	
	getInputVoltage	
	getInputCurrent	
	getModuleHardwareOffset	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	routeToMe	
	getUnregulatedVoltage	
	getUnregulatedCurrent	
	getPowerLimit	
	getPowerLimitMax	
	setPowerLimitMax	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
temperature[0-2]	getValue	
	getValueMax	
	getValueMin	
	resetLoggedValues	
port[0-7]	getVbusVoltage	
	getVbusCurrent	
	getVconnVoltage	
	getVconnCurrent	
	getPowerEnabled	
	setPowerEnabled	
	getPowerMode	
	setPowerMode	
	getEnabled	
	setEnabled	
	getDataEnabled	

continues on next page

Table 9 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getDataHS1Enabled	
	setDataHS1Enabled	
	getDataHS2Enabled	
	setDataHS2Enabled	
	getDataSSEnabled	
	setDataSSEnabled	
	getDataSS1Enabled	
	setDataSS1Enabled	
	getDataSS2Enabled	
	setDataSS2Enabled	
	getVconnEnabled	
	setVconnEnabled	
	getVconn1Enabled	
	setVconn1Enabled	
	getVconn2Enabled	
	setVconn2Enabled	
	getDataRole	
	getDataSpeed	
	getCCEnabled	
	setCCEnabled	
	getCC1Enabled	
	setCC1Enabled	
	getCC2Enabled	
	setCC2Enabled	
	getCCBias	
	setCCBias	
	getMode	
	setMode	
	getState	
	getName	
	setName	
	getCurrentLimit	
	setCurrentLimit	
	getAllocatedPower	
	getAvailablePower	
	getPowerLimit	
	setPowerLimit	
	getVbusAccumulatedPower	
	resetVbusAccumulatedPower	
	getVconnAccumulatedPower	
	resetVconnAccumulatedPower	
	getHSBoost	
	setHSBoost	
	getDataHSRoutingBehavior	
	setDataHSRoutingBehavior	
	getDataSSRoutingBehavior	
	setDataSSRoutingBehavior	

continues on next page

Table 9 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
USBSystem [0]	getCC1State	
	getCC2State	
	getCCCCurrentLimit	
	setCCCCurrentLimit	
	getVoltageSetpoint	
	setVoltageSetpoint	
	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
	getDataHSMaxDatarate	
	getDataSSMaxDatarate	
	getOverride	
	getSelectorMode	
	getUpstreamHS	
	getUpstreamSS	
	setDataHSMaxDatarate	
	setDataSSMaxDatarate	
	setOverride	
	setSelectorMode	
	setUpstreamHS	
	setUpstreamSS	
PowerDelivery [0-8]	getConnectionState	
	getNumberOfPowerDataObjects	
	setPowerDataObject	
	getPowerDataObject	
	resetPowerDataObjectToDefault	
	getPowerDataObjectList	
	setPowerDataObjectEnabled	
	getPowerDataObjectEnabled	
	getPowerDataObjectEnabledList	
	setRequestDataObject	
	getRequestDataObject	
	getPowerRole	
	setPowerRole	
	getPowerRolePreferred	
	setPowerRolePreferred	
	getCableVoltageMax	
	getCableCurrentMax	
	getCableSpeedMax	
	getCableType	
	getCableOrientation	
	setOverrides	
	getOverrides	
	request	
	setCurrentLimitBehavior	
	getCurrentLimitBehavior	
	getPeakCurrentConfiguration	
	setPeakCurrentConfiguration	
	getFastRoleSwapCurrent	

continues on next page

Table 9 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
UART[0-1]	setFastRoleSwapCurrent	
	getAttachTimeElapsed	
	getDataRoleCapabilities	
	getFlagMode	
	getLinkState	
	getPowerRoleCapabilities	
	getProtocolRevisionMax	
	setEnabled	
	setProtocolRevisionMax	
	vDM	
	resetEntityToFactoryDefaults	
	setEnabled	
	getEnabled	
	setBaudRate	
	getBaudRate	
	setProtocol	
	getProtocol	
	setLinkChannel	
	getLinkChannel	
	setStopBits	
	getStopBits	
rail[0-6]	setEnabled	
	getEnabled	
i2c[0]	read	
	write	
	setPullup	
	setSpeed	
usb[0]	getSpeed	
	getCC1Enable	
	getCC2Enable	
	getPortCurrent	
	getPortCurrentLimit	
	getPortState	
	getPortVoltage	
	setPortEnable	Ports 0-5
	setPortDisable	Ports 0-5
	setDataEnable	Ports 0-5
	setDataDisable	Ports 0-5
	setHiSpeedDataEnable	Ports 0-5
	setHiSpeedDataDisable	Ports 0-5
	setSuperSpeedDataEnable	Ports 0-5
	setSuperSpeedDataDisable	Ports 0-5

continues on next page

Table 9 – continued from previous page

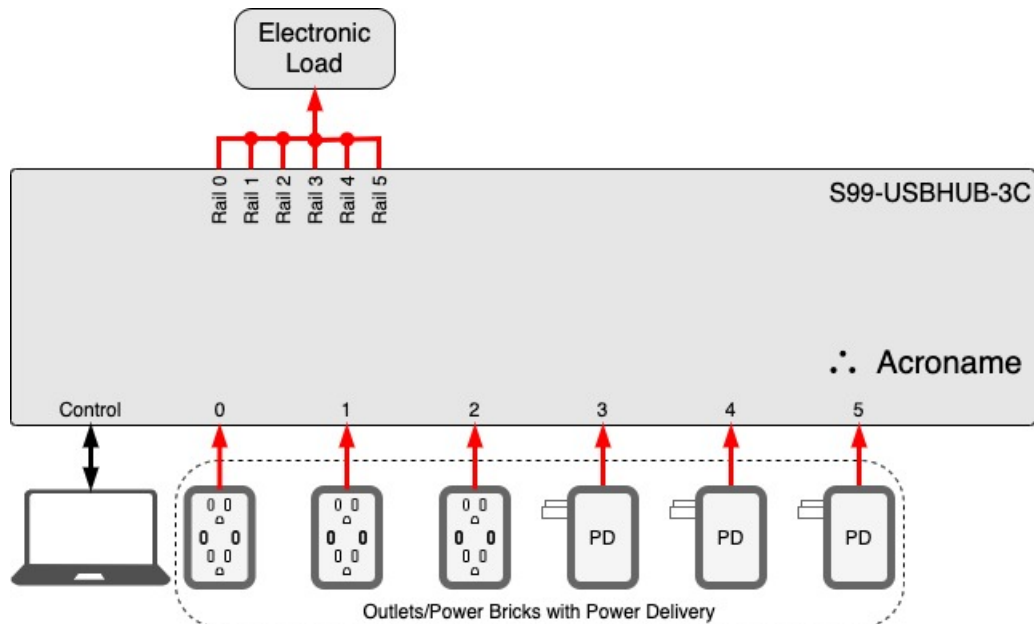
Entity Class	Entity Option	Variable(s) Notes
	setPowerEnable	Ports 0-5
	setPowerDisable	Ports 0-5
	setCC1Enable	Ports 0-5
	getCC1Enable	Ports 0-5
	setCC2Enable	Ports 0-5
	getCC2Enable	Ports 0-5
	getPortVoltage	Ports 0-5
	getPortCurrent	Ports 0-5
	getPortCurrentLimit	Ports 0-5
	setPortCurrentLimit	Ports 0-5

1.2.6 USBHub3c Software Features

Note: Some software features may require a license update or purchase from Acroname. See [Updating Software Licenses](#) for information on how to update your device's licenses.

External Load Testing

The External Load software feature allows for load testing of devices connected to ports 0-5 by way of an external connector on the back of the hub. This connector allows you to wire the USBHub3c to programmable or resistive loads so that you can test if your device is sourcing power properly.



This software feature can be exploited through the *USBHub3c Rail Entity*.

Although the external load feature will work in any mode it is only recommended to be used when the USBHub3c is sinking.

Example:**C++**

```

static const int TEST_PORT = 1;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

// Check if we are sourcing power
uint8_t connectionState = 0;
stem.pd[TEST_PORT].getConnectionState(&connectionState);

// Ensure we are sinking
if (connectionState == powerdeliveryPowerRoleSink) {
    stem.rail[TEST_PORT].setEnabled(true);

    // Do Stuff
    int32_t voltage = 0;
    int32_t current = 0;
    stem.hub.port[TEST_PORT].getVbusVoltage(&voltage);
    stem.hub.port[TEST_PORT].getVbusCurrent(&current);
    // Do Stuff

    stem.rail[TEST_PORT].setEnabled(false);
}

stem.disconnect();

```

Python

```

TEST_PORT = 1;

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB);

# Check if we are sourcing power
connection_state_result = stem.pd[TEST_PORT].getConnectionState();

# Ensure we are sinking
if ((connection_state_result.value == _BS_C.powerdeliveryPowerRoleSink):
    stem.rail[TEST_PORT].setEnabled(true)

    # Do Stuff
    voltage_result = stem.hub.port[TEST_PORT].getVbusVoltage();
    current_result = stem.hub.port[TEST_PORT].getVbusCurrent();
    # Do Stuff

    stem.rail[TEST_PORT].setEnabled(false)

stem.disconnect()

```

Relevant API's

```
stem.rail[x].setEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.rail[x].getEnabled() [cpp] [python] [NET] [CCA] [REST]
```

PD Builder

The PD Builder feature allows the user to modify all *Power Data Objects (PDO)* presented by the USBHub3c. This includes both sourcing and sinking PDOs of the USBHub3c.

Use Cases



- Designing PDOs for you own device
- Exposing DUTs to PDOs
- Restricting DUTs from PDOs









Editable Items

PDO Types	PDO Flags	PDO Limits
Fixed	Unchucked message support	Voltage
Variable	Dual role data	Current
Battery	USB communications possible	Power
APDO	External power	
	USB suspend	
	Dual role power	
	High capability	
	Fast role swap current	

HubTool

HubTool allows the user to visually build a PDO without needing to know anything about the Power Delivery specification. Once created you can immediately apply it to the USBHub3c.

Port: 1  Power Type: Local Source Rules 

En	Rule	PDO Type	Min Voltage (mV)	Max Voltage (mV)	Power (mW)	Current (mA)	Raw	Set Rule	Set Default
	1	Fixed	5000	5000		3000	0x3F01912C	Set	Reset
	2	Fixed	9000	9000		3000	0x0002D12C	Set	Reset
	3	Variable	3300	21000		5000	0x9A4109F4	Set	Reset
	4	Battery	<div>3400</div>	21000	100000		0x5A410990	Set	Reset
	5	ADPO	3000	21000		5000	0xC1A41E64	Set	Reset
	6	Fixed	0	0		0	0x00000000	Set	Reset
	7	Fixed	0	0		0	0x00000000	Set	Reset

Example

C++

```
static const int TEST_PORT = 1;
static const uint32_t MY_CUSTOM_SOURCE_PDO = 0x0001912C;
static const uint32_t MY_CUSTOM_SINK_PDO = 0x0002D12C;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

//Change local SOURCE PDO 1 to MY_CUSTOM_SOURCE_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSource, 1, MY_CUSTOM_
↳SOURCE_PDO)

//Change local SINK PDO 1 to MY_CUSTOM_SINK_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSink, 1, MY_CUSTOM_SINK_
↳PDO)

//Do Stuff

stem.disconnect()
```

Python

```
TEST_PORT = 1;
MY_CUSTOM_SOURCE_PDO = 0x0001912C;
MY_CUSTOM_SINK_PDO = 0x0002D12C;

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

#Change local SOURCE PDO 1 to MY_CUSTOM_SOURCE_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSource, 1, MY_CUSTOM_
↳SOURCE_PDO)

#Change local SINK PDO 1 to MY_CUSTOM_SINK_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSink, 1, MY_CUSTOM_SINK_
↳PDO)

#Do Stuff

stem.disconnect()
```

Relevant API's

```
stem.hub.pd[x].setPowerDataObject() [cpp] [python] [NET] [CCA] [REST]
stem.hub.pd[x].getPowerDataObject() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.hub.pd[x].setPowerDataObjectEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.pd[x].getPowerDataObjectEnabled() [cpp] [python] [NET] [CCA] [REST]
```

PD Logging

The PD Logging feature allows you to monitor Power Delivery communication on all 8 ports of the USBHub3c.

Use Cases

- Capture Power Delivery (PD) events across all USB-C ports
- Decode PD messages
- Export PD message logs to CSV
- Filter USB-PD traffic by message type.
- Clearly show PD message direction.
- Send arbitrary Vendor Defined Messages (VDMs)

HubTool

With HubTool you can easily visualize the Power Delivery log.

The screenshot displays the HubTool application window. At the top, 'System Information' shows SN: 0x4F7A0C15, Model: 24, Firmware: 2.9.6, Module: 6, Voltage: 20.0 VDC, Current: 0.5 A, Temperature: 30 C, Hardware Offset: 0, Router: 6, and SW Offset: 0. Below this are tabs for Summary, Port 0, Port 2, Port 3, Port 4, Port 5, Control, Power C, IO Expander, Power, and PD Logging. The PD Logging tab is active, showing a table of logs with columns: Timestamp (S:uS), Port, Direction, Spec, SOP, Power Role, Data Role, ID, Packet Type, Msg Type, and Raw. The table contains 25 entries of PD messages. At the bottom, there are sections for 'Software Feature' (Rail Enable, QC Mode, PDO Editing, VBUS Validation, PD Logging all Enabled) and 'Device Type' (USBHub3c, USBHub3p).

	Timestamp (S:uS)	Port	Direction	Spec	SOP	Power Role	Data Role	ID	Packet Type	Msg Type	Raw
1	1429:635630	1	RX	V2.0	SOP	Source	DFP	0	Data	Source Capabilities	0x61 0x11 0x96 0x90 0x01 0x36
2	1429:645760	1	TX	V2.0	SOP	Sink	UFP	0	Data	Request	0x42 0x10 0x64 0x90 0x01 0x10
3	1429:654670	1	RX	V2.0	SOP	Source	DFP	1	Control	Accept	0x63 0x03
4	1429:680910	1	RX	V2.0	SOP	Source	DFP	2	Control	PS Ready	0x66 0x05
5	1429:690560	1	RX	V2.0	SOP	Source	DFP	3	Data	Vendor Defined	0x6F 0x17 0x01 0x80 0x00 0xFF
6	1429:701040	1	TX	V2.0	SOP	Sink	UFP	1	Data	Vendor Defined	0x4F 0x72 0x41 0x80 0x00 0xFF 0xFF 0x24 0xA...
7	1429:707240	1	RX	V2.0	SOP	Source	DFP	4	Data	Vendor Defined	0x6F 0x19 0x02 0x80 0x00 0xFF
8	1429:712110	1	TX	V2.0	SOP	Sink	UFP	2	Data	Vendor Defined	0x4F 0x24 0x42 0x80 0x00 0xFF 0x00 0x00 0xF...
9	1429:717770	1	RX	V2.0	SOP	Source	DFP	5	Data	Vendor Defined	0x6F 0x1B 0x02 0x80 0x00 0xFF
10	1429:724220	1	TX	V2.0	SOP	Sink	UFP	3	Data	Vendor Defined	0x4F 0x26 0x42 0x80 0x00 0xFF 0x00 0x00 0xF...
11	1429:948940	1	TX	V2.0	SOP	Sink	UFP	4	Control	VConn Swap	0x4B 0x08
12	1429:957240	1	RX	V2.0	SOP	Source	DFP	6	Control	Accept	0x63 0x0D
13	1430:18980	1	TX	V2.0	SOP	Sink	UFP	5	Control	PS Ready	0x46 0x0A
14	1430:72450	1	TX	V2.0	S...	Sink	UFP	0	Data	Vendor Defined	0x4F 0x10 0x01 0x80 0x00 0xFF
15	1430:158980	1	TX	V2.0	S...	Sink	UFP	1	Data	Vendor Defined	0x4F 0x12 0x01 0x80 0x00 0xFF
16	1430:168330	1	RX	V2.0	S...	Source	UFP	1	Data	Vendor Defined	0x4F 0x53 0x41 0x80 0x00 0xFF 0x11 0x07 0x00...
17	1430:262130	1	TX	V2.0	SOP	Sink	UFP	6	Control	DR Swap	0x49 0x0C
18	1430:269210	1	RX	V2.0	SOP	Source	DFP	7	Control	Accept	0x63 0x0F
19	1430:515440	1	TX	V2.0	SOP	Sink	DFP	7	Control	PR Swap	0x6A 0x0E
20	1430:523230	1	RX	V2.0	SOP	Source	UFP	0	Control	Accept	0x43 0x01
21	1430:625040	1	RX	V2.0	SOP	Sink	UFP	1	Control	PS Ready	0x46 0x02
22	1430:741540	1	TX	V2.0	SOP	Source	DFP	0	Control	PS Ready	0x66 0x01
23	1430:903350	1	TX	V2.0	SOP	Source	DFP	0	Data	Source Capabilities	0x61 0x51 0x2C 0x91 0x01 0x3E 0x2C 0xD1 0x02...
24	1430:915290	1	RX	V2.0	SOP	Sink	UFP	0	Data	Request	0x42 0x10 0x2C 0xB1 0x04 0x13
25	1430:919090	1	TX	V2.0	SOP	Source	DFP	1	Control	Accept	0x63 0x03

Software Feature: Rail Enable: Enabled
 Software Feature: QC Mode: Enabled
 Software Feature: PDO Editing: Enabled
 Software Feature: VBUS Validation: Enabled
 Software Feature: PD Logging: Enabled
 USBHub3c: 0x4F7A0C15, Error: 7 CMD: stem.hub.port[x].setVoltageSetpoint

Device Type: USBHub3c
 Serial Number: 4F7A0C15
 USBHub3p AF62130D

Relevant API's

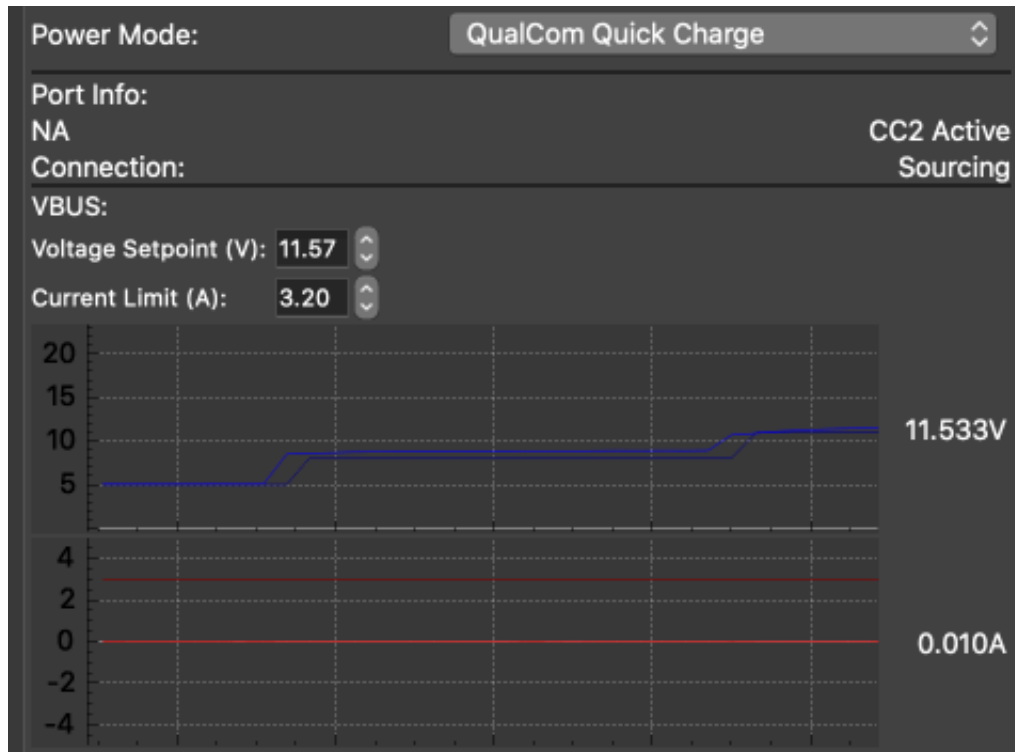
```
stem.pd[x].setUEIBytes() [cpp] [python] [NET] [CCA]
```

Quick Charge™

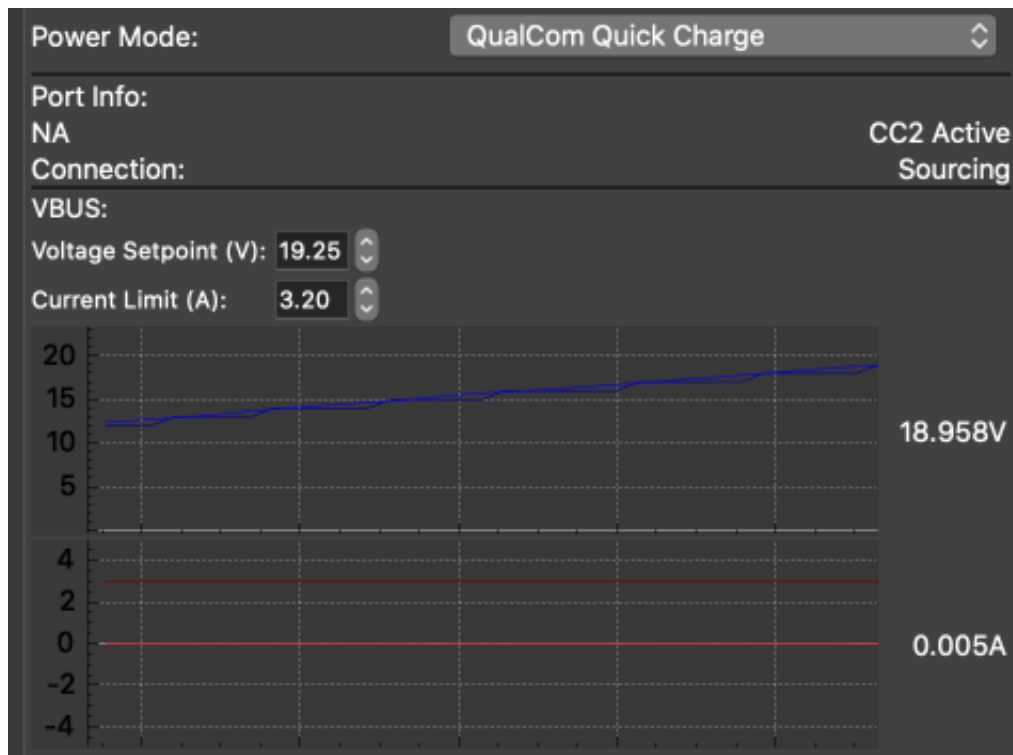
The USBHub3c Quick Charge™ feature adds the ability to enable the Quick Charge™ port power mode on one or more ports. This enables fast charging on devices that support Quick Charge™.

Version	Voltage	Current	Power
QC1	0-6.3V	2A	10W
QC2	Class A: 5V, 9V, 12V Class B: 5V, 9V, 12V, 20V	1.67A, 2A, or 3A	18W
QC3	3.6-22V in 200mV steps	2.6A or 4.6A	36W
QC4	3.6-20V in 20mV steps 5V, 9V (PD compatible) 3-21V in 20mV steps (PD3 PPS mode)	2.6A or 4.6A 3A (PD modes)	100W

HubTool - QC 2.0



HubTool - QC 3.0



This software feature can be exploited through the *USBHub3c Port Entity*

Example

C++

```
static const int TEST_PORT = 1;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

//Configure the Power mode: Quick Charge™
stem.hub.port[TEST_PORT].setPowerMode(portPowerMode_qc_Value);

//Do Stuff

stem.disconnect();
```

Python

```
TEST_PORT = 1;

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

## Configure the Power mode: Quick Charge™
stem.hub.port[TEST_PORT].setPowerMode(_BS_C.portPowerMode_qc_Value);

#Do Stuff
```

(continues on next page)

(continued from previous page)

```
stem.disconnect()
```

Relevant API's

```
stem.hub.port[x].setPowerMode() [cpp] [python] [NET] [CCA] [REST] stem.hub.  
port[x].getPowerMode() [cpp] [python] [NET] [CCA] [REST]
```

VBus Validation

The VBus Validation software feature gives the user full control of current limit and voltage setpoint for ports 0-5. This feature can be used in two different applications; Within the Power Delivery specification or as a fully programmable power supply.

Use Cases

- Over voltage testing
- Under voltage testing
- 6 channel bench top power supply

Note: This feature has the ability to damage your device. Acroname is not responsible for any damage incurred by using this feature.

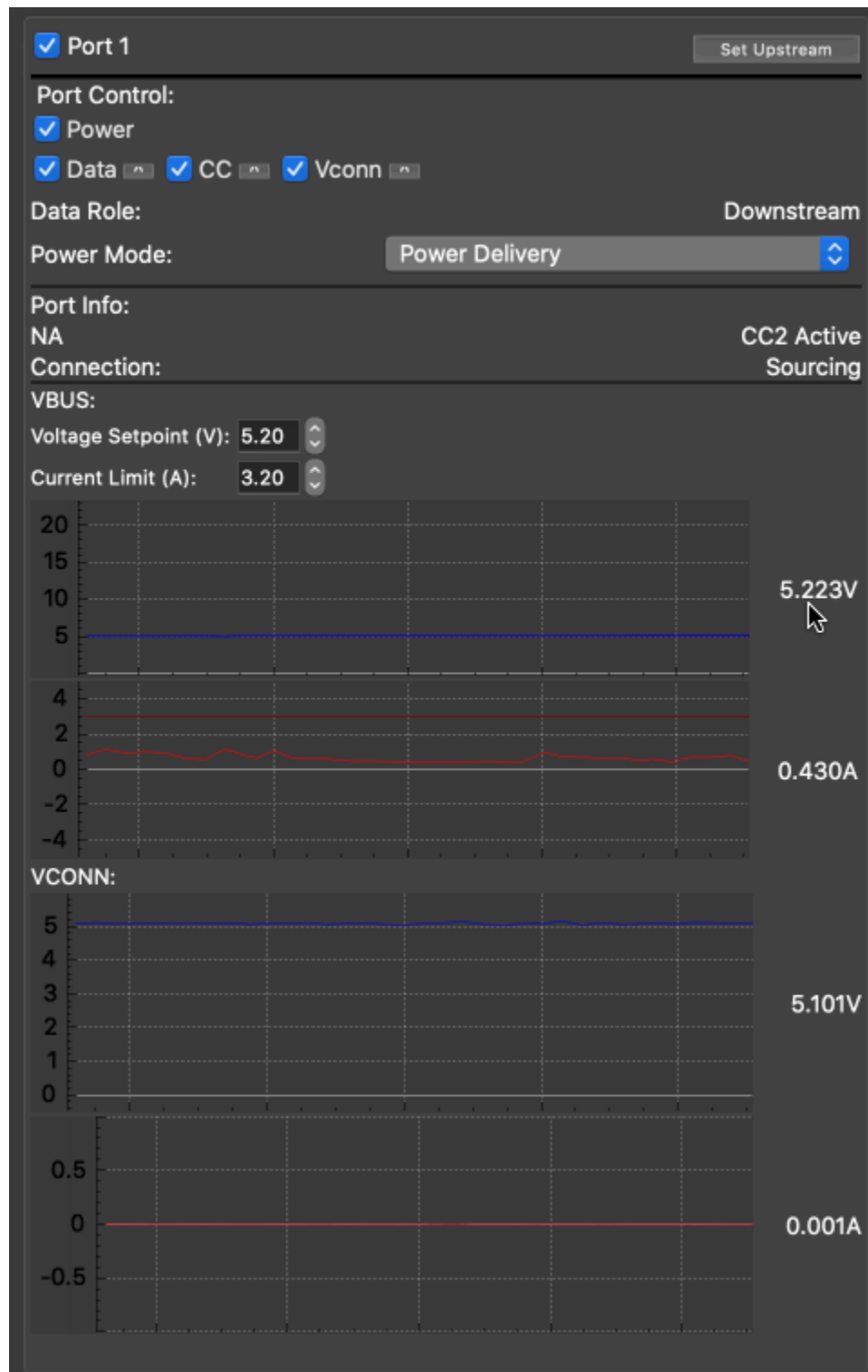
This software feature can be exploited through the *USBHub3c Port Entity*

VBus Validation - Power Delivery Mode

Using the VBus Validation feature within the power delivery mode allows the user to test if their device responds properly when a power source is behaving incorrectly or operating at the edge of specification.

It is important to remember that in this mode the USBHub3c 's power delivery engine also has access to these controls and therefore it is important to allow the USBHub3c and the device to finish negotiations before adjusting these values. Additionally, any PD events or errors can trigger re-negotiations which will replace any values the user has set.

This mode should only be used when the Power Delivery connection state is sourcing.



Example

C++

```
static const int TEST_PORT = 1;
```

(continues on next page)

(continued from previous page)

```

aUSBHub3c stem;
stem.discoverAndConnect(USB);

//Configure the Power mode: Power Delivery (default)
stem.hub.port[TEST_PORT].setPowerMode(portPowerMode_pd_Value);

//Check if we are sourcing power
uint8_t connectionState = 0;
stem.pd[TEST_PORT].getConnectionState(&connectionState);

//Ensure we have an RDO from the remote.
//This ensures we have finished negotiating.
uint32_t rdo = 0;
stem.pd[TEST_PORT].getRequestDataObject(powerdeliveryPartnerRemote, &rdo);

if((connectionState == powerdeliveryPowerRoleSource) &&
    (rdo > 0))
{
    //Do Stuff
    //Set desired port voltage and limit.
    stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000); //5.5VDC
    stem.hub.port[TEST_PORT].setCurrentLimit(500000); //500mA
    //Do Stuff
}

stem.disconnect()

```

Python

```

TEST_PORT = 1;

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

#Configure the Power mode: Power Delivery (default)
stem.hub.port[TEST_PORT].setPowerMode(_BS_C.portPowerMode_pd_Value);

#Check if we are sourcing power
connection_state_result = stem.pd[TEST_PORT].getConnectionState();

#Ensure we have an RDO from the remote.
#This ensures we have finished negotiating.
rdo_result = stem.pd[TEST_PORT].getRequestDataObject(_BS_C.
    powerdeliveryPartnerRemote);

if ((connection_state_result.value == _BS_C.powerdeliveryPowerRoleSource) and
    (rdo_result.value > 0)):

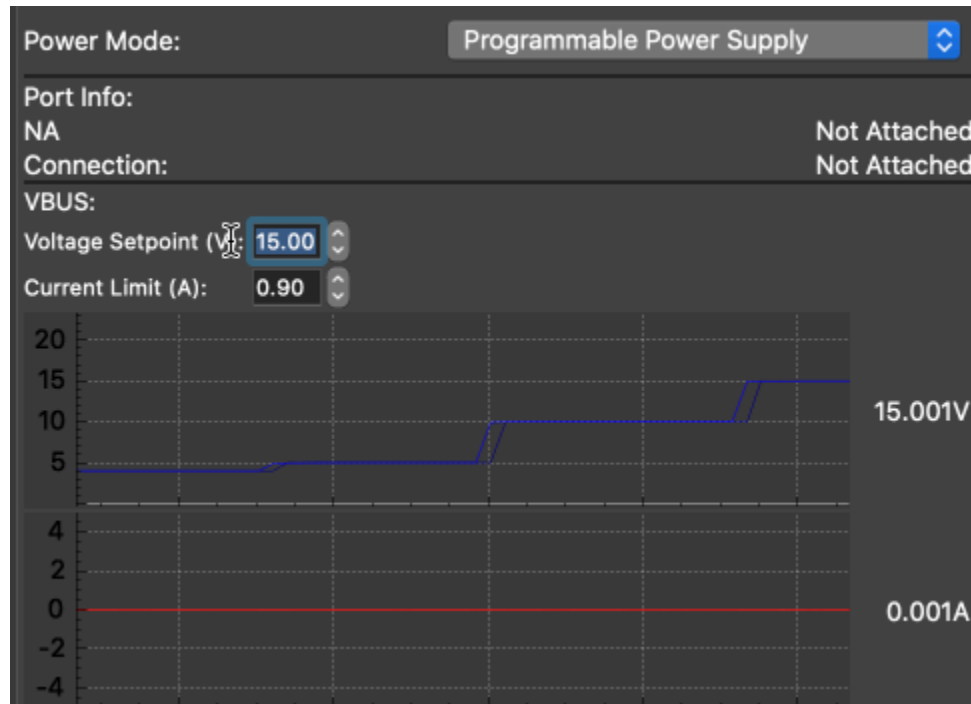
    #Do Stuff
    #Set desired port voltage and limit.
    stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000); #5.5VDC
    stem.hub.port[TEST_PORT].setCurrentLimit(500000); #500mA
    #Do Stuff

stem.disconnect()

```

VBus Validation - Programmable Power Supply Mode

In this mode the USBHub3c is transformed into a 6 channel programmable power supply capable of supplying 100 Watts per channel.



Example

C++

```
static const int TEST_PORT = 1;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

//Disable port while we configure
stem.hub.port[TEST_PORT].setEnabled(false);

//Configure the Power mode: Programmable Power Supply
stem.hub.port[TEST_PORT].setPowerMode(portPowerMode_ps_Value);

//Set desired port voltage and limit.
stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000); //5.5VDC
stem.hub.port[TEST_PORT].setCurrentLimit(500000); //500mA

//enable the port when ready.
stem.hub.port[TEST_PORT].setEnabled(true);

//Do Stuff

//Return port to safe state.
stem.hub.port[TEST_PORT].setEnabled(false)

stem.disconnect()
```

Python

```
TEST_PORT = 1;

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

#Disable port while we configure
stem.hub.port[TEST_PORT].setEnabled(false)

//Configure the Power mode: Programmable Power Supply
stem.hub.port[TEST_PORT].setPowerMode(_BS_C.portPowerMode_ps_Value);

#Set desired port voltage and limit.
stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000)      #5.5VDC
stem.hub.port[TEST_PORT].setCurrentLimit(500000)          #500mA

#Enable the port when ready.
stem.hub.port[TEST_PORT].setEnabled(true)

#Do Stuff

#Return port to safe state.
stem.hub.port[TEST_PORT].setEnabled(false)

stem.disconnect()
```

Relevant API's

```
stem.hub.port[x].setVoltageSetpoint() [cpp] [python] [NET] [CCA] [REST] stem.hub.
port[x].getVoltageSetpoint() [cpp] [python] [NET] [CCA] [REST]

stem.hub.port[x].setCurrentLimit() [cpp] [python] [NET] [CCA] [REST] stem.hub.
port[x].getCurrentLimit() [cpp] [python] [NET] [CCA] [REST]

stem.hub.port[x].setPowerMode() [cpp] [python] [NET] [CCA] [REST] stem.hub.
port[x].getPowerMode() [cpp] [python] [NET] [CCA] [REST]

stem.hub.pd[x].setRequestDataObject() [cpp] [python] [NET] [CCA] [REST] stem.hub.
pd[x].getRequestDataObject() [cpp] [python] [NET] [CCA] [REST]

stem.hub.pd[x].getConnectionState() [cpp] [python] [NET] [CCA] [REST]
```

UART Serial Communication

The UART Serial Communication feature allows one to send commands to affect control and functionality of the USBHub3c.

Use Cases

- Affecting USBHub3c control.
- Audio/Video applications.

Configuration

The default configuration of the UART Serial Communication feature is:

- 8 Data bits
- No Parity
- No Flow Control
- 1 Stop bit
- 9600 Baudrate

This feature does have some configurability through the *USBHub3c UART Entity*

Extron Compatible Serial Commands

The UART Serial Communication feature is capable of changing the USBHub3c upstream port, requesting the current status of the upstream/downstream connections, enable/disable of ports, and the USBHub3c part number/firmware version queries. This can be accomplished with a protocol that is compatible with Extron's Simple Instruction Set over RS232.

Commands

The following is a list of all commands the USBHub3c supports with their arguments, descriptions, and expected responses.

Cmd	Arguments	Description	Expected Responses
!	None	Get current upstream port index	Chn #\r\n
# !	# Port	Change upstream port to # port number	Chn #\r\n
# ^	# Port	Change upstream port to # port number	Chn #\r\n
I	None	Get connection status	Chn # InACT\$\$\$\$\$\$ OutACT\$\$\$\$\$\$\r\n
N	None	Get part number	S99-USBHUB-3C-PRO\r\n S99-USBHUB-3C-LAB\r\n
Q	None	Get firmware version	<M>.<m>.<p>\r\n
# P	# Port	Get enable/disable status of # port number	Port #*0\r\n Port #*1\r\n
# * \$ P	# Port \$ Enable 0/1	Set \$ enable/disable of # port number	Port #*\$\r\n

Error Codes

The following is a list of all error codes the USBHub3c supports with descriptions.

Code	Description
E01	invalid port number, check the port number and make sure it's valid.
E10	invalid command, verify that you formatted the command correctly.
E13	invalid value, verify that the value is within the acceptable range for this command.
E14	invalid configuration, verify the system is in a state it can accept this command.

General Notes

All commands are ASCII strings.

\r is the ASCII character for carriage return.

\n is the ASCII character for new line.

Examples

Extron Compatible Serial Commands:

Upstream Change

Change Upstream to Port 1

Tx: 1!

Rx: Chn 1\r\n

Port Status

Get Port status with Upstream set to port 1, an upstream device on port 1 and Downstream device on port 2

Tx: I

Rx: Chn 1 InACT010000 OutACT001000\r\n

Port Disable

Disable Port 3

Tx: 3*0P

Rx: Port 3*0\r\n

API Configurations:

C++

```
static const int TEST_SERIAL = 0;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

// Enable the port
stem.uart[TEST_SERIAL].setEnabled(true);

// Change baudrate to 115200 from default 9600
stem.uart[TEST_SERIAL].setBaudRate(115200);
```

(continues on next page)

(continued from previous page)

```
// Change Protocol to Extron Compatible
// 0 - Disabled/Undefined
// 1 - Extron Compatible
// 2 - Brainstem Transport
// 6 - Loopback
stem.uart[TEST_SERIAL].setProtocol(1)

// Perform a system save so the changes persist
// through power cycles
stem.system.save();

stem.disconnect();
```

Python

```
TEST_SERIAL = 0

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

# Enable the port
stem.uart[TEST_SERIAL].setEnabled(1)

# Change baudrate to 115200 from default 9600
stem.uart[TEST_SERIAL].setBaudRate(115200)

# Change Protocol to Extron Compatible
# 0 - Disabled/Undefined
# 1 - Extron Compatible
# 2 - Brainstem Transport
# 6 - Loopback
stem.uart[TEST_SERIAL].setProtocol(1)

# Perform a system save so the changes persist
# through power cycles
stem.system.save()

stem.disconnect()
```

Loopback Protocol

```
# Configure UART[0] for loopback testing
# Data sent to the UART will be echoed back
TEST_SERIAL = 0

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

stem.uart[TEST_SERIAL].setEnabled(1)
stem.uart[TEST_SERIAL].setBaudRate(115200)
stem.uart[TEST_SERIAL].setProtocol(6) # Loopback protocol

# Now any data sent to UART[0] will be echoed back
stem.system.save()
stem.disconnect()
```

VCOM to Hardware UART Link

```

# Link VCOM channel (UART[1]) to hardware RS232 (UART[0])
# This allows USB serial communication to be forwarded to RS232
stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

# Enable both channels
stem.uart[0].setEnabled(1)    # Hardware RS232
stem.uart[1].setEnabled(1)    # VCOM channel

# Configure matching baud rates
stem.uart[0].setBaudRate(115200)
stem.uart[1].setBaudRate(115200)

# Link VCOM to hardware UART
stem.uart[0].setLinkChannel(1)
stem.uart[1].setLinkChannel(0)

# Data received on VCOM will now be forwarded to RS232,
# and data received on RS232 will be forwarded to VCOM
stem.system.save()
stem.disconnect()

```

Loopback Protocol

```

// Configure UART[0] for loopback testing
// Data sent to the UART will be echoed back
static const int TEST_SERIAL = 0;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

stem.uart[TEST_SERIAL].setEnabled(true);
stem.uart[TEST_SERIAL].setBaudRate(115200);
stem.uart[TEST_SERIAL].setProtocol(6); // Loopback protocol

// Now any data sent to UART[0] will be echoed back
stem.system.save();
stem.disconnect();

```

VCOM to Hardware UART Link

```

// Link VCOM channel (UART[1]) to hardware RS232 (UART[0])
// This allows USB serial communication to be forwarded to RS232
aUSBHub3c stem;
stem.discoverAndConnect(USB);

// Enable both channels
stem.uart[0].setEnabled(true); // Hardware RS232
stem.uart[1].setEnabled(true); // VCOM channel

// Configure matching baud rates
stem.uart[0].setBaudRate(115200);
stem.uart[1].setBaudRate(115200);

// Link VCOM to hardware UART
stem.uart[1].setLinkChannel(0);
stem.uart[0].setLinkChannel(1);

```

(continues on next page)

(continued from previous page)

```
// Data received on VCOM will now be forwarded to RS232,  
// and data received on RS232 will be forwarded to VCOM  
stem.system.save();  
stem.disconnect();
```

Relevant API's

```
stem.uart[x].setEnabled() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getEnable() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].setBaudRate() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getBaudRate() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].setProtocol() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getProtocol() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].setLinkChannel() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getLinkChannel() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].setStopBits() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getStopBits() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].setParity() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getParity() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].setDataBits() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getDataBits() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getCapableProtocols() [cpp] [python] [NET] [CCA] [REST]  
stem.uart[x].getAvailableProtocols() [cpp] [python] [NET] [CCA] [REST]
```

1.3 USBHub2x4



The USBHub2x4 gives engineers advanced flexibility and configurability over USB ports in testing and development applications. It can be used to enable/disable individual USB ports, measure current or voltage on downstream USB ports, set programmable current limits, set USB charging protocol behavior and otherwise automate USB port behaviors in development and testing.

To get up to speed with the USBHub2x4 and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [Programming interface](#) of the USBHub2x4 for a more in depth view.

1.3.1 Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)⁶ for your particular operating system and architecture.
- Download [HubTool](#)⁷ for your particular operating system and architecture.

⁶ <https://acroname.com/api>

⁷ <https://acroname.com/hubtool>

2. Connect Power

- Using the provided universal power supply connect the barrel jack into the hub.
- Connect the other end into a 120/240V AC outlet.

3. Connect Data

- With the provided USB 3.0 A-MiniB cable connect the A side to your host computer and the MiniB side to the connection labeled “Up0”.

4. Run System

- Open HubTool
- On the bottom right side of the application select the USBHub2x4 device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the USBHub2x4. For more information please take a look at our [Getting Started Guide](#)

1.3.2 Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a USBHub2x4 module.
    aUSBHub2x4 hub;

    //Connect to the hardware.
    err = hub.discoverAndConnect(USB);
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
        return 1;
    } else { printf("Connected to BrainStem module.\n"); }

    //Basic initialization (Get everything turned off).
    hub.usb.setPortDisable(0);
    hub.usb.setPortDisable(1);
    hub.usb.setPortDisable(2);
    hub.usb.setPortDisable(3);

    ////////////
    //Do Stuff: other test initialization
    ////////////
}
```

(continues on next page)

(continued from previous page)

```

//Ready for testing
//Enable Port(s)
hub.usb.setPortEnable(0);
hub.usb.setPortDisable(1);
hub.usb.setPortDisable(2);
hub.usb.setPortDisable(3);

//////////
//Do Stuff on Port 0
//////////

hub.usb.setPortDisable(0);
hub.usb.setPortEnable(1);
hub.usb.setPortDisable(2);
hub.usb.setPortDisable(3);

//////////
//Do Stuff on Port 1
//////////

hub.usb.setPortDisable(0);
hub.usb.setPortDisable(1);
hub.usb.setPortEnable(2);
hub.usb.setPortDisable(3);

//////////
//Do Stuff on Port 2
//////////

hub.usb.setPortDisable(0);
hub.usb.setPortDisable(1);
hub.usb.setPortEnable(2);
hub.usb.setPortDisable(3);

//////////
//Do Stuff on Port 3
//////////

//Finished with testing.
//De-initialize.
hub.usb.setPortDisable(0);
hub.usb.setPortDisable(1);
hub.usb.setPortDisable(2);
hub.usb.setPortDisable(3);

//Disconnect
hub.disconnect();
}

```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

```

(continues on next page)

(continued from previous page)

```

# Create an instance of a USBHub2x4 module.
hub = brainstem.stem.USBHub2x4()

# Locate and connect to the first object you find on USB
result = hub.discoverAndConnect(brainstem.link.Spec.USB)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
hub.usb.setPortDisable(0)
hub.usb.setPortDisable(1)
hub.usb.setPortDisable(2)
hub.usb.setPortDisable(3)

#####
##Do Stuff other test initialization
#####

##Ready for testing
##Enable Port(s)
hub.usb.setPortEnable(0)
hub.usb.setPortDisable(1)
hub.usb.setPortDisable(2)
hub.usb.setPortDisable(3)

#####
##Do Stuff on Port 0
#####

hub.usb.setPortDisable(0)
hub.usb.setPortEnable(1)
hub.usb.setPortDisable(2)
hub.usb.setPortDisable(3)

#####
##Do Stuff on Port 1
#####

hub.usb.setPortDisable(0)
hub.usb.setPortDisable(1)
hub.usb.setPortEnable(2)
hub.usb.setPortDisable(3)

#####
##Do Stuff on Port 2
#####

hub.usb.setPortDisable(0)
hub.usb.setPortDisable(1)
hub.usb.setPortDisable(2)
hub.usb.setPortEnable(3)

#####

```

(continues on next page)

(continued from previous page)

```

##Do Stuff on Port 3
#####

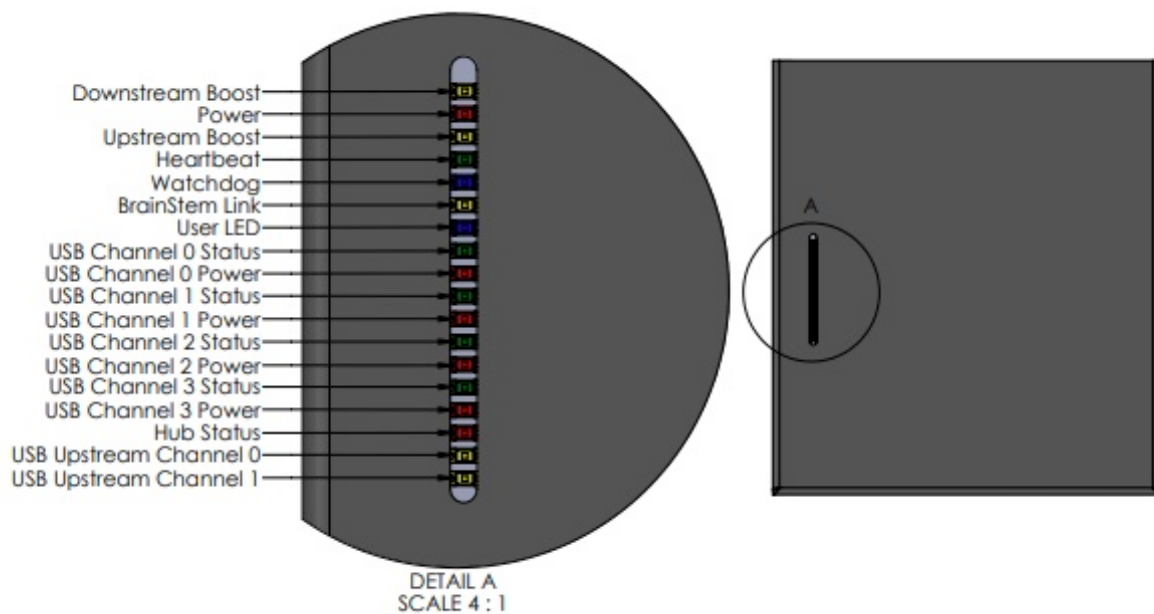
##Finished with testing.
##De-initialize.
hub.usb.setPortDisable(0)
hub.usb.setPortDisable(1)
hub.usb.setPortDisable(2)
hub.usb.setPortDisable(3)

# Disconnect from device.
hub.disconnect();
print("Disconnected from BrainStem module. \n")

```

1.3.3 Indicators and Connections

LEDs and Connections



LED Name	Color	Description
Downstream Boost	Yellow	Indicated the Downstream Data Boost is enabled through the USB Entity API
Power	Red	Shows that a 3.3V voltage regulation system is up and running properly.
Upstream Boost	Yellow	Indicated the Upstream Data Boost is enabled through the USB Entity API
Heartbeat	Green	Communication is occurring with the BrainStem module
Watchdog	Blue	Indication the internal watchdog is running and the connection with the Host is healthy
BrainStem Link	Yellow	The BrainStem USB interface is created on a host computer
User LED	Blue	A software controllable indicator accessed via the System BrainStem Entity. See the System Entity API Reference Page.
USB Channel 0:3 Status	Green	Indicates whether the downstream device has enumerated on the host computer
USB Channel 0:3 Power	Red	Indicates an error on USB power (Vbus) such as overcurrent
Hub Status	Red	The USB hub communicates with a host computer.
USB Upstream Channel 0	Yellow	Indicated Upstream 0 has been selected
USB Upstream Channel 1	Yellow	Indicated Upstream 1 has been selected

1.3.4 Programming Interface

The USBHub2x4 is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the USBHub2x4.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

Software control of the features of the USBHub2x4 is done with the BrainStem API via a BrainStem link. BrainStem links are done over USB and can be established via upstream port 0 (Up0) and upstream port 1 (Up1). After one or more of these ports is connected to a host machine, a user can connect to it via software API:

```
stem.link.discoverAndConnect(USB)
```

When multiple Acroname devices are connected to a host, connecting to a specific hub can be done by providing the hub serial number. Further, all connected devices can be found using

```
brainstem.discover.findAllModules(USB) (Python)
Acroname::BrainStem::Link::sDiscover() (C++)
```

Using Multiple Hosts with USBHub2x4

The two upstream-facing host ports can be connected to two different host computers. Due to limitations of USB specification, only one host computer can access downstream USB ports at any time. Through the BrainStem API, the upstream port used can be controlled, or the system can automatically select the upstream port (see USB Hub Upstream Mode). When automatically selecting the upstream port, the USBHub2x4 will default to using Up0 if it is connected.

Device Drivers

The USBHub2x4 leverages operating system user space interfaces that do not require custom drivers for operation on modern operating systems.

Some older operating systems may require the installation of a BrainStem USB driver to enable software control. Installation details on installing USB drivers can be found within the BrainStem Development Kit under the “drivers” folder. For example, Windows 7 requires the supplied INF to communicate with BrainStem USB devices.

1.3.5 USBHub2x4 Module Entities

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every USBHub2x4 is assigned a unique serial number at the factory. This facilitates an arbitrary number of USBHub2x4 devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for USBHub2x4 is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the USBHub2x4 away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Pressing the reset button two times within 5 seconds will return all settings to factory defaults: all ports' data and power enabled, CDP mode, enumeration delay of 0, 2500mA current limit.

Savable Items	
Software Offset	I2C Rate
Router Address	Port Enumeration Delay
Boot Slot	Downstream Boost
Port Mode (SDP, CDP) - each port	Current Limit - per port
Upstream Boost	Port state (data and power)

Temperature

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Certain modules have a temperature measurement available. The temperature entity gives access to these measurements. Check your module datasheet to see if your module has a temperature entity.

System Temperature

The temperature of the USBHub2x4 can be measured with:

```
stem.temperature[0].getValue(μC) [cpp] [python] [NET] [CCA] [REST]
```

where temperature is in micro-degrees Celcius.

USB

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USB Entity provides the software control interface for USB related features. This entity is supported by BrainStem products which have programmatically controlled USB features.

Downstream Channel Control

Downstream USB channels can be manipulated through the usb entity command to enable and disable USB data and Vbus lines, measure current, measure Vbus voltage, boost data line signals, and measure temperature.

Manipulating Hi-Speed data and Vbus lines simultaneously for a single port can be done by calling the following methods with channel in [0-3] being the port index:

```
stem.usb.setPortEnable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.usb.setPortDisable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Manipulating Hi-Speed data while not affecting the Vbus lines simultaneously for a single port can be done by calling the following methods with channel [0-3]. The following methods provide equivalent functionality; the two methods are offered for compatibility with other products.

```
stem.usb.setDataEnable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.usb.setDataDisable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.usb.setHiSpeedDataEnable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.usb.setHiSpeedDataDisable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Manipulating just the USB Vbus line for a single port can be done by calling the following method with channel [0-3]:

```
stem.usb.setPowerEnable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.usb.setPowerDisable(channel) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

To affect multiple ports and lines simultaneously, see `usb.setHubMode()` later in this section.

Note that transitions between power and data enables states where power is enabled and only data is changing, require the USBHub2x4 to toggle Vbus power. This appears as a port cycle event and the USBHub2x4 hardware will cycle Vbus even if the Vbus/Power setting is enabled.

Downstream Measurements

The USB Vbus voltage, as well as the current consumed on Vbus, can be read for each channel by calling the following methods with channel [0-3], where the second variable passed into the method is the location for the measurement result:

```
stem.usb.getPortVoltage(channel,  $\mu$ V) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.usb.getPortCurrent(channel,  $\mu$ A) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Downstream Current Limiting

Current-limit trip point settings can be accessed for each port by calling the following methods with channel [0-3], where the second variable passed into the method is either the set value or the write location of the result:

```
stem.usb.getPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

The current-limiting behavior follows the USB BC1.2 specification which allows for many different behaviors. The USBHub2x4 has two stages of current-limiting. When a downstream device consumes current higher than the programmed current limit, the hub will enter a “constant current” mode and is indicated in the `getPortState()` bitfield with the constant current bit. In the constant current mode, the Vbus voltage will be reduced to attempt maintain a constant current at the set current limit. The time and amount of voltage reduction and maximum allowed current draw depends on the current limit set point.

As the Vbus voltage is reduced, if the device continues to increase its current draw (reduce it’s effective resistance), the USBHub2x4 will “trip off” by disabling the Vbus and high-speed data lines. This state is indicated with the error bit in the `getPortState()` bitfield. The Channel X Power error LED will also illuminate when this error occurs. See the [LED Indicators](#).

Downstream Enumeration Speed

The enumeration state and speed of each downstream port can be read with

```
stem.usb.getDownstreamDataSpeed [cpp] [python] [NET] [CCA] [REST]
```

Value	Hub Downstream Speed Descriptions
0	No device enumerated
1	Hi-Speed device enumerated

Downstream Operational Mode

The USB port operational mode controls the behavior of each downstream port’s charging behavior. Each port can be setup to support different modes in the USB Battery Charge Specification 1.2 (BC1.2). Standard Downstream Port (SDP) mode will cause BC1.2 compliant or older USB devices to consume 500mA or less. Configuring a port as a Charging Downstream Port (CDP) will cause the hub signal to downstream devices that devices may consume up to 5A, the maximum allowed by BC1.2. If there is no upstream USB host connected to the hub, downstream ports set to CDP will behave as Dedicated Charging Ports (DCP).

The actual current consumed by the device is controlled by the downstream device and not the USBHub2x4. Devices which are not compliant with BC1.2 or the previous USB power specifications may draw more current than specified above.

The operational mode is set or read by calling the methods:

```
stem.usb.getPortMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

Value	Hub Port Mode Descriptions
0	Standard downstream port (SDP)
1	Charging downstream port (CDP)

Note: A `system.save()` and `system.reset()` is required before the new setting will take affect.

Downstream Enumeration Delay

Once a USB device is detected by the USBHub2x4 it is possible to delay its connection to an upstream host computer and subsequent enumeration on the USB bus. The enumeration delay can mitigate or eliminate host kernel instabilities by forcing devices to enumerate in slow succession, allowing a focus on validation of drivers and software. The enumeration delay is configured in milliseconds, representing the time delay between enabling each successive downstream port from 0 to 3. Enumeration delay is applied when the hub powers on or when a new upstream connection is made.

```
stem.usb.setEnumerationDelay(delay) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getEnumerationDelay(delay) [cpp] [python] [NET] [CCA] [REST]
```

Hub Operational Mode

In addition to targeting individual downstream USB ports, a bit-mapped hub mode interface is also available. This interface allows the reading or setting of all USB downstream ports in one functional call.

Auto Vbus Toggle

By default the USBHub2x4 will toggle its downstream ports anytime the host connection is lost, changed or disconnected. Disabling (setting the bit) will cause the hub to not cycle downstream power on upstream changes. This behavior can be helpful for certain host controllers and devices. Enumeration delay will override this setting.

```
stem.usb.getHubMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setHubMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

This command returns a 32-bit value which indicates:

Bit	Hub Operational Mode Bitwise Description
0	USB Channel 0 USB Hi-Speed Data Enabled
1	USB Channel 0 USB Vbus Enabled
2	USB Channel 1 USB Hi-Speed Data Enabled
3	USB Channel 1 USB Vbus Enabled
4	USB Channel 2 USB Hi-Speed Data Enabled
5	USB Channel 2 USB Vbus Enabled
6	USB Channel 3 USB Hi-Speed Data Enabled
7	USB Channel 3 USB Vbus Enabled
8:31	Reserved

Hub Upstream Channels

The USBHub2x4 is perfect for environments where multiple devices need to be shared or switched between two host computers using two host (upstream) connections via USB standard-B connectors. The upstream connection can be automatically detected or specifically selected using the following methods:

```
stem.usb.getUpstreamMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setUpstreamMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

The mode parameter can be defined as the following:

Value	Hub Upstream Mode Descriptions
0	Force upstream port 0 to be selected
1	Force upstream port 1 to be selected
2	Automatically detect upstream port

Predefined C++ macros for these can be found in aProtocoldef.h, and Python's built-in help interface.

The default operational mode is to auto detect which upstream USB port is selected. Automatic detection uses the presence of Vbus on the USB type-B upstream connector to determine presence of a host. If only one upstream port is connected to a host, it will be used for upstream USB. If both upstream ports are connected, the hub will use upstream port 0.

If the Hub Upstream Mode is set to disconnect both upstream ports (or the only active upstream port), the only path available to establish a BrainStem link to the USBHub2x4 will be via a host connected to the BrainStem Control Port.

Hub Upstream State

The USBHub2x4 can provide status information on which upstream port is actively selected as data path to the downstream ports:

```
stem.usb.getUpstreamState(mode) [cpp] [python] [NET] [CCA] [REST]
```

Value	Definitions	Hub Upstream Mode Descriptions
0	usbUpstreamStatePort0	Upstream port 0 is actively selected
1	usbUpstreamStatePort1	Upstream port 1 is actively selected
2	usbUpstreamStateNone	No upstream port is selected

Port State

Each downstream port reports information regarding its operating state represented in bit-packed results from:

```
stem.usb.getPortState(state) [cpp] [python] [NET] [CCA] [REST]
```

where channel can be [0-3], and the value status is 32-bit word, defined as the following:

Bit	Port State: Result Bitwise Description
0	USB Vbus Enabled
1	USB2 Data Enabled
2:18	Reserved
19	USB Error Flag
20	USB2 Boost Enabled
21:22	Reserved
23	Device Attached
24	Constant Current Mode
25:31	Reserved

Port Error Status Mapping

Error states for all downstream ports are bit-packed in 32-bit words available from:

```
stem.usb.getPortError(channel) [cpp] [python] [NET] [CCA] [REST]
```

where channel is [0-3].

Errors can be cleared on each individual channel by calling the following method:

```
stem.usb.clearPortErrorStatus(channel) [cpp] [python] [NET] [CCA] [REST]
```

Calling this command clears the port-related error bit flags in the port error state. Global bits for hub errors cannot be cleared by this command.

Details about the port error status 32-bit word are as follows:

Bit	Port Error Status Bitwise Description
0	USB port current limit exceeded
1	USB port back-drive condition detected
2	Reserved
3	Hub over temperature condition
4	VBus Discharge error
5:31	Reserved

Boost Mode

Boost mode increases the drive strength of the USB 2.0 Hi-Speed data signals (power signals are not changed). Boosting the data signal drive strength may help to overcome connectivity issues when using long cables or connecting through relays, “pogo” pins or other adverse conditions. This setting is applied after a `system.save()` call and reset or power cycle of the hub. The system setting is persistent until changed or the hub is hard reset. After a hard reset, the default value of 0% boost is restored. A hard reset is done by pressing the “Reset” button on the back of the hub while the hub is powered.

Boost mode can be applied to both the upstream and downstream USB ports with the follow methods:

```
stem.usb.getDownstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setDownstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getUpstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```



```
stem.usb.setUpstreamBoostMode(setting) [cpp] [python] [NET] [CCA] [REST]
```

The setting parameter is an integer that correlates to the following:

Value	Hub Boost Mode Descriptions
0	Normal drive strength
1	4% increase in drive strength
2	8% increase in drive strength
3	12% increase in drive strength

Port

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

The Port Entity provides control over the most basic items related to a USB Port. This includes actions ranging from a complete port enable and disable to the individual interface control. Voltage and current measurements are also included for devices which support the Port Entity.

Port Control

The USBHub2x4 has a Port Entity for every port on the device; however, not all ports have the same capabilities. These ports can be referenced by their instance (port[x]) index.

Port Label	Index (port[x])
0	0
1	1
2	2
3	3
Up0	4
Up1	5

One of the most powerful features of the USBHub2x4 is its ability to turn ports on and off which is available on Ports 0-3.

```
stem.hub.port[x].setEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB Vbus line for a single port can be done by calling the following method on Ports 0-3.

```
stem.hub.port[x].setPowerEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating data lines while not affecting the Vbus lines simultaneously for a single port can be done by calling the following method for Ports 0-3. (For USBHub2x4 the only data lines happen to be USB 2.0 Hi-Speed)

```
stem.hub.port[x].setDataEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Manipulating just the USB 2.0 Hi-Speed data lines for a single port can be done by calling the following for Ports 0-3.

```
stem.hub.port[x].setDataHSEnabled() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getDataHSEnabled() [cpp] [python] [NET] [CCA] [REST]
```

Voltage and Current Measurements

The USBHub2x4 provides Voltage and Current measurements for Vbus. These values can be acquired for all 4 ports through the following APIs

```
stem.hub.port[x].getVbusVoltage() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getVbusCurrent() [cpp] [python] [NET] [CCA] [REST]
```

Power Modes

The ports of the USBHub2x4 are capable of providing power in multiple formats. The default is Charging Downstream Port (CDP), but that can be changed to things like: Standard Downstream Port (SDP), Charging Downstream Port (CDP) / Dedicated Charging Port (DCP). These modes can be set through:

```
stem.hub.port[x].setPowerMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getPowerMode() [cpp] [python] [NET] [CCA] [REST]
```

Power Mode	Value	Define
None	0	portPowerMode_none_Value
SDP	1	portPowerMode_sdp_Value
CDP/DCP	2	portPowerMode_cdp_dcp_Value

Warning: The USBHub2x4 does not have a dedicated control port, therefore all control is done through the upstream. If you change upstreams while controlling it, the control connection will be lost and the new host will have to take over.

Port Mode

As outlined in the “Port Control” section the USBHub2x4 can individually manipulate almost every pin on the connector; however, depending on your application that might require multiple function calls in order to configure the port how you want it. Port Mode on the other hand is a one stop shop that allows you to pick and choose which lines you want enabled or disabled through a single call. Additionally, it has a few other features tucked away inside of it.

```
stem.hub.port[x].setMode() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getMode() [cpp] [python] [NET] [CCA] [REST]
```

Port Mode Item	Bit	Value	Define
Power Enable	0	0/1	portPortMode_powerEnabled_Bit
HS 1 Enable	1	0/1	portPortMode_HS1Enabled_Bit
Power Mode: Offset		16	portPortMode_portPowerMode_Offset
Power Mode: Mask		0x7	portPortMode_portPowerMode_Mask
Power Mode: None	16-18	0	portPortMode_portPowerMode_none_Value
Power Mode: SDP	16-18	1	portPortMode_portPowerMode_sdp_Value
Power Mode: CDP/DCP	16-18	2	portPortMode_cdp_dcp_Value

Data Role

The data role describes the current configuration of the port in regards to its data direction. In most cases this evaluates to an Upstream Facing Port (UFP) or a Downstream Facing Port (DFP). Upstream in this case means the host side of the port and Downstream refers to the device side. The Data Role can be aquired through:

```
stem.hub.port[x].getDataRole() [cpp] [python] [NET] [CCA] [REST]
```

Data Role	Value	Define
Disabled	0	portDataRole_Disabled_Value
Upstream	1	portDataRole_Upstream_Value
Downstream	2	portDataRole_Downstream_Value
Control	3	portDataRole_Control_Value

Port Limits and Modes

At the Port level the user has the ability to define current limit.

```
stem.hub.port[x].setCurrentLimit() [cpp] [python] [NET] [CCA] [REST]
stem.hub.port[x].getCurrentLimit() [cpp] [python] [NET] [CCA] [REST]
```

Downstream Data Speed

The USBHub2x4 can detect if a device has been enumerated. Additionally, it can detect at what speed a device has enumerated at.

```
stem.hub.port[x].getDataSpeed() [cpp] [python] [NET] [CCA] [REST]
```

Data Speed	Bit	Value	Define
1.5 Mbit/s	0	0/1	portDataSpeed_ls_1p5M_Bit
12 Mbit/s	1	0/1	portDataSpeed_fs_12M_Bit
480 Mbit/s	2	0/1	portDataSpeed_hs_480M_Bit
5 Gbit/s	3	0/1	portDataSpeed_ss_5G_Bit
10 Gbit/s	4	0/1	portDataSpeed_ss_10G_Bit
USB 2.0	6	0/1	portDataSpeed_Connected_2p0_Bit
USB 3.0	7	0/1	portDataSpeed_Connected_3p0_Bit

USB System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USBSystem class provides high level control of the lower level *Port Entity*

Upstream Control

The USBHub2x4 has the ability to designate one of the upstream ports (4-5) as the upstream connection. This is very useful for moving devices between hosts.

```
stem.hub.setUpstream() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getUpstream() [cpp] [python] [NET] [CCA] [REST]
```

Note: The Power Modes can only be changed when the port power is disabled.

Enumeration Delay

Once a USB device is detected by the USBHub2x4 it is possible to delay its connection to an upstream host computer and subsequent enumeration on the USB bus. The enumeration delay can mitigate or eliminate host kernel instabilities by forcing devices to enumerate in slow succession, allowing a focus on validation of drivers and software. The enumeration delay is configured in milliseconds, representing the time delay between enabling each successive port. Enumeration delay is applied when the hub powers on or when a new upstream connection is made.

```
stem.hub.setEnumerationDelay() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getEnumerationDelay() [cpp] [python] [NET] [CCA] [REST]
```

Data Behavior

The USBHub2x4 is capable of a few different behaviors for how it switches upstream port connections. It can auto switch based on port priority or have a fixed upstream port. The method in which these events are handled is referred to as data behavior.

List of Available Data Behaviors for USBHub2x4

Behavior	Value	Define
Hard Coded	0	usbsystemDataBehavior_HardCoded
Reserved	1	usbsystemDataBehavior_Reserved
Port Priority	2	usbsystemDataBehavior_PortPriority

Hard Coded (Default Configuration)

The Hard Coded data behavior is used to fix the Upstream port to a single port and not allow it to move except for a command through the [Set Upstream](#) API.

Port Priority

The Port Priority data behavior prioritizes making the Upstream port the lowest numbered port on the USB-Hub2x4 that is capable of being an Upstream port.

Relevant API's

```
stem.hub.setDataRoleBehavior() [cpp] [python] [NET] [CCA] [REST]
stem.hub.getDataRoleBehavior() [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Non Standard Value
App[0-3]	execute	
	return	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	
	getTransferStore	
	setTransferStore	
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	
	setInt	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
system[0]	slotSize	
	getModel	
	getHardwareVersion	
	getModule	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getVersion	
	getSerialNumber	
	save	
	reset	
	getInputVoltage	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
timer[0-8]	getName	
	setName	
	resetDeviceToFactoryDefaults	
temperature[0]	getExpiration	
	setExpiration	
usb[0]	getTemperature	
	setPortEnable	Channels 0-3

continues on next page

Table 10 – continued from previous page

Entity Class	Entity Option	Non Standard Value
	setPortDisable	Channels 0-3
	setDataEnable	Channels 0-3
	setDataDisable	Channels 0-3
	setHiSpeedDataEnable	Channels 0-3
	setHiSpeedDataDisable	Channels 0-3
	setPowerEnable	Channels 0-3
	setPowerDisable	Channels 0-3
	getPortVoltage	Channels 0-3
	getPortCurrent	Channels 0-3
	getPortCurrentLimit	Channels 0-3
	setPortCurrentLimit	Channels 0-3
	setPortMode	Channels 0-3
	getPortMode	Channels 0-3
	getDownstreamDataSpeed	Channels 0-3
	getHubMode	
	setHubMode	
	getPortState	Channels 0-3
	getPortError	
	getEnumerationDelay	
	setEnumerationDelay	
	clearPortErrorStatus	
	getUpstreamMode	
	setUpstreamMode	
	getUpstreamState	
	getUpstreamBoostMode	
	setUpstreamBoostMode	
	getDownstreamBoostMode	
	setDownstreamBoostMode	
port[0-3]	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getPowerEnabled	
	setPowerEnabled	
	getHSBoost	
	setHSBoost	
	getMode	
	setMode	
	getCurrentLimit	
	setCurrentLimit	
	getVbusVoltage	
	getVbusCurrent	
	getState	
	getName	
	setName	
	getPowerMode	
	setPowerMode	
	getDataRole	

continues on next page

Table 10 – continued from previous page

Entity Class	Entity Option	Non Standard Value
port[4-5]	getDataSpeed	
	getErrors	
	getHSBoost	
	setHSBoost	
	getName	
USBSystem [0]	setName	
	getDataRole	
	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
	getEnumerationDelay	
	setEnumerationDelay	

1.4 USB-C-Switch Pro

1.4.1 Quick Start Guide

1. Install Required Software

- Download the [BrainStem Development Kit \(BDK\)](#)⁸ for your operating system and architecture.
- Download [HubTool](#)⁹ for your operating system and architecture.

2. Connect the Hardware

- Using standard USB-C cables, connect the **Common** and **Control** ports of the USB-C-Switch Pro to the host computer.
- Connect DC power supply (optional)
- Connect devices to any of **Mux Ports 0-3** using standard USB-C cables, or - for testing both sides of a port - use one of the included **Acroname Universal Orientation Cables (UOCs)**.
 - If using standard cables or a **C67 UOC**, you may need to **flip the Switch-side of the cable** to establish a connection.
 - The **C70 UOC** works in either orientation.

Note: Two UOC cable types are available:

- **C70:** for devices with **independent HS sides**
- **C67:** for devices with **shorted HS sides A and B**

See [Which universal orientation cable should I use?](#)¹⁰ for details.

3. Launch HubTool

- Open **HubTool**.
- In the lower-right corner of the application window, select the USB-C-Switch Pro device from the device list.

Note: On **Linux**, run the script named “udev.sh” located in the “BrainStem_linux_Driverless” folder before communicating with BrainStem devices.

Congratulations! You’re now ready to explore the capabilities of USB-C-Switch Pro. For more information, see the [Getting Started Guide](#).

⁸ <https://acroname.com/api>

⁹ <https://acroname.com/hubtool>

¹⁰ <https://acroname.com/blog/which-universal-orientation-cable-should-i-use>

1.4.2 Functionality

Overview

The USB-C-Switch Pro has 6 USB-C ports: 1 Common port, 4 Mux Ports, and a dedicated Control Port. Software control manages signal routing and other port-level features. Each USB-C-Switch Pro is addressable and controllable from a host system via USB (Control Port), net (TCP/IP), I²C, or RS-232. Once connected, a BrainStem® link is established to the onboard controller, enabling software control through the BrainStem API. This API provides full access to all module functionality from a host system.

USB Ports

The Common and Mux ports of the USB-C-Switch Pro implement separate, independently-switched USB HS and SS data, CC, V_{CONN} and current-limited V_{BUS} lines. USB power, HS data and SS data can be independently disconnected for advanced USB testing applications. The USB-C-Switch Pro has a dedicated control channel on the Type C connector labeled “Control”. This is a high-speed USB 2.0 connection for BrainStem interface and power delivery only. No other USB traffic can flow on this connection.

Port-level features are controlled by the *USB Entity*, while Mux port channel selection, port priority, and split mode are controlled by the *Mux Entity*.

Port	Index
Mux 0-3	0-3
Common	4
Control	5

Cable Orientation

A key feature of USB-C is its reversible connector, allowing insertion in either orientation. In a standard USB-C cable, only one of the CC lines and one HS pair are wired through. Orientation is defined by the male plug, which connects only one of the receptacle’s two CC (Configuration Channel) pins through the cable. The connected device detects which CC pin is active and routes signals accordingly.

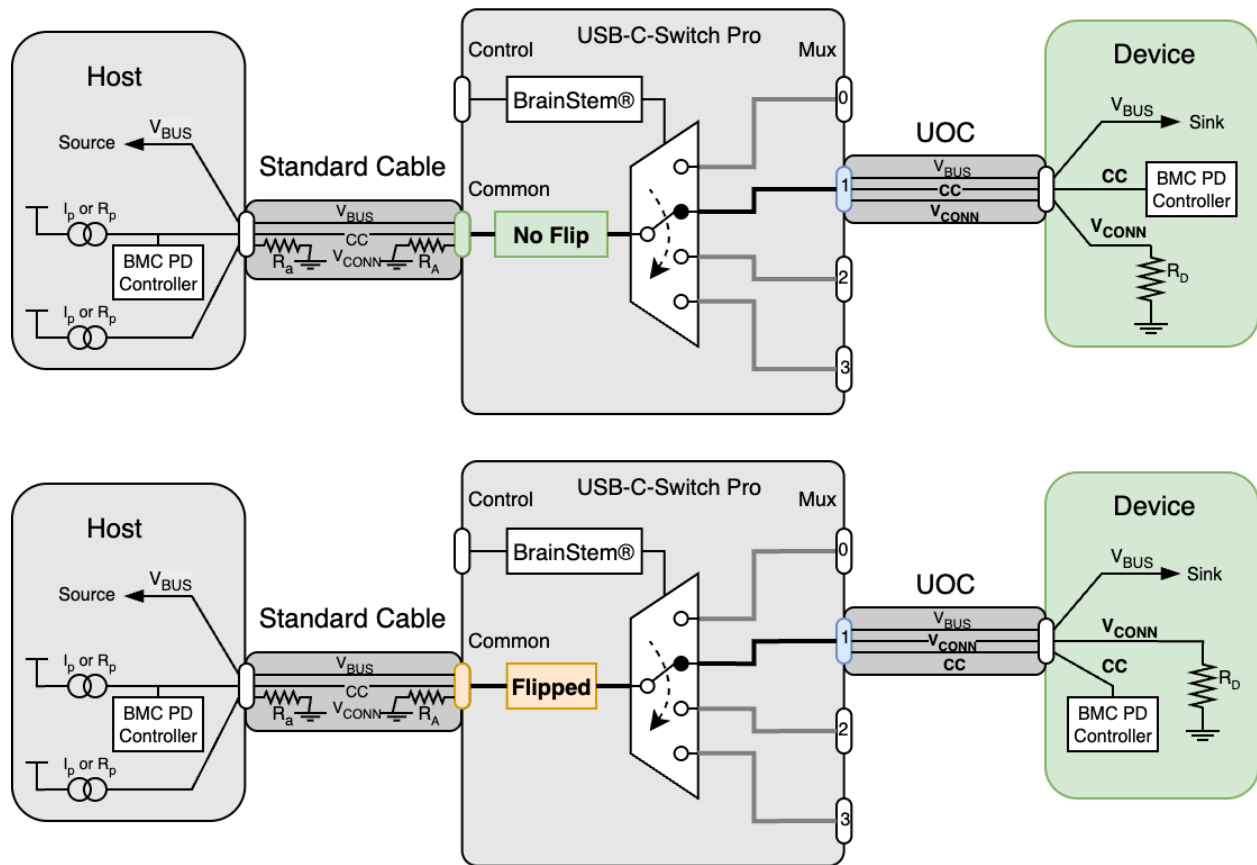
Note: In normal operation, standard USB cables on the Mux and Common sides of the Switch need to be in the same orientation so you may need to flip one of the cables to enable the connection.

Cable Flip

In normal operation, the USB-C-Switch Pro connects signals directly from the Common port to the active Mux port — side A to A and side B to B. The switch can also invert this mapping programmatically, rerouting CC, V_{CONN} , SBU, and USB 2/3 data lines so that side A of the Common port connects to side B of the Mux port (and vice versa). This “cable flip” operation mimics physically reversing the connection between Common and Mux ports.

When used with an Acroname Universal Orientation Cable (UOC), cable flip enables automated testing of both sides of a USB port.

Depicted below are example block diagrams of the flip feature when connecting a host through a standard full-featured, non-marked cable to a direct-connected downstream device. USB SS, HS and SBU lines are also routed, but omitted from the diagram for clarity.



The UOC should be connected to the device under test and either the Common port or Mux port, depending on whether the DUTs or testers are being multiplexed.

UOC Selection

Device USB-C ports either **short** sides A and B USB 2.0 (HS) data lines or use a **mux** to keep the two sides electrically independent. Choose the UOC that matches your DUT port.

Two UOC cable types are included:

- **C70:** routes both CC lines and **two HS pairs**, for devices with **muxed** USB 2.0 sides
- **C67:** routes both CC lines and **one HS pair**, for devices with **shorted** USB 2.0 sides

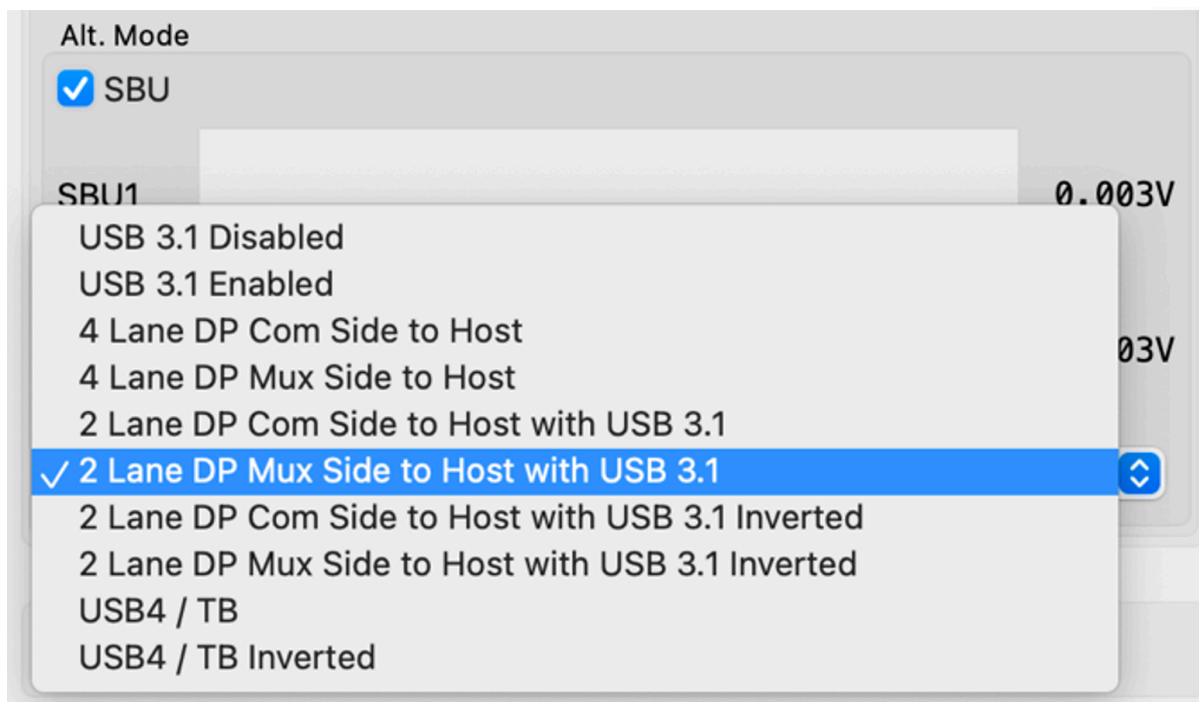
When using C67, it may be necessary to flip the cable at the Switch end to establish a USB 2 connection.

Alt-Mode Configurations

For Alt-modes, the pin mappings and directions may affect connectivity and functionality. In many cases, the connected devices will simply negotiate through the switch. However, when using DisplayPort Alt Mode, USB4, or ThunderBolt 3, some functional groups need to be assigned a specific direction.

In HubTool:

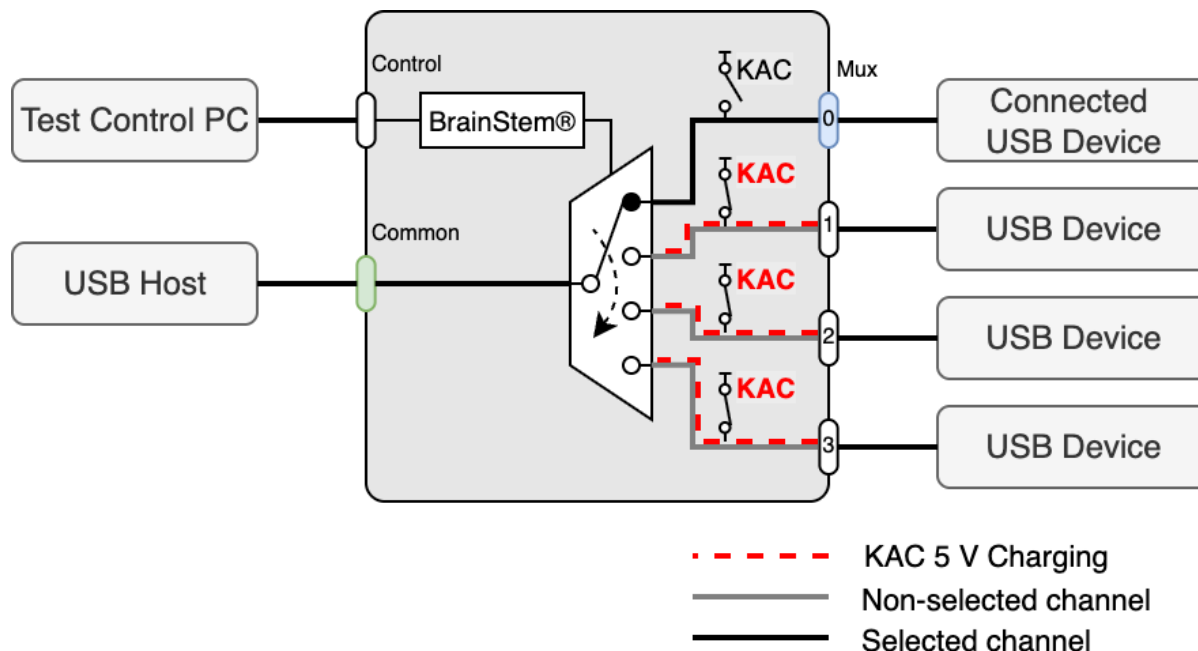
- If device VBUS is not active, toggle cable flip or check cable orientations
- Under Alt Mode, select the corresponding configuration depending on whether the host is on the Common or Mux port and the protocol used. For 2-lane DP and USB4 / TB, it may be necessary to try the inverted and non-inverted option.



Alt-mode configuration menu in HubTool

Keep-Alive Charging (KAC)

It is common to use battery powered devices on either side of the USB-C-Switch Pro. When these devices are not in the active path, the device may go into low-power or sleep mode, or its battery may discharge. The USB-C-Switch Pro has the unique feature of Keep-Alive Charging (KAC) for the Mux channel connections. KAC is only active when USB-C-Switch Pro is connected to its external DC power supply.



When KAC is enabled on an inactive Mux port, the port's V_{BUS} connects to the KAC circuit to provide 5 V 500 mA. The KAC circuit does not provide USB power-delivery (USB-PD), USB battery charge specification (BC1.2) or QuickCharge® to the selected ports. The KAC circuit has thermal and overcurrent protection and will stop providing power if limits are exceeded. KAC must be disabled and re-enabled to restore charging. KAC is automatically disabled when mux split mode is enabled.

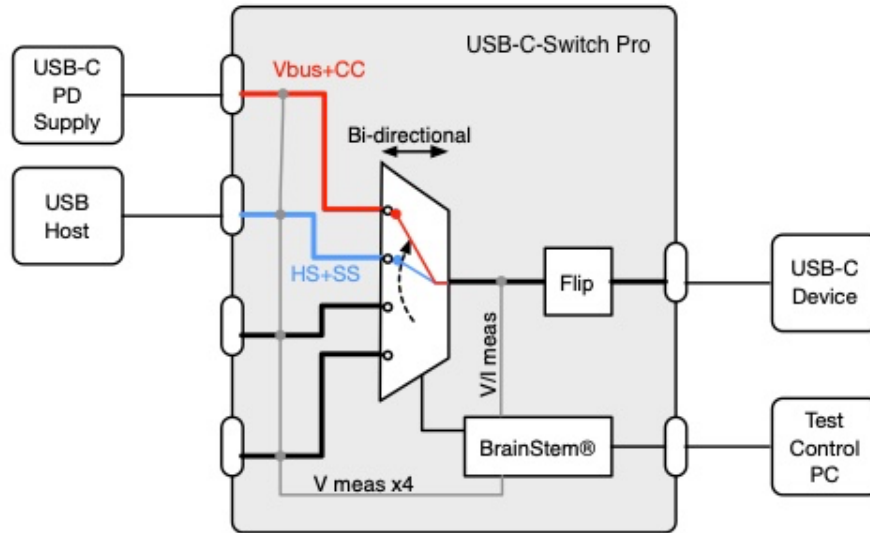
Mux Split Mode

The default behavior of the USB-C-Switch Pro is to act as a port selector, where all USB-C lines are connected between the common port and one selected mux channel. In some cases, it is desirable to split the connections in a USB-C cable and route them to different mux paths.

Split mode gives control over individual signal groups, allowing each group to be connect to a mux channel. V_{BUS} can be connected to any combination of mux channels or disabled on the mux channels.

Common application include:

- Providing a data connection between a USB device the host machine while connecting V_{BUS} charging from a device-specific charger on a different port.
- Splitting USB HS and SS lines to provide two independent device connections to a single host port.



Signal groups under Split control assignment are:

- V_{BUS}
- SS(TX 1/2 +/-, RX 1/2 +/-)
- HS (D+/-),
- CC1, CC2, SBU1, and SBU2

When split mode is enabled, V_{BUS} can be assigned to multiple Mux ports simultaneously, which is useful for powering multiple devices. However, Acroname recommends that V_{BUS} be assigned to only one mux channel. Caution should be used with multi-point V_{BUS} assignments as it is possible to apply a V_{BUS} voltage to a device that has not negotiated for high V_{BUS} voltages which could damage connected devices.

When split mode is enabled, USB-C-Switch Pro will automatically disable the Keep-Alive-Charging (KAC) feature.

Warning: Split mode can create connections and configurations not possible or compliant with standard USB equipment. Using this feature could cause unexpected voltages to be applied to devices which may damage connected equipment

Ethernet Control

The USB-C-Switch Pro can be managed over Ethernet using the HubTool application, BrainStem API, REST interface, or built-in web interface. Connections are made through the Ethernet jack using TCP/IP sockets and are supported on the local link segment only.

By default, the USB-C-Switch Pro acts as a DHCP client and will receive an IP address from a DHCP server. If no server is detected, the USB-C-Switch Pro falls back to a static IP address of **192.168.44.42**. In static mode, the host computer interface IP must be set to an address in the **192.168.44.x** range. The DHCP client is limited to hosts on the local link and does not operate across network bridges or gateways.

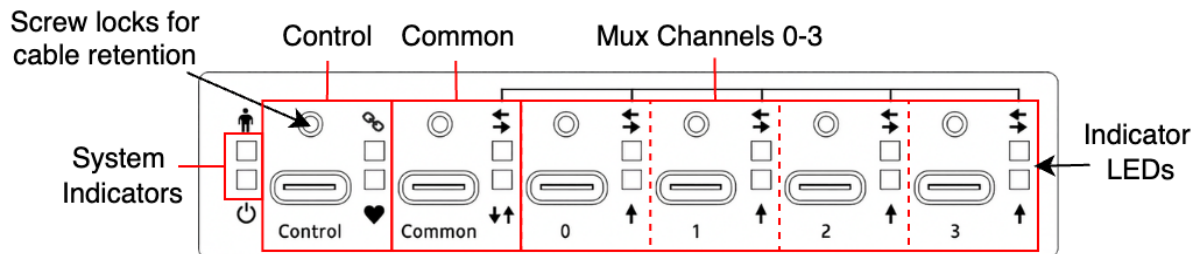
The USB-C-Switch Pro responds to ICMP “ping” requests including broadcast pings. The BrainStem API interface performs a discovery process prior to establishing communication by sending a UDP multicast request on port 9888. The USB-C-Switch Pro responds with a message to UDP port 9889. The USB-C-Switch Pro listens for socket connections on TCP port 8000. The Rest interface uses TCP on ports 9005 and 9006

Host firewall rules must allow:

- Outgoing UDP multicast on port 9888
- Incoming UDP responses on port 9889
- Outgoing TCP connections to port 8000
- Incoming / Outgoing TCP connections on ports 9005 and 9006

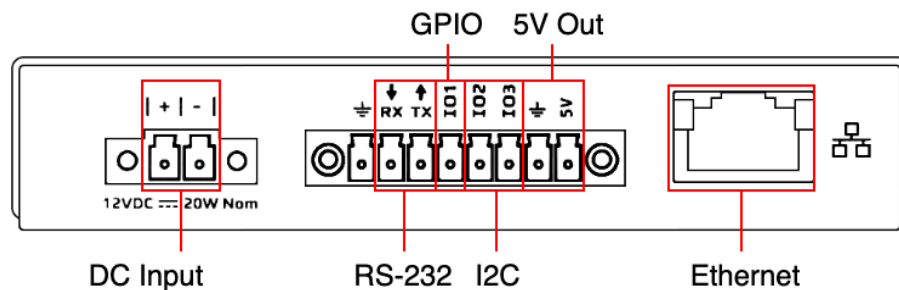
1.4.3 Indicators and Connections

Front Panel



The USB-C-Switch front panel contains all USB ports: Control, Common, and Mux 0-3, along with port and system LED indicators. The control port supports USB Type C power at 5 V, 3 A.

Rear Panel



The rear panel contains the Input Power Connector, Ethernet port, and the Expansion Connector.

Input Power Connector

Power for the USB-C-Switch Pro can be provided by the V_{BUS} line on the control port or by external power on the rear panel connection, V_{INPUT} . A Euro-style 2-pin terminal block (“Euroblock”) is used for the external power connection, with pin spacing of 3.81 mm (0.150”).

Expansion Connector

The USB-C-Switch expansion connector is Euro-style 8-pin terminal block (“Euroblock”) with pin spacing of 3.50 mm (0.138”). This interface provides additional mechanisms for expandability and test scenarios. Rail 0 is a software-controlled current-limited fixed 5 V source (disabled by default). IO1 is a general-purpose input and output that can also be configured as a selector to cycle through Mux channels.

Table 11: Expansion connector pinout

Connection Name	Pin Number	Description
GND	1	Ground
RX	2	RS-232 Serial Receive (data to USBCSwitchPro)
TX	3	RS-232 Serial Transmit (data from USBCSwitchPro)
IO1	4	General Purpose Input//Selector
IO2	5	I ² C SDA
IO3	6	I ² C SCL
GND	7	Ground
5V (Rail 0)	8	Current-limited 5 V source

LED Indicators

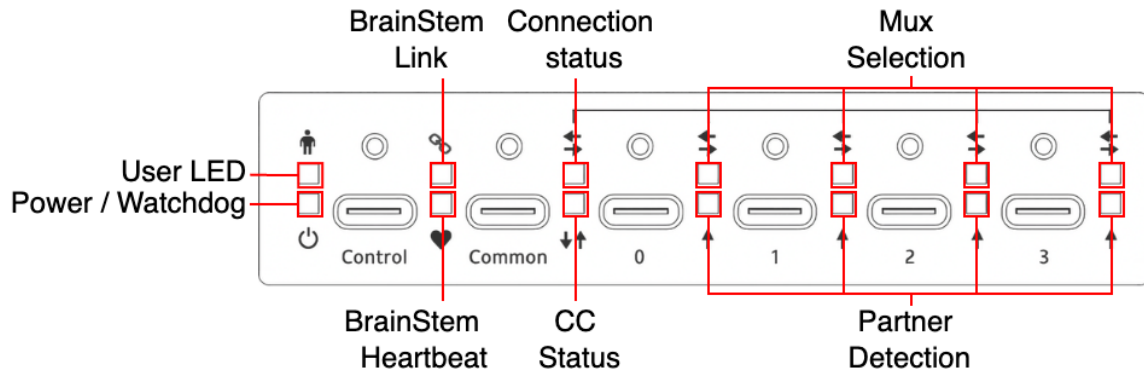























Table 12: LED Indicators and Function

Type	Icon	LED Name	Color	Description
System	 Person	User LED	 Blue	User-controllable LED
	 Power Icon.	Power/Watchdog	 Red /  Magenta	Alternating Red and Magenta when powered
Control	 Chain Links	BrainStem Link	 Yellow	Link present
	 Heart	BrainStem Heartbeat	 Green	Blink when Heartbeat received
Common	 Left/Right Arrow	Connection Status: Solid = connected	 Green	Unflipped
			 Yellow	Flipped
	 Up/Down Arrow	Partner detection	 Green	V _{BUS} present or PD negotiated
Mux Ports	 Left/Right Arrow	Mux selection: Solid = enabled Blinking = disabled	 Blue	Channel selected
			 Yellow	Split mode
	 Up Arrow	Partner Detection	 Green	V _{BUS} present or PD negotiated
Ethernet	Left	Ethernet Activity	 Green	Blinks with Ethernet activity
	Right	BrainStem Link	 Amber	BrainStem connection established

Power

The USB-C-Switch Pro can either be bus-powered via the Control Port, or through the 12 V DC input using the included power adapter.

Unit Reset

The USBC-Switch Pro can be reset to factory default settings using the reset button on bottom of the unit. Pressing the reset button once will restart the USBC-Switch Pro as if it had been power cycled. To restore factory default settings, press twice within 5 seconds.

1.4.4 Software Control

The USB-C-Switch Pro is part of a family of Acroname devices that share interface conventions and capabilities.

Acroname provides multi-language APIs for interacting with the USB-C-Switch Pro, with consistent syntax and structure across supported languages. The underlying protocol and API design is called BrainStem® (see [BrainStem](#)).

The API represents the the connection to a USB-C-Switch Pro as a Module class, an instance of which represents the connection to a specific USB-C-Switch Pro. This class provides access to hardware functions through a set of sub-objects called Entities, each representing a specific interface or capability of the device. Once instantiated and connected, the USB-C-Switch Pro can be controlled through its entity interfaces.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Example: Discovery and Connection

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    // Locate and connect to the first Acroname object you find
    aUSBCSwitchPro cswitch;
    auto err = cswitch.discoverAndConnect(USB);          // Over USB
    // auto err = cswitch.discoverAndConnect(TCPIP); // Over TCP/IP
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
        return 1;
    } else { printf("Connected to BrainStem module.\n"); }

    //Prep USBCSwitchPro for testing
    cswitch.usb.setPortDisable(0);
    cswitch.mux.setEnable(false);
    cswitch.mux.setChannel(0);

    ////////////
    //Do Stuff: other test initialization
    ////////////

    //Ready for testing
    //Enable Port AND Mux
    cswitch.usb.setPortEnable(0);
    cswitch.mux.setEnable(true);

    ////////////
    //Do Stuff on Mux Channel 0
    ////////////

    cswitch.mux.setChannel(1);

    ////////////
    //Do Stuff on Mux Channel 1
    ////////////

    cswitch.mux.setChannel(2);

    ////////////
    //Do Stuff on Mux Channel 2
    ////////////

    cswitch.mux.setChannel(3);

    ////////////
    //Do Stuff on Mux Channel 3
    ////////////

    //Finished with testing.
    //De-initialize.
    cswitch.usb.setPortDisable(0);
```

(continues on next page)

(continued from previous page)

```

cswitch.mux.setEnabled(false);

//Disconnect
cswitch.disconnect();
}

```

Python

```

import brainstem
# For easy access to error constants
from brainstem.result import Result
from time import sleep
import sys

# Create an instance of a USBSwitchPro module.
cswitch = brainstem.stem.USBSwitchPro()

# Locate and connect to the first object you find
result = cswitch.discoverAndConnect(brainstem.link.Spec.USB) # Over USB

# result = cswitch.discoverAndConnect(brainstem.link.Spec.TCPIP) # Over TCP/IP

if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % (result))
    sys.exit(1)
else:
    print ("Connected to BrainStem module.\n")

##Prep USBSwitchPro for testing
cswitch.usb.setPortDisable(0)
cswitch.mux.setEnabled(False)
cswitch.mux.setChannel(0)

#####
## Do Stuff: other test initialization
#####

##Ready for testing
##Enable Port AND Mux
cswitch.usb.setPortEnable(0)
cswitch.mux.setEnabled(True)

#####
##Do Stuff on Mux Channel 0
#####

cswitch.mux.setChannel(1)

#####
##Do Stuff on Mux Channel 1
#####

cswitch.mux.setChannel(2)

#####
##Do Stuff on Mux Channel 2

```

(continues on next page)

(continued from previous page)

```
#####  
  
cswitch.mux.setChannel(3)  
  
#####  
##Do Stuff on Mux Channel 3  
#####  
  
#Finished with testing.  
#De-initialize.  
cswitch.usb.setPortDisable(0)  
cswitch.mux.setEnable(False)  
  
# Disconnect from device.  
cswitch.disconnect()
```

Device Drivers

The USB-C-Switch Pro leverages common operating system drivers that do not require custom installations on modern operating systems.

Some older operating systems may require the installation of a BrainStem USB driver information files to enable software control. Installation details on installing USB drivers can be found within the BrainStem Development Kit under the “drivers” folder. For example, Windows 7 requires the supplied INF to communicate with BrainStem USB devices.

1.4.5 USB-C-Switch Pro Software Features

To enable a newly-licensed software feature, it is necessary to *update the firmware* of the USB-C-Switch Pro even if you are at on the current version.

PD Logging

The PD Logging feature allows you to monitor USB-PD communication between devices on the Common and Mux ports of the USB-C-Switch Pro, enabling message logging, decoding, and injection.

Features

- Capture and decode PD events
- Export message logs to CSV
- Filter traffic by message type
- Clearly show message direction

Applications

- Device debugging and bringup
- USB PD device testing

HubTool

With HubTool you can easily visualize the Power Delivery log.

The screenshot shows the HubTool application window. At the top, 'System Information' displays: SN: 0x4F7A0C15, Voltage: 20.0 VDC, Hardware Offset: 0, Router: 6, Model: 24, Current: 0.5 A, SW Offset: 0, Firmware: 2.9.6, Temperature: 30 C, and Module: 6. Below this are tabs for Summary, Port 0, Port 2, Port 3, Port 4, Port 5, Control, Power C, IO Expander, Power, and PD Logging. The 'PD Logging' tab is active, showing a table of power delivery events. The table has columns: Timestamp (S:uS), Port, Direction, Spec, SOP, Power Role, Data Role, ID, Packet Type, Msg Type, and Raw. It lists 25 events from 1429:635630 to 1430:919090. At the bottom, there are checkboxes for various software features (Rail Enable, QC Mode, PDO Editing, VBUS Validation, PD Logging) and a status bar showing USBHub3c: 0x4F7A0C15, Error: 7 CMD: stem.hub.port[x].setVoltageSetpoint.

	Timestamp (S:uS)	Port	Direction	Spec	SOP	Power Role	Data Role	ID	Packet Type	Msg Type	Raw
1	1429:635630	1	RX	V2.0	SOP	Source	DFP	0	Data	Source Capabilities	0x61 0x11 0x96 0x90 0x01 0x36
2	1429:645760	1	TX	V2.0	SOP	Sink	UFP	0	Data	Request	0x42 0x10 0x64 0x90 0x01 0x10
3	1429:654670	1	RX	V2.0	SOP	Source	DFP	1	Control	Accept	0x63 0x03
4	1429:680910	1	RX	V2.0	SOP	Source	DFP	2	Control	PS Ready	0x66 0x05
5	1429:690560	1	RX	V2.0	SOP	Source	DFP	3	Data	Vendor Defined	0x6F 0x17 0x01 0x80 0x00 0xFF
6	1429:701040	1	TX	V2.0	SOP	Sink	UFP	1	Data	Vendor Defined	0x4F 0x72 0x41 0x80 0x00 0xFF 0xFF 0x24 0xA...
7	1429:707240	1	RX	V2.0	SOP	Source	DFP	4	Data	Vendor Defined	0x6F 0x19 0x02 0x80 0x00 0xFF
8	1429:712110	1	TX	V2.0	SOP	Sink	UFP	2	Data	Vendor Defined	0x4F 0x24 0x42 0x80 0x00 0xFF 0x00 0x00 0xF...
9	1429:717770	1	RX	V2.0	SOP	Source	DFP	5	Data	Vendor Defined	0x6F 0x1B 0x02 0x80 0x00 0xFF
10	1429:724220	1	TX	V2.0	SOP	Sink	UFP	3	Data	Vendor Defined	0x4F 0x26 0x42 0x80 0x00 0xFF 0x00 0x00 0xF...
11	1429:948940	1	TX	V2.0	SOP	Sink	UFP	4	Control	VConn Swap	0x4B 0x08
12	1429:957240	1	RX	V2.0	SOP	Source	DFP	6	Control	Accept	0x63 0x0D
13	1430:18980	1	TX	V2.0	SOP	Sink	UFP	5	Control	PS Ready	0x46 0x0A
14	1430:72450	1	TX	V2.0	S...	Sink	UFP	0	Data	Vendor Defined	0x4F 0x10 0x01 0x80 0x00 0xFF
15	1430:158980	1	TX	V2.0	S...	Sink	UFP	1	Data	Vendor Defined	0x4F 0x12 0x01 0x80 0x00 0xFF
16	1430:168330	1	RX	V2.0	S...	Source	UFP	1	Data	Vendor Defined	0x4F 0x53 0x41 0x80 0x00 0xFF 0x11 0x07 0x00...
17	1430:262130	1	TX	V2.0	SOP	Sink	UFP	6	Control	DR Swap	0x49 0x0C
18	1430:269210	1	RX	V2.0	SOP	Source	DFP	7	Control	Accept	0x63 0x0F
19	1430:515440	1	TX	V2.0	SOP	Sink	DFP	7	Control	PR Swap	0x6A 0x0E
20	1430:523230	1	RX	V2.0	SOP	Source	UFP	0	Control	Accept	0x43 0x01
21	1430:625040	1	RX	V2.0	SOP	Sink	UFP	1	Control	PS Ready	0x46 0x02
22	1430:741540	1	TX	V2.0	SOP	Source	DFP	0	Control	PS Ready	0x66 0x01
23	1430:903350	1	TX	V2.0	SOP	Source	DFP	0	Data	Source Capabilities	0x61 0x51 0x2C 0x91 0x01 0x3E 0x2C 0xD1 0x02...
24	1430:915290	1	RX	V2.0	SOP	Sink	UFP	0	Data	Request	0x42 0x10 0x2C 0xB1 0x04 0x13
25	1430:919090	1	TX	V2.0	SOP	Source	DFP	1	Control	Accept	0x63 0x03

Software Feature: Rail Enable: Enabled
 Software Feature: QC Mode: Enabled
 Software Feature: PDO Editing: Enabled
 Software Feature: VBUS Validation: Enabled
 Software Feature: PD Logging: Enabled
 USBHub3c: 0x4F7A0C15, Error: 7 CMD: stem.hub.port[x].setVoltageSetpoint

Device Type: | Serial Number:
 USBHub3c 4F7A0C15
 USBHub3p AF62130D

Relevant APIs

```
stem.pd[x].setUEIBytes() [cpp] [python]
```

RS232 Serial Communication

The RS232 Serial Communication feature allows one to send commands to control and configure USB-C-Switch Pro.

Use Cases

- Affecting USB-C-Switch Pro control.
- Audio/Video applications.

Configuration

The default configuration of the RS232 Serial Communication feature is:

- 8 Data bits
- No Parity
- No Flow Control
- 1 Stop bit
- 9600 Baudrate

This feature does have some configurability through the USB-C-Switch Pro *UART Entity*

Extron Compatible Serial Commands

Using a protocol compatible with Extron's *Simple Instruction Set* over RS232, the RS232 Serial Communication feature can:

- Select / Enable / Disable Mux ports
- Query the USB-C-Switch Pro part number and firmware version.

Commands

The following is a list of all commands the USB-C-Switch Pro supports with their arguments, descriptions, and expected responses. For USB-C-Switch Pro, "Upstream Port" refers to the Mux port index independent of the port direction.

Cmd	Arguments	Description	Expected Responses
!	None	Get current mux port index	Chn #\r\n
# !	# Port	Change mux port to # port number	Chn #\r\n
# ^	# Port	Change mux port to # port number	Chn #\r\n
N	None	Get part number	S105-USB-C-SWITCH-PRO\r\n
Q	None	Get firmware version	<M>.<m>.<p>\r\n
# P	# Port	Get enable/disable status of # port number	Port #*0\r\n Port #*1\r\n
# * \$ P	# Port \$ Enable 0/1	Set \$ enable/disable of # port number	Port #*\$\r\n

Error Codes

The following is a list of all error codes the USB-C-Switch Pro supports with descriptions.

Code	Description
E01	invalid port number, check the port number and make sure it's valid.
E10	invalid command, verify that you formatted the command correctly.
E13	invalid value, verify that the value is within the acceptable range for this command.
E14	invalid configuration, verify the system is in a state it can accept this command.

General Notes

All commands are ASCII strings.

\r is the ASCII character for carriage return.

\n is the ASCII character for new line.

Examples

Extron Compatible Serial Commands:

Mux Port Selection

Select Mux Port 1

Tx: 1!

Rx: Chn 1\r\n

Port Disable

Disable Port 3

Tx: 3*0P

Rx: Port 3*0\r\n

API Configurations:

C++

```
static const int TEST_SERIAL = 0;

aUSBCSwitchPro stem;
stem.discoverAndConnect(USB);

// Enable the port
stem.uart[TEST_SERIAL].setEnabled(true);
```

(continues on next page)

(continued from previous page)

```
// Change baudrate to 115200 from default 9600
stem.uart[TEST_SERIAL].setBaudRate(115200);

// Change Protocol to Extron Compatible
// 0 - Disabled/Undefined
// 1 - Extron Compatible
// 2 - Brainstem Transport
// 6 - Loopback
stem.uart[TEST_SERIAL].setProtocol(1)

// Perform a system save so the changes persist
// through power cycles
stem.system.save();

stem.disconnect();
```

Python

```
TEST_SERIAL = 0

stem = brainstem.stem.USBCSwitchPro()
stem.discoverAndConnect(brainstem.link.Spec.USB)

# Enable the port
stem.uart[TEST_SERIAL].setEnabled(1)

# Change baudrate to 115200 from default 9600
stem.uart[TEST_SERIAL].setBaudRate(115200)

# Change Protocol to Extron Compatible
# 0 - Disabled/Undefined
# 1 - Extron Compatible
# 2 - Brainstem Transport
# 6 - Loopback
stem.uart[TEST_SERIAL].setProtocol(1)

# Perform a system save so the changes persist
# through power cycles
stem.system.save()

stem.disconnect()
```

Loopback Protocol

```
# Configure UART[0] for loopback testing
# Data sent to the UART will be echoed back
TEST_SERIAL = 0

stem = brainstem.stem.USBCSwitchPro()
stem.discoverAndConnect(brainstem.link.Spec.USB)

stem.uart[TEST_SERIAL].setEnabled(1)
stem.uart[TEST_SERIAL].setBaudRate(115200)
stem.uart[TEST_SERIAL].setProtocol(6) # Loopback protocol
```

(continues on next page)

(continued from previous page)

```
# Now any data sent to UART[0] will be echoed back
stem.system.save()
stem.disconnect()
```

VCOM to Hardware UART Link

```
# Link VCOM channel (UART[1]) to hardware RS232 (UART[0])
# This allows USB serial communication to be forwarded to RS232
stem = brainstem.stem.USBCSwitchPro()
stem.discoverAndConnect(brainstem.link.Spec.USB)

# Enable both channels
stem.uart[0].setEnabled(1)    # Hardware RS232
stem.uart[1].setEnabled(1)    # VCOM channel

# Configure matching baud rates
stem.uart[0].setBaudRate(115200)
stem.uart[1].setBaudRate(115200)

# Link VCOM to hardware UART
stem.uart[0].setLinkChannel(1)
stem.uart[1].setLinkChannel(0)

# Data received on VCOM will now be forwarded to RS232,
# and data received on RS232 will be forwarded to VCOM
stem.system.save()
stem.disconnect()
```

Loopback Protocol

```
// Configure UART[0] for loopback testing
// Data sent to the UART will be echoed back
static const int TEST_SERIAL = 0;

aUSBCSwitchPro stem;
stem.discoverAndConnect(USB);

stem.uart[TEST_SERIAL].setEnabled(true);
stem.uart[TEST_SERIAL].setBaudRate(115200);
stem.uart[TEST_SERIAL].setProtocol(6); // Loopback protocol

// Now any data sent to UART[0] will be echoed back
stem.system.save();
stem.disconnect();
```

VCOM to Hardware UART Link

```
// Link VCOM channel (UART[1]) to hardware RS232 (UART[0])
// This allows USB serial communication to be forwarded to RS232
aUSBCSwitchPro stem;
stem.discoverAndConnect(USB);

// Enable both channels
stem.uart[0].setEnabled(true); // Hardware RS232
stem.uart[1].setEnabled(true); // VCOM channel
```

(continues on next page)

(continued from previous page)

```
// Configure matching baud rates
stem.uart[0].setBaudRate(115200);
stem.uart[1].setBaudRate(115200);

// Link VCOM to hardware UART
stem.uart[0].setLinkChannel(1);
stem.uart[1].setLinkChannel(0);

// Data received on VCOM will now be forwarded to RS232,
// and data received on RS232 will be forwarded to VCOM
stem.system.save();
stem.disconnect();
```

Relevant API's

```
stem.uart[x].setEnabled() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getEnabled() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setBaudRate() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getBaudRate() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setProtocol() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getProtocol() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setLinkChannel() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getLinkChannel() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setStopBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getStopBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setParity() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getParity() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setDataBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getDataBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getCapableProtocols() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getAvailableProtocols() [cpp] [python] [NET] [LabVIEW]
```

1.4.6 USB-C-Switch Pro Module Entities

Ethernet Entity

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

The Ethernet Entity provides control over network configuration and monitoring for devices with Ethernet connectivity. This includes IP address management, network configuration modes, and interface settings.

Ethernet Enable/Disable (Get/Set)

```

ethernet . getEnabled <= (unsigned char) enabled
ethernet . setEnabled => (unsigned char) enabled

```

Provides control (Set) and monitoring (Get) over the Ethernet interface. When enabled, the Ethernet interface is active and can be used for network communication.

Network Configuration (Get/Set)

```

ethernet . getNetworkConfiguration <= (unsigned char) configuration
ethernet . setNetworkConfiguration => (unsigned char) configuration

```

Returns or sets the network configuration mode for the Ethernet interface. This controls whether the interface uses DHCP or static IP configuration.

Table 13: Network Configuration Options

Value	Name	Description
0	None	No network configuration
1	Static	Static IP configuration
2	DHCP	DHCP automatic configuration

Static IPv4 Address (Get/Set)

```

ethernet . getStaticIPv4Address <= (unsigned char[]) address
ethernet . setStaticIPv4Address => (unsigned char[]) address

```

Returns or sets the static IPv4 address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the IP address (e.g., [192, 168, 1, 100] for “192.168.1.100”).

Static IPv4 Netmask (Get/Set)

```

ethernet . getStaticIPv4Netmask <= (unsigned char[]) netmask
ethernet . setStaticIPv4Netmask => (unsigned char[]) netmask

```

Returns or sets the static IPv4 netmask for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the netmask (e.g., [255, 255, 255, 0] for “255.255.255.0”).

Static IPv4 Gateway (Get/Set)

```
ethernet . getStaticIPv4Gateway <= (unsigned char[]) gateway  
ethernet . setStaticIPv4Gateway => (unsigned char[]) gateway
```

Returns or sets the static IPv4 gateway address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the gateway address (e.g., [192, 168, 1, 1] for “192.168.1.1”).

IPv4 Address (Get)

```
ethernet . getIPv4Address <= (unsigned char[]) address
```

Returns the current IPv4 address of the Ethernet interface as a 4-byte array. This reflects the actual address being used, whether obtained via DHCP or set statically. Each byte represents one octet of the IP address.

IPv4 Netmask (Get)

```
ethernet . getIPv4Netmask <= (unsigned char[]) netmask
```

Returns the current IPv4 netmask of the Ethernet interface as a 4-byte array. This reflects the actual netmask being used, whether obtained via DHCP or set statically. Each byte represents one octet of the netmask.

IPv4 Gateway (Get)

```
ethernet . getIPv4Gateway <= (unsigned char[]) gateway
```

Returns the current IPv4 gateway address of the Ethernet interface as a 4-byte array. This reflects the actual gateway being used, whether obtained via DHCP or set statically. Each byte represents one octet of the gateway address.

Static IPv4 DNS Address (Get/Set)

```
ethernet . getStaticIPv4DNSAddress <= (unsigned char[]) dns  
ethernet . setStaticIPv4DNSAddress => (unsigned char[]) dns
```

Returns or sets the static IPv4 DNS server address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the DNS address.

IPv4 DNS Address (Get)

```
ethernet . getIPv4DNSAddress <= (unsigned char[]) dns
```

Returns the current IPv4 DNS server address of the Ethernet interface as a 4-byte array. This reflects the actual DNS server being used, whether obtained via DHCP or set statically. Each byte represents one octet of the DNS address.

Hostname (Get/Set)

```
ethernet . getHostname <= (unsigned char[]) hostname
ethernet . setHostname => (unsigned char[]) hostname
```

Returns or sets the hostname for the Ethernet interface. The hostname is used for network identification and can be up to 63 characters long.

MAC Address (Get)

```
ethernet . getMACAddress <= (unsigned char[]) mac
```

Returns the MAC address of the Ethernet interface. This is a unique hardware identifier for the network interface.

Interface Port (Get/Set)

```
ethernet . getInterfacePort (unsigned char) service <= (unsigned short) port
ethernet . setInterfacePort => (unsigned char) service, (unsigned short) port
```

Returns or sets the interface port for specific services on the Ethernet interface. The service parameter specifies which service to configure, and the port parameter sets the port number for that service.

Table 14: Interface Port Services

Name	Value	Description
RestServer_HTTP	0	HTTP REST API server
RestServer_HTTPS	1	HTTPS REST API server
BrainStem_TCP	2	BrainStem TCP communication
Brain-Stem_DiscoveryRequest	3	BrainStem discovery request port
Brain-Stem_DiscoveryReply	4	BrainStem discovery reply port

Examples

C++

```
// Enable Ethernet interface
ethernet.setEnabled(1);

// Set static IP configuration
ethernet.setNetworkConfiguration(1); // Static mode
unsigned char ip[4] = {192, 168, 1, 100};
unsigned char netmask[4] = {255, 255, 255, 0};
unsigned char gateway[4] = {192, 168, 1, 1};
ethernet.setStaticIPv4Address(ip, 4);
ethernet.setStaticIPv4Netmask(netmask, 4);
ethernet.setStaticIPv4Gateway(gateway, 4);

// Get current IP address
unsigned char current_ip[4];
ethernet.getIPv4Address(current_ip, 4);

// Set hostname
ethernet.setHostname("brainstem-device");
```

Python

```
# Enable Ethernet interface
stem.ethernet.setEnabled(1)

# Set static IP configuration
stem.ethernet.setNetworkConfiguration(1) # Static mode
ip = [192, 168, 1, 100]
netmask = [255, 255, 255, 0]
gateway = [192, 168, 1, 1]
stem.ethernet.setStaticIPv4Address(ip)
stem.ethernet.setStaticIPv4Netmask(netmask)
stem.ethernet.setStaticIPv4Gateway(gateway)

# Get current IP address
current_ip = stem.ethernet.getIPv4Address()
print(current_ip.value)

# Set hostname
stem.ethernet.setHostname("brainstem-device")
```

I²C Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read

```
i2c [ index ] . read => (unsigned char) address, (unsigned char) length <= (unsigned_
↳char*) data
```

Reads up to 26 bytes from the I²C bus given by the index. The parameters are the I²C address of the device on the bus, and the number of bytes to read. The result is the data that was read or an error.

Write

```
i2c [ index ] . write => (unsigned char) address, (unsigned char) length, (unsigned_
↳char*) data <= (unsigned char) result
```

Writes up to 26 bytes to the I²C bus given by the index. The parameters are the I²C address of the device on the bus, the number of bytes to write, and the data to write. The result is the result error condition or none.

Set Pullup

```
i2c [ index ] . setPullup => (unsigned char) bool
```

Sets software controlled pullup state on modules which have software controllable pullups. This setting is saved when a call to `system.save` is made so that Pullup settings on bus 0 can persist.

Speed (Get/Set)

```
i2c [ index ] . getSpeed <= (unsigned char) speed
i2c [ index ] . setSpeed => (unsigned char) speed
```

Returns or sets the I2C bus communication speed. The speed setting controls the clock frequency for I2C transactions on the specified bus.

Table 15: I2C Speed Options

Value	Name	Description
0	Default	Default I2C speed (typically 100 kHz)
1	100Khz	Standard I2C speed (100 kHz)
2	400Khz	Fast I2C speed (400 kHz)
3	1000Khz	Fast Plus I2C speed (1 MHz)

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.
char buff[2];
stem.i2c[0].read(0x42, 0x02, buff); // reads two from device with address 0x42.
char wrbuff[] = {0xBE, 0xEF};
stem.i2c[0].write(0x42, 0x02, wrbuff); // writes 0xBEEF to the device with address_
↳0x42
stem.i2c[0].setPullup(true); //enables pullup on bus 0
stem.i2c[0].setSpeed(1); // sets I2C speed to 100 kHz (i2cSpeed_100Khz)
stem.i2c[0].setSpeed(2); // sets I2C speed to 400 kHz (i2cSpeed_400Khz)
stem.i2c[0].setSpeed(3); // sets I2C speed to 1 MHz (i2cSpeed_1000Khz)
unsigned char speed;
stem.i2c[0].getSpeed(&speed); // gets current I2C speed
```

Python

The length parameter for I²C write is not used in python.

```
result = stem.i2c[0].read(0x42, 0x02) # reads two bytes from the i2c bus. The value_
↳is given in result.value
print result.value
err = stem.i2c[0].write(0x42, b'\xbe\xef') # writes b'\xbe\xef' to the i2c bus.
print err
err = stem.i2c[0].setPullup(True)
print err
err = stem.i2c[0].setSpeed(1) # sets I2C speed to 100 kHz (i2cSpeed_100Khz)
print err
speed = stem.i2c[0].getSpeed() # gets current I2C speed
print speed.value
```

Mux Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs.

Some mux entities can simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

Mux Channel

The mux entity primarily selects one active mux port to connect to the common port using the channel option:

```
stem.mux.setChannel(channel) [cpp] [python] [NET] [LabVIEW] [REST]
```

```
stem.mux.getChannel(channel) [cpp] [python] [NET] [LabVIEW] [REST]
```

where (channel) is an index 0-3.

Mux Configuration

Default configuration of the mux is to switch all enabled USB-C lines to a single mux channel. If desired, the switch can split the USB-C functional groups and route them to selected mux ports. This feature is referred to as “split mode”. Default and split mode can be enabled with:

```
stem.mux.getConfiguration(config) [cpp] [python] [NET] [LabVIEW] [REST]
```

```
stem.mux.setConfiguration(config) [cpp] [python] [NET] [LabVIEW] [REST]
```

where (config) is 0 for default, 1 for Split Mode.

Split Mode

After enabling split mode the USB-C functional groups can be individually assigned to separate mux channels with:

```
stem.mux.getSplitMode(splitMode) [cpp] [python] [NET] [LabVIEW] [REST]
```

```
stem.mux.setSplitMode(splitMode) [cpp] [python] [NET] [LabVIEW] [REST]
```

where (splitMode) is a 32-bit word, defined below. Each bit pair is a 2-bit binary number from 0-3 representing the mux port to which to route the functional signal group. V_{BUS} and CC use 4-bits to define which mux ports are connected to the common port V_{BUS} /CC lines.

Bit	Mux Split Mode Bit Map
0:1	SBU1
2:3	SBU2
4	CC1 enable CH0
5	CC1 enable CH1
6	CC1 enable CH2
7	CC1 enable CH3
8	CC2 enable CH0
9	CC2 enable CH1
10	CC2 enable CH2
11	CC2 enable CH3
12:13	HS Data
14:15	Reserved
16:17	SS Data
18:19	Reserved
20	V _{BUS} enable CH0
21	V _{BUS} enable CH1
22	V _{BUS} enable CH2
23	V _{BUS} enable CH3
24:31	Reserved

Port Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Port Entity provides control over the most basic items related to a USB Port. This includes actions ranging from a complete port enable and disable to the individual interface control. Voltage and current measurements are also included for devices which support the Port Entity.

- Cable flip
- Alt mode configuration
- Error clearing
- Connect mode
- SBU enable/disable

Ports are referenced by their instance index `port[x]` as shown below.

Port Label	Index (port[x])
Mux 0-3	0-3
Control	4
Common	5

Data and Power Control (Common Port)

```
stem.hub.port[x].setEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enable or disable the entire port, including both data and power lines.

```
stem.hub.port[x].setPowerEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getPowerEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enable or disable V_{BUS} power only (independent of data path).

```
stem.hub.port[x].setDataEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enable or disable USB data lines (both USB 2.0 Hi-Speed and USB 3.x SuperSpeed) without affecting V_{BUS} .

```
stem.hub.port[x].setDataHSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataHSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enable or disable only the USB 2.0 Hi-Speed data lines.

```
stem.hub.port[x].setDataSSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataSSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enable or disable only the USB 3.x SuperSpeed lanes.

```
stem.hub.port[x].setCCEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCCEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enable or disable both CC lines simultaneously.

```
stem.hub.port[x].setCC1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

```
stem.hub.port[x].setCC2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enable or disable individual CC1 / CC2 pins.

Electrical Measurements (Common, Mux, and Control Ports)

Voltage (μV)

```
stem.hub.port[x].getVbusVoltage() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconnVoltage() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getSBU1Voltage() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getSBU2Voltage() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC1Voltage() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC2Voltage() [cpp] [python] [NET] [LabVIEW] [REST]
```

Current (μA)

```
stem.hub.port[x].getVbusCurrent() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconnCurrent() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC1Current() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC2Current() [cpp] [python] [NET] [LabVIEW] [REST]
```

Note: SBU1 / SBU2 expose **voltage** only.

Port Mode

```
stem.hub.port[x].setMode() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getMode() [cpp] [python] [NET] [LabVIEW] [REST]
```

Port Mode provides a higher-level shortcut that bundles these controls into a single bit-mapped configuration, allowing you to enable or disable groups of lines and set routing behaviors in one operation.

Bit	Port Mode Bit Map
0	Reserved
1	Reserved
2	Keep Alive Charging Enable
3	Reserved
4	HS Data enable
5	Reserved
6	V _{BUS} enable
7	SS Data enable
8:11	Reserved
12	CC1 enable
13	CC2 enable
14	SBU enable
15	CC Flip enable
16	Super-Speed Flip enable
17	SBU Flip enable
18	Hi-Speed Flip enable
19:31	Reserved

Accumulated Power

```
stem.hub.port[x].getVbusAccumulatedPower() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconnAccumulatedPower() [cpp] [python] [NET] [LabVIEW] [REST]
```

Reads total accumulated energy (mWh) sourced or sunk on V_{BUS} and V_{CONN}.

```
stem.hub.port[x].resetVbusAccumulatedPower() [cpp] [python] [NET] [LabVIEW]
[REST]
stem.hub.port[x].resetVconnAccumulatedPower() [cpp] [python] [NET] [LabVIEW]
[REST]
```

Resets accumulated energy and measurement interval.

Power Delivery Entity

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface. This Entity will allow you to directly access the vast landscape of PD.

When the capabilities of a PD system are fully realized everything in the system is “smart”. That includes the device, the host and even the cable. All of these elements contain electronics that identify themselves and what they are capable of doing. Because of this complexity it is important to align on a few terms that will be used throughout this Entity.

Partner This refers to the side of the PD connection in question. The possible options for this parameter are.

- **Local** Indicates the context/perspective of the Acroname device you are communicating with through a BrainStem connection.
- **Remote** The context/perspective of anything other than the Acroname device.

Partner Type	Value	Define
Local	0	powerdeliveryPartnerLocal
Remote	1	powerdeliveryPartnerRemote

Power Role Indicates the direction of power. This value is typically used in the context of a “Partner”. i.e. The remote partner is sinking, which would mean the local partner is sourcing. The possible options for this context are:

- **Sink** Indicates that the partner is taking power in/from.
- **Source** Indicates that the partner is providing power out/to.

Power Roles are also used in the context of what a port is capable of doing.

- **Sink** Device is capable of consuming power.
- **Source** Device is capable of producing power.
- **Sink/Source** Device is capable of both consuming or producing power. Dual Role Port (DRP)

Power Role	Value	Define
Disabled	0	powerdeliveryPowerRoleDisabled
Source	1	powerdeliveryPowerRoleSource
Sink	2	powerdeliveryPowerRoleSink
Source/Sink	3	powerdeliveryPowerRoleSourceSink

Power Data Objects (PDO)

- PDO's define what a device is capable of doing in the world of Power Delivery. PDO's are bit packed integers defined by the PD Specification which vary in meaning based on the type of PDO.

Request Data Objects (RDO)

- RDO's are the final agreement after successful Power Delivery negotiations. This RDO is always sent by the sinking device and is the result of the sources advertised PDO's and the needs/requirements of the sinking device. Only one RDO exists per valid connection.

PD Logging and Events

The USB-C-Switch Pro supports Power Delivery (PD) logging and event monitoring on the Common port (port 4). This allows you to monitor and capture PD communication for debugging and analysis purposes.

PD Logging Enable/Disable

PD logging is enabled or disabled for the Common port using the streaming interface. When enabled, PD packets and events are streamed as notifications.

PD Log Packet (Streaming)

When PD logging is enabled, PD packets are streamed as notifications. Each packet contains the PD message data, direction, timestamp, and other metadata.

The PD log packet streaming uses option code `powerdeliveryLogPacket` (46) and provides real-time access to PD communication packets.

PD Log Event (Streaming)

PD events are streamed as notifications when they occur. Events include connection/disconnection, resets, VBUS/VCONN state changes, and other PD protocol events.

The PD log event streaming uses option code `powerdeliveryLogEvent` (47). Available event types include:

Table 16: PD Event Types

Value	Event Type
0	None
1	Packet
2	Connect
3	Disconnect
4	Cable Reset Received
5	Cable Reset Sent
6	Hard Reset Received
7	Hard Reset Sent
8	Message Transmit Failed
9	Message Transmit Discarded
10	PD Function Disabled
11	VBUS Enabled
12	VBUS Disabled
13	VCONN Enabled
14	VCONN Disabled
15	Rp1A5 (Source Atomic Message Sequences)
16	Rp3A0 (Source Atomic Message Sequences)
17	BIST Enter
18	BIST Exit

For more information on using PD logging, see the [PD Logging software feature](#).

Rail

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Rail entity provides power control to connected devices on some modules. Check the module datasheet to determine if the module has this capability.

the Rail entity controls power provided to downstream devices, it has the ability to enable and disable power, can read voltage on the rail, and provides current consumption information on some modules. There are additional capabilities that certain modules provide which enhance basic power delivery through Kelvin sensing, or by bringing online separate power management functionality.

Certain modules may provide more than one power rail. These are independently controlled and can be accessed via the entity index.

Rail Control

The USB-C-Switch Pro provides a fixed 5V power output rail that can be enabled or disabled. The rail provides voltage and current monitoring capabilities for the external power output.

Rail Enable/Disable

The rail can be enabled or disabled using the following APIs:

```
stem.rail[0].setEnabled() \[cpp\] \[python\] \[.NET\] \[LabVIEW\] \[REST\]  
stem.rail[0].getEnable() \[cpp\] \[python\] \[.NET\] \[LabVIEW\] \[REST\]
```

Voltage and Current Monitoring

The rail provides real-time voltage and current measurements:

```
stem.rail[0].getVoltage( $\mu$ V) \[cpp\] \[python\] \[.NET\] \[LabVIEW\] \[REST\]  
stem.rail[0].getCurrent( $\mu$ A) \[cpp\] \[python\] \[.NET\] \[LabVIEW\] \[REST\]  
stem.rail[0].getPower(mW) \[cpp\] \[python\] \[.NET\] \[LabVIEW\] \[REST\]
```

where voltage is in microvolts (μ V), current is in microamperes (μ A), and power is in milliwatts (mW).

The USB-C-Switch Pro has one rail (index 0) that provides a fixed 5V output with monitoring capabilities.

System Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every USB-C-Switch Pro is assigned a unique serial number at the factory. This facilitates an arbitrary number of USB-C-Switch Pro devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) \[cpp\] \[python\] \[NET\] \[LabVIEW\] \[REST\]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the USB-C-Switch Pro away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. Use the following command to save changes to system settings before reboot:

```
stem.system.save() \[cpp\] \[python\] \[NET\] \[LabVIEW\] \[REST\]
```

Table 17: Saved Parameters

Entity	Parameters
Ethernet Entity	Network Configuration (DHCP/Static) Static IP Address Static IP Netmask Static IP Gateway Static IP DNS BrainD Port
Mux Entity	Enable Channel Configuration (Normal/Split Mode) Split Mode Settings
Port Entity	Name
Rail Entity	Enable
System Entity	Module Software Offset Address Router Address Name LED Brightness
UART Entity	Enable Baud Rate Protocol
USB System Entity	Selector Mode

Temperature

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Certain modules have a temperature measurement available. The temperature entity gives access to these measurements. Check your module datasheet to see if your module has a temperature entity.

System Temperature

The temperature of the USB-C-Switch Pro can be measured with:

```
stem.temperature[x].getValue(μC) \[cpp\] \[python\] \[.NET\] \[LabVIEW\] \[REST\]
```

```
stem.temperature[x].getValueMin(μC) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]
```

```
stem.temperature[x].getValueMax(μC) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]
```

where temperature is in micro-degrees Celsius.

The USB-C-Switch Pro has 5 temperature sensors:

Index	Sensor
0	Channel 0 temperature sensor
1	Channel 1 temperature sensor
2	Channel 2 temperature sensor
3	Channel 3 temperature sensor
4	Common temperature sensor

UART Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The UART entity is a class which allows a user to configure and control an arbitrary stream of serial data. These streams can represent a number of different transports, including a RS232 external interface, a virtual COM port, and onboard UART interfaces from system components. This entity allows each transport to be configured as either an endpoint, or as a passthrough between transports, similar to a switchboard of a telephone operator.

UART Control

USB-C-Switch Pro has the ability to be controlled via RS-232 on the RX and TX pins of the Expansion Connector. The USB-C-Switch Pro also supports VCOM (Virtual COM port) channels for USB-based serial communication. For additional information about the serial protocol, reference *USB-C-Switch Pro Serial Communication Feature*

UART Entity Indices

The USB-C-Switch Pro provides the following UART entity indices:

Table 18: UART Entity Indices

Index	Description
0	Hardware RS232 (Expansion Connector)
1	VCOM_0 (Virtual COM port channel)

UART Protocols

USB-C-Switch Pro has four protocol values enumerated.

Value	Description
0	Disabled/Undefined
1	Extron Compatible Protocol
2	Brainstem Transport
6	Loopback

UART APIs

UARTs are controlled through the *following APIs*:

```
stem.uart[x].setEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getEnable() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setBaudRate() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getBaudRate() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setProtocol() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getProtocol() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setLinkChannel() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getLinkChannel() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setStopBits() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getStopBits() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setParity() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getParity() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setDataBits() [cpp] [python] [NET] [LabVIEW] [REST]
```

```
stem.uart[x].getDataBits() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setFlowControl() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getFlowControl() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getCapableProtocols() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getAvailableProtocols() [cpp] [python] [NET] [LabVIEW] [REST]
```

USB Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USB Entity provides the software control interface for USB related features. This entity is supported by BrainStem products which have programmatically controlled USB features.

Alt. Mode Configuration

The USB-C-Switch Pro provides an intermediary receiver and amplifier on the HS and SS data lines. Alt-modes such as DisplayPort require different directional uses of the SS data lines. As such, it is required to define the alt-mode and direction of the connection. These modes are responsible for setting the direction of the SS data lines and related SBU lines.

```
stem.usb.getAltModeConfig(0, configuration) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setAltModeConfig(0, configuration) [cpp] [python] [NET] [LabVIEW] [REST]
```

where configuration is an integer value defined below. Details of the pin mapping and data direction are also depicted below.

Index	Alt Mode Configuration
0	SS Redrivers Disabled
1	USB 3.2 Enabled (No DisplayPort)
2	4-Lane DisplayPort Host on Common Port
3	4-Lane DisplayPort Host on Mux Port
4	2-Lane DisplayPort with USB 3.2 – Host on Common Port
5	2-Lane DisplayPort with USB 3.2 – Host on Mux Port
6	2-Lane DisplayPort Host on Common Port with USB 3.2 Inverted
7	2-Lane DisplayPort Host on Mux Port with USB 3.2 Inverted
8	USB4 / TB Enabled
9	USB4 / TB Inverted

Cable Flip

In normal operation, USB-C-Switch Pro connects signals on side A of the Common port to side A of the active Mux port. USB-C-Switch Pro can flip this mapping programmatically, rerouting CC/V_{CONN}, SBU, and USB data so that side A of the Common Port connects to side B of the Mux ports.

```
stem.usb.getCableFlip(setting) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setCableFlip(setting) [cpp] [python] [NET] [LabVIEW] [REST]
```

where the parameter (setting) is an interger value of 0 or 1, where 0 is normal and 1 is full cable flip.

Individual functional groups of the USB connection can be flipped using the portMode option.

Port Operational State

The portState option provide an interface to the state of the common port and internals of the USB-C-Switch Pro system.

```
stem.usb.getPortState(0, state) [cpp] [python] [NET] [LabVIEW] [REST]
```

where (state) is a 32-bit word, defined below.

Bit	Port State Bit Map
0	Vbus enable
1	HS Side A Data enable
2	HS Side B Data enable
3	SBU enable
4	SS Lane 1 Data enable
5	SS Lane 2 Data enable
6	CC1 enable
7	CC2 enable
13:8	Reserved
14	CC Flip enable
15	Super-Speed Flip enable
16	SBU Flip enable
17	High-Speed Flip enable
19:18	Daughter-Card status
23:20	Reserved
24	KAC enable
31:25	Reserved

SBU Manipulation

```
stem.usb.setSBUEnable(0, enabled) [cpp] [python] [NET] [LabVIEW] [REST]
```

```
stem.usb.getSBUEnable(0, enabled) [cpp] [python] [NET] [LabVIEW] [REST]
```

where (enable) is a Boolean vale of 0 or 1.

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
App[0-3]	execute	
	return	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	
	getTransferStore	
	setTransferStore	

continues on next page

Table 19 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	
	setInt	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	getModel	
	getHardwareVersion	
	getModule	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getVersion	
	getSerialNumber	
	save	
	reset	
	getInputVoltage	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	getUptime	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
timer[0-8]	getExpiration	
	setExpiration	
mux[0]	setEnabled	
	getEnabled	
	setChannel	
	getChannel	
	getConfiguration	
	setConfiguration	
	getSplitMode	
	setSplitMode	
	getVoltage	Channels 0-3
equalizer[0-1]	setReceiverConfig	
	getReceiverConfig	

continues on next page

Table 19 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
UART[0-1]	setTransmitterConfig	
	getTransmitterConfig	
	setEnabled	
	getEnabled	
	setBaudRate	
	getBaudRate	
	setProtocol	
	getProtocol	
	setLinkChannel	
	getLinkChannel	
	setStopBits	
	getStopBits	
	setParity	
	getParity	
	setDataBits	
	getDataBits	
	setFlowControl	
	getFlowControl	
	getCapableProtocols	
	getAvailableProtocols	
digital[0]	getConfiguration	
	getState	
	getValue	
	setConfiguration	
ethernet[0]	setValue	
	getIPv4Address	
	getIPv4DNSAddress	
	getIPv4Gateway	
	getIPv4Netmask	
	getInterfacePort	
	getMACAddress	
	getNetworkConfiguration	
	getStaticIPv4Address	
	getStaticIPv4Gateway	
	getStaticIPv4Netmask	
	setHostname	
	setInterfacePort	
	setNetworkConfiguration	
i2c[0]	setStaticIPv4DNSAddress	
	read	Single I2C entity
	write	
	setPullup	
port[0-5]	setSpeed	
	getSpeed	
	getVbusVoltage	
	getVbusCurrent	
	getVconnVoltage	
	getVconnCurrent	
	getPowerEnabled	
	setPowerEnabled	

continues on next page

Table 19 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getPowerMode	
	setPowerMode	
	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getDataHS1Enabled	
	setDataHS1Enabled	
	getDataHS2Enabled	
	setDataHS2Enabled	
	getDataSSEnabled	
	setDataSSEnabled	
	getDataSS1Enabled	
	setDataSS1Enabled	
	getDataSS2Enabled	
	setDataSS2Enabled	
	getVconnEnabled	
	setVconnEnabled	
	getVconn1Enabled	
	setVconn1Enabled	
	getVconn2Enabled	
	setVconn2Enabled	
	getDataRole	
	getDataSpeed	
	getCCEnabled	
	setCCEnabled	
	getCC1Enabled	
	setCC1Enabled	
	getCC2Enabled	
	setCC2Enabled	
	getCCBias	
	setCCBias	
	getMode	
	setMode	
	getState	
	getName	
	setName	
	getCurrentLimit	
	setCurrentLimit	
	getAllocatedPower	
	getAvailablePower	
	getPowerLimit	
	setPowerLimit	
	getVbusAccumulatedPower	
	resetVbusAccumulatedPower	
	getVconnAccumulatedPower	
	resetVconnAccumulatedPower	
	getHSBoost	

continues on next page

Table 19 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
powerdelivery[0-5]	setHSBoost	
	getDataHSRoutingBehavior	
	setDataHSRoutingBehavior	
	getDataSSRoutingBehavior	
	setDataSSRoutingBehavior	
	getCC1AccumulatedPower	
	getCC1State	
	getCC2AccumulatedPower	
	getCC2State	
	getCCCurentLimit	
	setCCCurentLimit	
	getErrors	
	setPowerMode	
	getSBUEnabled	
	setSBUEnabled	
	getConnectionState	
	getNumberOfPowerDataObjects	
	setPowerDataObject	
	getPowerDataObject	
	resetPowerDataObjectToDefault	
	getPowerDataObjectList	
	setPowerDataObjectEnabled	
	getPowerDataObjectEnabled	
	getPowerDataObjectEnabledList	
	setRequestDataObject	
	getRequestDataObject	
	getPowerRole	
	setPowerRole	
	getPowerRolePreferred	
	setPowerRolePreferred	
	getCableVoltageMax	
	getCableCurrentMax	
	getCableSpeedMax	
	getCableType	
	getCableOrientation	
	setOverrides	
	getOverrides	
	request	
	setCurrentLimitBehavior	
	getCurrentLimitBehavior	
	getPeakCurrentConfiguration	
	setPeakCurrentConfiguration	
	getFastRoleSwapCurrent	
	setFastRoleSwapCurrent	
	setEnabled	
	vDM	
rail[0]	setEnabled	
	getEnable	
	clearFaults	
	getOperationalState	

continues on next page

Table 19 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
temperature[0-4]	getValue	
	getValueMax	
	getValueMin	
	resetLoggedValues	
usbsystem[0]	getSelectorMode	
	setSelectorMode	
usb[0]	getCC1Enable	Legacy USB entity (cmdUSB)
	getCC2Enable	
	getPortCurrent	
	getPortVoltage	
	setCC1Enable	
	setCC2Enable	
	setDataDisable	
	setDataEnable	
	setHiSpeedDataDisable	
	setHiSpeedDataEnable	
	setSuperSpeedDataDisable	
	setSuperSpeedDataEnable	
	setPortDisable	
	setPortEnable	
	setPowerDisable	
	setPowerEnable	
	setPortMode	
	getPortMode	
	getPortState	
	getCableFlip	
	setCableFlip	
	getConnectMode	
	setConnectMode	
	getCC1Voltage	
	getCC2Voltage	
	getCC1Current	
	getCC2Current	
	setSBUEnable	
	getSBUEnable	

1.4.7 USB-C-Switch Pro Cabling Guide

- If you are not specifically testing both orientations of a USB-C port, **use standard USB-C cables** rather than UOCs (see below).
- Use a standard USB-C cable to connect the **test runner or Control PC** to the **Control Port**.
- The total length of the **Common–Mux path** (Common cable + Mux cable) should be below the nominal limits for the highest USB speed in use:

USB Speed	Approximate Maximum Total Cable Length (Common + Mux)
USB 2.0	5 m
USB 5Gbps	2–3 m
USB 20Gbps	1 m
USB 40Gbps	0.8–1 m

- For USB4 40Gbps and Thunderbolt links, Acroname recommends using **0.3 m 80 Gbps-rated cables**.

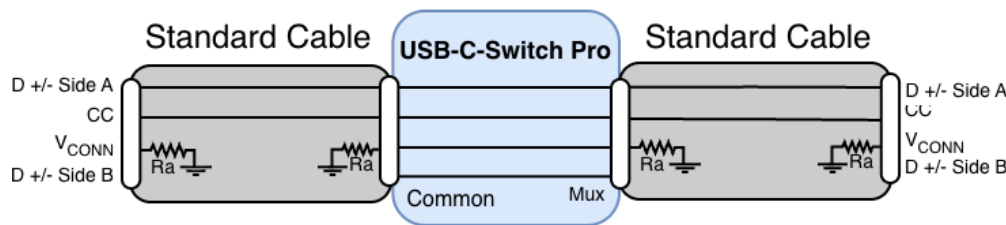
Orientation Matching

In normal operation, a USB-C cable can be plugged in either orientation; hosts and devices then negotiate the orientation and remap pins internally. Standard USB-C cables pass **CC** and **USB2 D+/D-** signals only from **one side** of the connector.

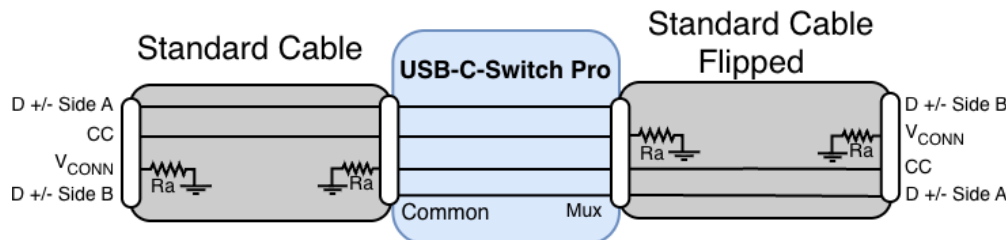
Because the USB-C-Switch Pro joins two cables mid-connection (Common-to-Mux), the **orientations of the two cables must match** for CC and USB2 to pass through.

If cable orientations do not match:

- CC communication fails: **no USB-PD negotiation**
- USB2 D+/D- are blocked: **no USB 2.0 or HS signaling**



Standard cables, orientation matched (all signals pass).



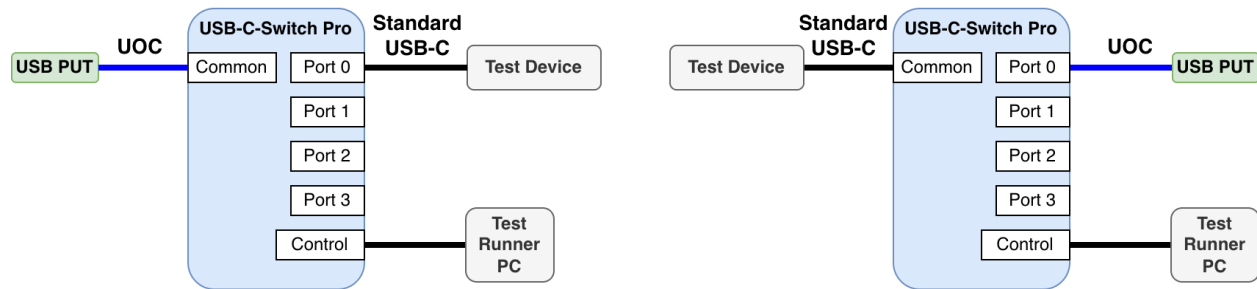
Standard cables, orientation flipped (CC and HS blocked).

In all cases — including with UOCs — these signals pass through in both orientations: V_{BUS} , SS lanes, SBU, and GND. (Omitted from the diagrams for clarity).

Port Testing: UOCs

Acroname Universal Orientation Cables (UOCs) are nonstandard USB-C cables designed to automate testing of both sides of a USB-C receptacle when used with the USB-C-Switch Pro. These cables pass V_{CONN} and CC signals without the standard R_a termination.

- Each **Port Under Test (PUT)** connects through a UOC.
- The **testing device** connects using a standard cable.
- The PUT may be connected to either the **Common port** or a **Mux port**, depending on the test flow.

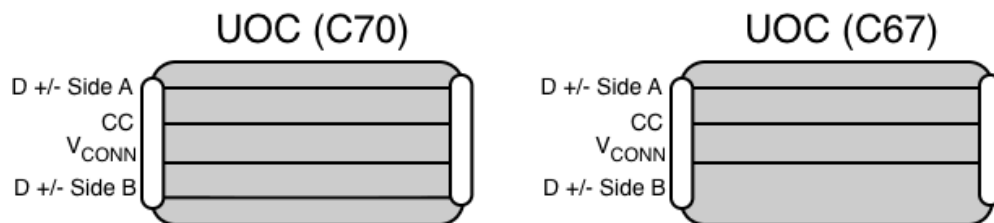


Connect the UOC to the Port Under Test (PUT)

UOC Selection

Device USB-C ports either **short** sides A and B USB 2.0 (HS) data lines or use a **mux** to keep the two sides electrically independent. Choose the UOC that matches your DUT port.

Two UOC cable types are included:

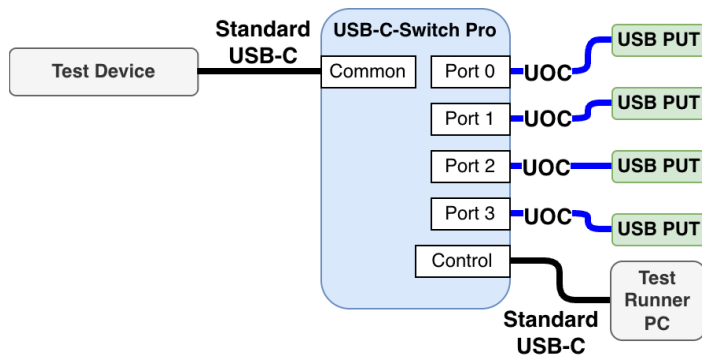


- **C70 — for muxed USB2 ports** - Routes **both CC lines** and **two independent HS pairs** (for devices with independent A/B HS pins).
- **C67 — for shorted USB2 ports** - Routes **both CC lines** and **one shared HS pair** (for devices that short HS A to B).

If C70 is used on a port that shorts HS A/B, the second HS pair in the cable acts as an antenna, leading to poor signal quality. Connecting C67 to a port with Muxed HS sides results in no USB 2.0 connection in one of the cable flip orientations.

UOC Connection Examples

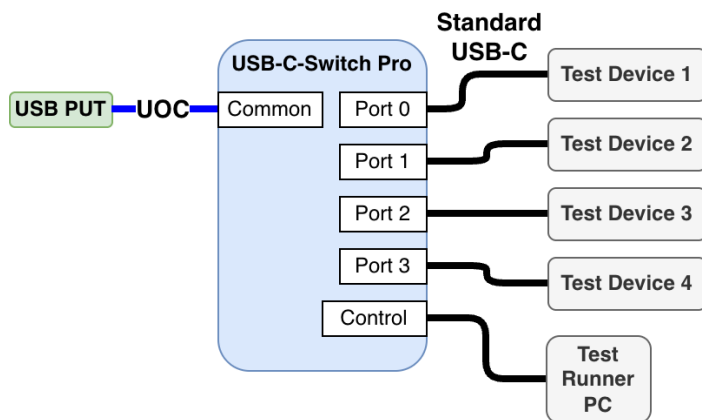
Multiple PUTs, one testing device



To test **multiple DUT ports** (or multiple DUTs) against a **single tester**:

- Connect each PUT to a **Mux port** using the correct UOC.
- Connect the testing device to the **Common port** using a standard cable.
- Connect the test runner to the **Control port**.

One PUT, multiple testing devices



To test **one PUT** against **multiple testers**:

- Connect the PUT via UOC to the **Common port**.
- Connect each tester to the **Mux ports** with standard cables.
- Connect the test runner to the **Control port**.

Any attached DUT or testing device may connect to the USB-C-Switch Pro Control Port and act as the test runner.

1.5 USB-C-Switch

1.5.1 Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)¹¹ for your particular operating system and architecture.
- Download [HubTool](#)¹² for your particular operating system and architecture.

2. Connect Device(s)

- Using standard USB-C cables, connect the Common and Control Ports of the USB-C-Switch Pro to the host.
- Connect devices to any of Mux Ports 0-3 with standard USB-C cables or one of the included Acroname Universal Orientation Cables (UOCs).
 - If using standard cables, you may need to *flip the Switch side of the device cable* to get it to connect.
 - UOCs will work in either orientation. UOC's come in two types: use the C70 cable if your device has independent HS sides and use C67 for when HS sides A and B are shorted. See [Which universal orientation cable should I use?](#)¹³

3. Run System

- Open HubTool
- On the bottom right side of the application select the USB-C-Switch device.

Note: *Linux users will need run the script labeled "udev.sh" located in the "BrainStem_linux_Driverless" folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the USB-C-Switch. For more information please take a look at our [Getting Started Guide](#)

1.5.2 Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    // Connect to the hardware.
    aUSBCSwitch cswitch;
    auto err = cswitch.discoverAndConnect(USB);
```

(continues on next page)

¹¹ <https://acroname.com/api>

¹² <https://acroname.com/hubtool>

¹³ <https://acroname.com/blog/which-universal-orientation-cable-should-i-use>

(continued from previous page)

```

if (err != aErrNone) {
    printf("Error %d encountered connecting to BrainStem module\n", err);
    return 1;

} else { printf("Connected to BrainStem module.\n"); }

//Prep USBSwitch for testing
cswitch.usb.setPortDisable(0);
cswitch.mux.setEnable(false);
cswitch.mux.setChannel(0);

//////////
//Do Stuff: other test initialization
//////////

//Ready for testing
//Enable Port AND Mux
cswitch.usb.setPortEnable(0);
cswitch.mux.setEnable(true);

//////////
//Do Stuff on Mux Channel 0
//////////

cswitch.mux.setChannel(1);

//////////
//Do Stuff on Mux Channel 1
//////////

cswitch.mux.setChannel(2);

//////////
//Do Stuff on Mux Channel 2
//////////

cswitch.mux.setChannel(3);

//////////
//Do Stuff on Mux Channel 3
//////////

//Finished with testing.
//De-initialize.
cswitch.usb.setPortDisable(0);
cswitch.mux.setEnable(false);

//Disconnect
cswitch.disconnect();
}

```

Python

```

import brainstem
# For easy access to error constants
from brainstem.result import Result

```

(continues on next page)

(continued from previous page)

```

from time import sleep
import sys

# Create an instance of a USBSwitch module.
cswitch = brainstem.stem.USBSwitch()

# Locate and connect to the first object you find on USB
result = cswitch.discoverAndConnect(brainstem.link.Spec.USB)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % (result))
    sys.exit(1)
else:
    print ("Connected to BrainStem module.\n")

##Prep USBSwitch for testing
cswitch.usb.setPortDisable(0)
cswitch.mux.setEnable(False)
cswitch.mux.setChannel(0)

#####
## Do Stuff: other test initialization
#####

##Ready for testing
##Enable Port AND Mux
cswitch.usb.setPortEnable(0)
cswitch.mux.setEnable(True)

#####
##Do Stuff on Mux Channel 0
#####

cswitch.mux.setChannel(1)

#####
##Do Stuff on Mux Channel 1
#####

cswitch.mux.setChannel(2)

#####
##Do Stuff on Mux Channel 2
#####

cswitch.mux.setChannel(3)

#####
##Do Stuff on Mux Channel 3
#####

#Finished with testing.
#De-initialize.
cswitch.usb.setPortDisable(0)
cswitch.mux.setEnable(False)

```

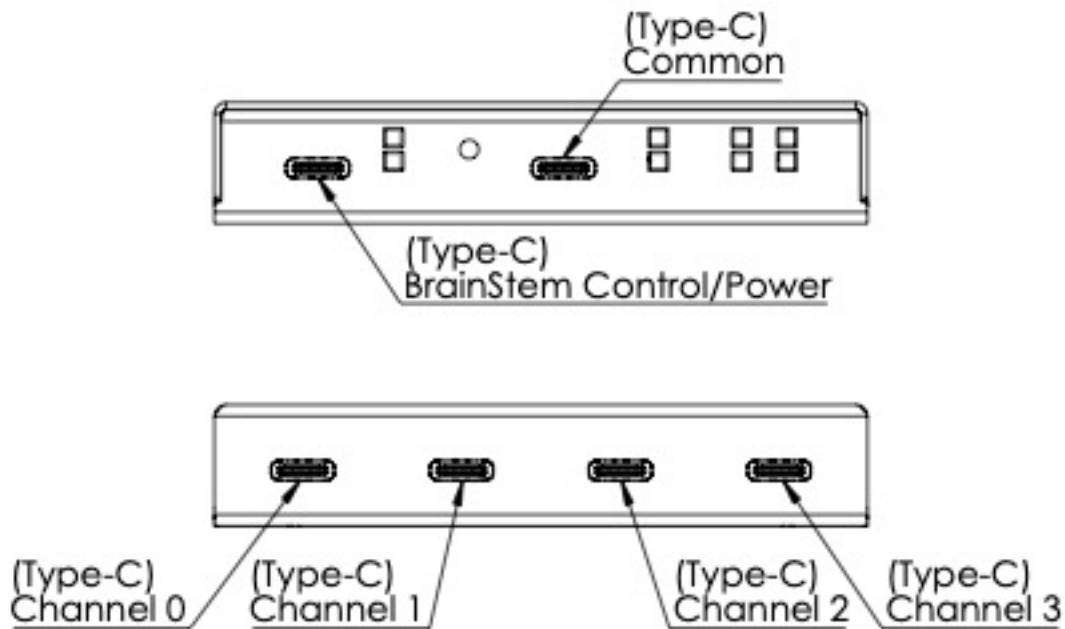
(continues on next page)

(continued from previous page)

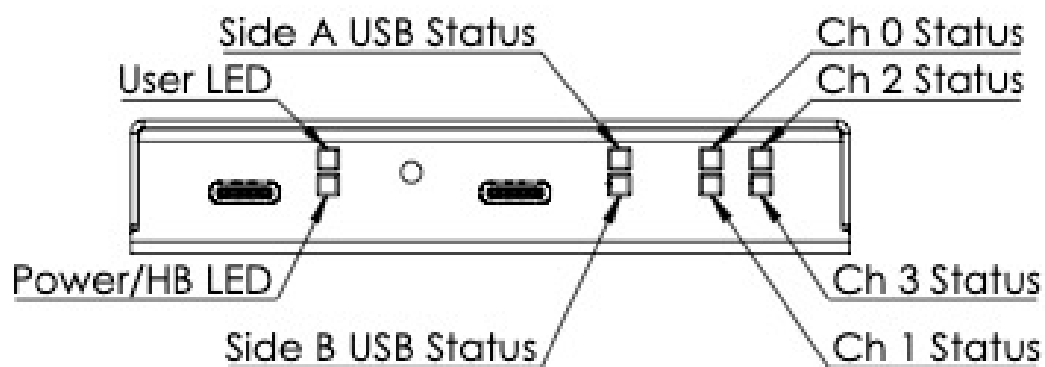
```
# Disconnect from device.
cswitch.disconnect()
```

1.5.3 Indicators and Connections

USB Channels



LED Indicators



LED Name	Color	Description
User	Blue	Can be manipulated through the available APIs
Power/ Heartbeat	Red/Green	Red indicates system is powered. Flashing green is the heartbeat which indicates an active software connection. Pulses at a rate determined by the system heartbeat rate to indicate an active BrainStem link.
Side A USB Status	Green/Yellow/Grey	Un-flipped/Flipped/CC1 Disabled
Side B USB Status	Green/Yellow/Grey	Un-flipped/Flipped/CC2 Disabled
Channel 0 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.
Channel 1 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.
Channel 2 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.
Channel 3 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.

1.5.4 Programming Interface

Generally, the Passive model is best for emulating off-the-shelf cables and for eye-diagram validation.

The Redriver model is optimal for general connectivity or longer connections. It includes a programmable, linear, equalizing redriver which allows USB signal tuning to compensate for insertion and cabling losses.

The USB-C-Switch contains many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the USB-C-Switch.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

Software control of the features of the USB-C-Switch is done with the BrainStem API via a BrainStem link. BrainStem links are done over USB and can be established via the Control Port. After this port is connected to a host machine, a user can connect to it via software API:

```
stem.link.discoverAndConnect(USB)
```

When multiple Acroname devices are connected to a host, connecting to a specific hub can be done by providing the hub serial number. Further, all connected devices can be found using

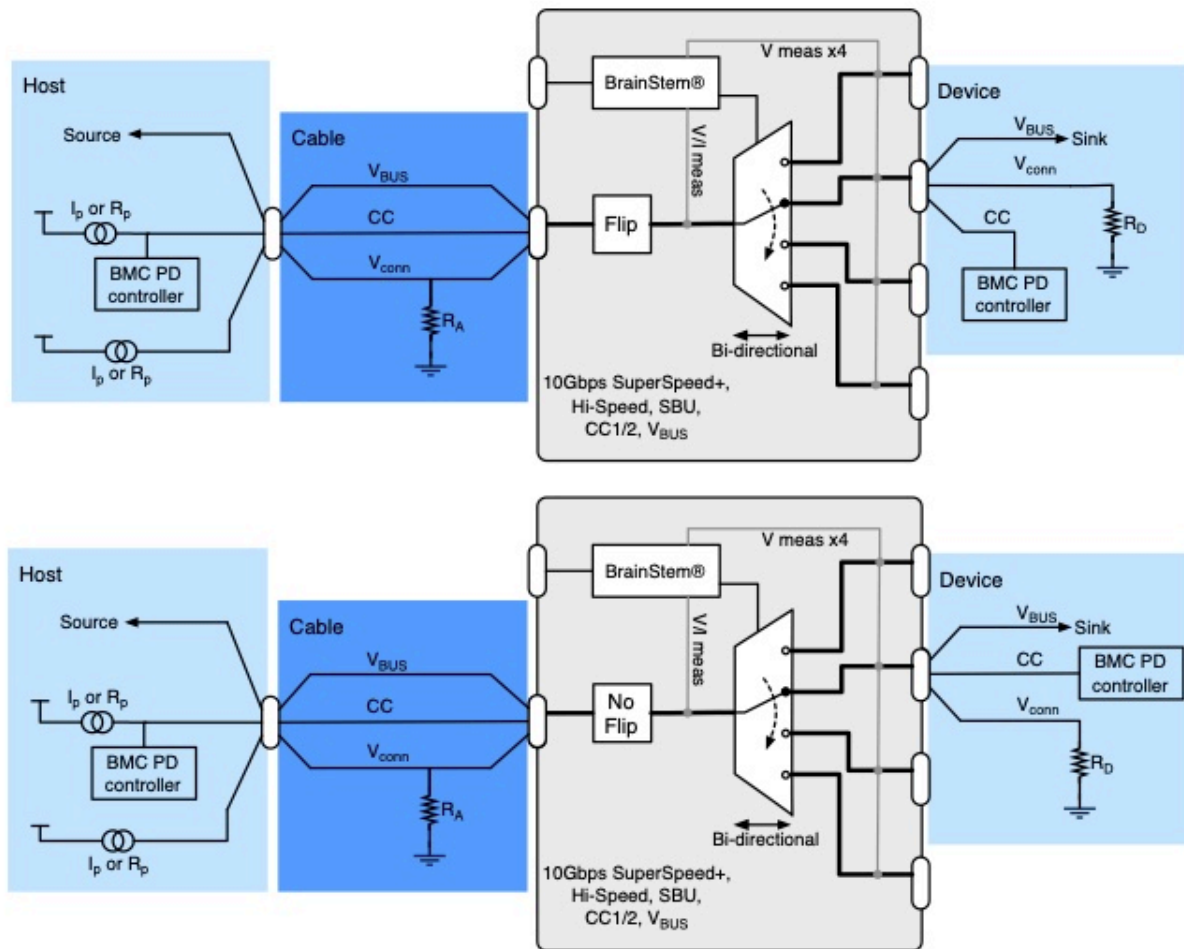
```
brainstem.discover.findAllModules(USB) (Python)
Acroname::BrainStem::Link::sDiscover() (C++)
```

Cable Flip

A key feature of the USB-C connector is its symmetric design allowing for insertion in either orientation. This makes the USB-C connector user-friendly yet complicates the development of devices using the USB-C standard. The orientation is defined by the cable or downstream device in the system; more specifically, by components inside of the USB-C male plug of a connection. The USB-C specification makes determining connector orientation a responsibility of the active devices in the system.

With an Acroname UOC cable, the USB-C-Switch enables the unique ability to affect a cable orientation flip. When this orientation flip occurs, it will appear to connected devices that the orientation of their connection has reversed. Most USB-C devices with a female socket will include at least one set of muxes in order to route signal to the correct side of the socket based on the orientation of the cable. When testing such a system it is import to test both orientations to ensure that these internal muxes are functioning. The USB-C-Switch allows flipping of USB-C cable connections to be programmatically automated.

Depicted below are the flip and no-flip setting for full-featured cable and device



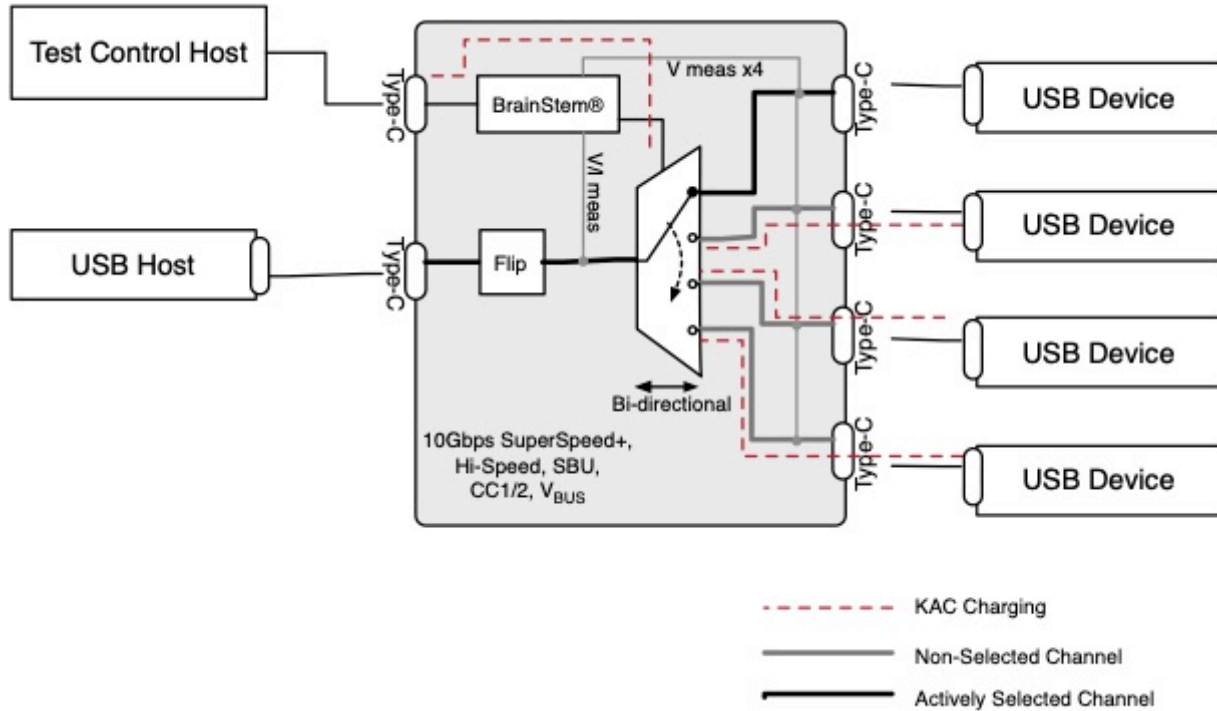
A UOC should be used on either the common port or mux port to enable automated cable flips. The UOC should be connected to the device under test.

When not using the cable flip feature, any standard USB-C cable can be used on both sides of the USB-C-Switch. The orientation of the cables need to be matched in order to facilitate a connection through the switch.

Keep-Alive Charging (KAC)

It is common to use battery powered devices on either side of the USB-C-Switch. When these devices are not in the active path, either on the common or mux side, the device battery may discharge. The USB-C-Switch has the unique feature of Keep-Alive Charging (KAC) for the mux channel connections.

Below is a diagram for the USB-C-Switch KAC capability:



When KAC is enabled, the KAC circuit connects power from the control port VBUS to all non-selected mux channel VBUS lines. KAC power is applied only to inactive mux channels and is not applied to the actively selected mux channel since the actively selected channel has a power path to the common port. KAC is automatically disabled when mux split mode is enabled.

Mux Modes

Default Mode

The default behavior of the USB-C-Switch is to act as a port selector, where all USB-C lines are connected between the common port and one selected mux channel.

Channel Priority Mode

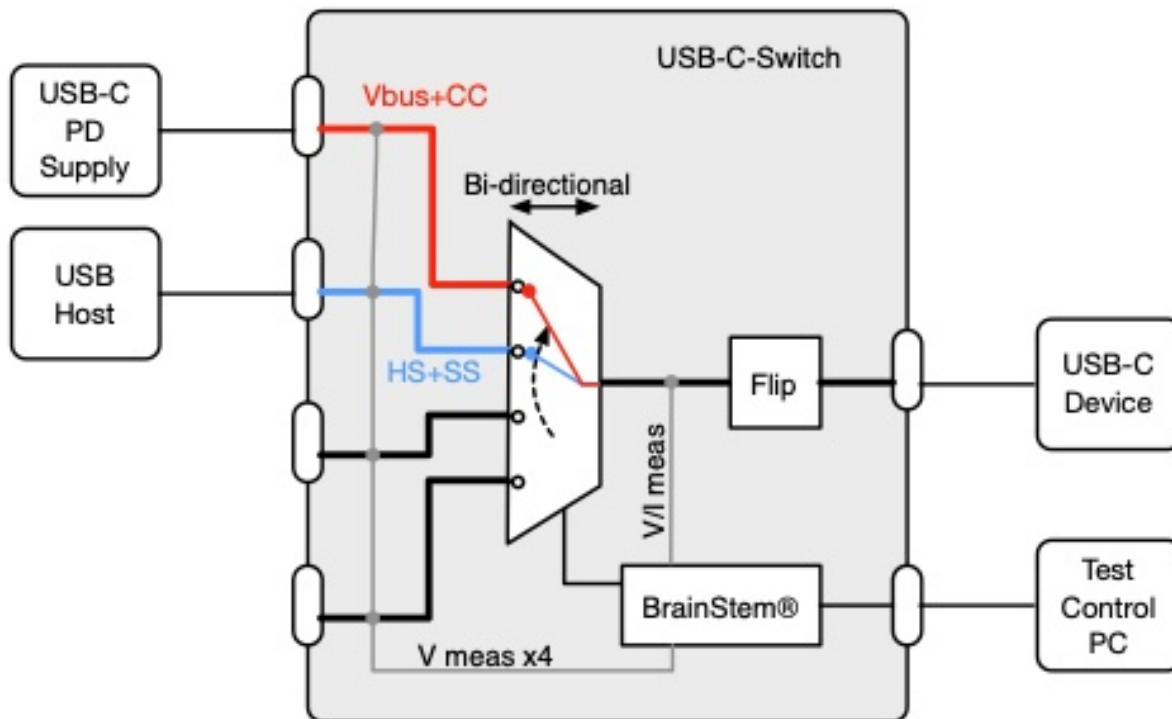
In channel priority mode, the USB-C-Switch automatically selects the lowest-numbered mux port where VBUS is detected, enabling simple automatic host selection. Note that this will only work with USB A-to-C cables, since VBUS is not immediately available on USB-C to USB-C connections.

Split Mode

In some cases, it is desirable to split the connections in a USB-C cable and route them to different mux paths. A common application is to be able connect a USB device to a host machine for USB data while connecting VBUS charging from a device specific charger.

Split mode gives control over individual signal groups, allowing each group to be connect to a mux channel. VBUS can be connected to any combination of mux channels or disabled on the mux channels. Signal groups under Split control assignment are: VBUS, SSA (TX1+/-, RX1+/-), SSB (TX2+/-, RX2+/-), HSA (D+/-, Side A), HSB (D+/-, Side B), CC1, CC2, SBU1, and SBU2.

A basic example of the USB-C-Switch Mux split mode is depicted.



When split mode is enabled, USB-C-Switch will automatically disable the Keep-Alive-Charging (KAC) feature.

Device Drivers

The USB-C-Switch leverages operating system user space interfaces that do not require custom drivers for operation on all modern operating systems including Windows, Linux and MacOS X. With a connection between a host PC and the USB-C control port, the host PC will recognize a USB full-speed device named “USBCSwitch”.

Legacy operating systems like Windows 7 may require the installation of a BrainStem USB driver. Installation details on installing USB drivers can be found within the BrainStem Development Kit under the “drivers” folder.

1.5.5 USB-C-Switch Module Entities

Equalizer

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Equalizer entity provides a concise interface for controlling equalizer and filter settings for receivers (inputs) and transmitters (outputs). Products supporting Equalizer are capable of applying frequency dependent gain to their signals. This can allow for compensation for signal loss and degradation due to cable quality, cable length and the number of connections. It can also act as a filter implemented in hardware or firmware. Products may implement one or more equalizers; each can be configured using the Equalizer index. Allowed index values are specified in the product data sheet.

Note: The Equalizer Entity is only functional on the Redriver Version of the USB-C-Switch.

Equalizer Mapping and Entities

The redriver model of the switch provides two equalizer entities. They provide programmatic control over linear equalizers and amplifiers (aka: redrivers) connected to the HS and SS sata lines. These equalizer entities split the configuration between receiver-side and transmitter-side settings allowing for compensation of signal integrity loss due to cable quality, length, and insertion losses. However, some of the settings can have combined effects between receiver and transmitter modes. The two equalizer entities are indexed to their respective data lines as defined below:

Index	Equalizer Entity Mapping
0	USB2 High Speed
1	USB3 SuperSpeed

The transmitter is responsible for driving and selectively amplifying the signals traveling out the redriver hardware after any receiver-side equalization. Each equalizer entity has transmitter options of:

```
stem.equalizer[x].setTransmitterConfig(config) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.equalizer[x].getTransmitterConfig(config) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The receiver attempts to compensate for distortion of the incoming signal. Each equalizer entity has receiver options such as:

```
stem.equalizer[x].setReceiverConfig(chan, config) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.equalizer[x].getReceiverConfig(chan, config) [cpp] [python] [NET] [CCA] [REST]
```

where (chan) paramters are defined below:

Value	Receiver Channel
0	Applies setting to both common and mux sides
1	Applies settings to mux side
2	Applies settings to common side

High Speed Redriver Configuration

Due to the half-duplex nature of the USB2 data lines, there is only one receiver and transmitter setting for both the common and mux ports. In addition, since the transmitter and receiver are tightly coupled, the linear gain achieved by transmitter setting varies with the equalizer receiver configuration. Approximate gains for example configurations are shown in the specifications table.

```
stem.equalizer[0].setTransmitterConfig(config) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.equalizer[0].getTransmitterConfig(config) [cpp] [python] [NET] [CCA] [REST]
```

The HS Equalizer entity transmitter option controls the gain applied to HS signals only; USB Low Speed (LS) and Full Speed (FS) signals are unaffected and uncompensated. This option changes the DC boost applied to HS signals which can help achieve sharper rising edges. The allowed values are shown below:

Value	High Speed Transmitter Configuration
0	40mV DC Boost
1	60mV DC Boost
2	80mV DC Boost
3	0mV DC Boost (disabled)

The chan parameter of the HS equalier receiver option can only be 0 because the HS data lines are half-duplex. All other values will result in an error return.

The HS equalizer receiver option configurations control the sensitivity of the redriver to incoming HS signals. The effect of this change in sensitivity can be considered a variable AC boost turned to the specific HS signal applied. Setting the HS equalizer to Level 0 will disable the HS redriver regardless of the HS entity transmitter configuration. The available options are shown below:

```
stem.equalizer[0].setReceiverConfig(0,config) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.equalizer[0].getReceiverConfig(0,config) [cpp] [python] [NET] [CCA] [REST]
```

Value	High Speed Receiver Equalization
0	Level 1
1	Level 2
2	Level 0 (disabled)

Super Speed Redriver Configuration

The SS equalizer option controls various transmitter gains for each side of the full-duplex SS data lines. Each configuration combines the transmitter gain and approximate peak-to-peak voltage for both the common and mux side transmitters. The available options are shown below.

```
stem.equalizer[1].setTransmitterConfig(config) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.equalizer[1].getTransmitterConfig(config) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Value	Mux Side	Com Side	Range
0	1db	0db	900mVpp
1	0db	1db	900mVpp
2	1db	1db	900mVpp
3	0db	0db	900mVpp
4	0db	0db	1100mVpp
5	1db	0db	1100mVpp
6	0db	1db	1100mVpp
7	2db	2db	1100mVpp
8	0db	0db	1300mVpp

The SS equalizers receiver option controls the receiver gain. The actual receiver gain is dependent on the alt-mode configuration and the port data direction (mux to common vs common to mux). There are independent receiver gain settings for the common and mux ports of the switch. Gains across settings, direction, and frequency is shown here:

```
stem.equalizer[1].setReceiverConfig(0,config) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.equalizer[1].getReceiverConfig(0,config) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Value	SS Recieve Gain Lever
0-15	Increasing levels of gain

Mux

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs.

Some mux entities can simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

Mux Channel

The mux entity primarily selects one active mux port to connect to the common port using the channel option:

```
stem.mux.setChannel(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.mux.getChannel(channel) [cpp] [python] [NET] [CCA] [REST]
```

where (channel) is an index 0-3.

Mux Configuration

Default configuration of the mux is to switch all enabled USB-C lines to a single mux channel. If desired, the switch can split the USB-C functional group and route them to selected mux ports. This feature is referred to as “split mode”. The switch can also auto select the lowest Mux channel that currently has VBUS present which is called “port priority”. Default, split mode, or port priority can be enabled with:

```
stem.mux.getConfiguration(config) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.mux.setConfiguration(config) [cpp] [python] [NET] [CCA] [REST]
```

where (config) is 0 for default, 1 for Split Mode, and 2 for Port Priority Mode.

Split Mode

After enabling split mode the USB-C functional groups can be individually assigned to separate mux channels with:

```
stem.mux.getSplitMode(splitMode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.mux.setSplitMode(splitMode) [cpp] [python] [NET] [CCA] [REST]
```

where (splitMode) is a 32-bit word, defined below. Each bit pair is a 2-bit binary number from 0-3 representing the mux port to which to route the functional signal group. Vbus and CC use 4-bits to define which mux ports are connected to the common port Vbus/CC lines.

Bit	Mux Split Mode Bit Map
0:1	SBU1
2:3	SBU2
4	CC1 enable CH0
5	CC1 enable CH1
6	CC1 enable CH2
7	CC1 enable CH3
8	CC2 enable CH0
9	CC2 enable CH1
10	CC2 enable CH2
11	CC2 enable CH3
12:13	HS Side A Data
14:15	HS Side B Data
16:17	SS Lane 1 Data
18:19	SS Lane 2 Data
20	Vbus enable CH0
21	Vbus enable CH1
22	Vbus enable CH2
23	Vbus enable CH3
24:31	Reserved

Note: The split mode interface changed in 2.10.0, ensure your firmware and software are up to date.

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every USB-C-Switch is assigned a unique serial number at the factory. This facilitates an arbitrary number of USB-C-Switch devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Hardware Version

Get the module's hardware version information.

```
stem.system.getHardwareVersion(hardwareVersion) [cpp] [python] [NET] [CCA] [REST]
```

The hardware version is a 32-bit value that encodes hardware revision and variant information. The format is:

Table 20: Hardware Version Bit Fields

Bits	Field	Description
24-31	Revision	Hardware revision identifier
16-23	Reserved	Reserved (always 0)
8-15	Variant	Hardware variant identifier
0-7	Reserved	Reserved (always 0)

Table 21: Hardware Revision Values

Value	Description
'B'	Revision B
'C'	Revision C
'D'	Revision D

Table 22: Hardware Variant Values

Value	Description
'P'	Passive
'R'	Basic Redriver
'S'	Redriver with SBU measurements
'I'	Redriver and Isolator with SBU Measurements

The hardware revision is determined by reading hardware strapping pins during initialization. The daughtercard variant is detected by reading the daughtercard type and available features (SBU ADC, USB2 isolator) to identify the specific daughtercard configuration.

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the USB-C-Switch away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Saved Configurations	
USB Mode (usb)	Equalizer Configuration (equalizer)
Mux Split Mode (mux)	Mux Enable (mux)
Mux Configuration (mux)	Mux Port (mux)

USB

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USB Entity provides the software control interface for USB related features. This entity is supported by BrainStem products which have programmatically controlled USB features.

Alt. Mode Configuration (Redriver Only)

The redriver model USB-C-Switch provides an intermediary receiver and amplifier on the HS and SS data lines. Various alt-modes such as DisplayPort require different directional uses of the SS data lines. As such, it is required to define the alt-mode and direction of the connection. These modes are responsible for setting the direction of the SS data lines and related SBU lines.

```
stem.usb.getAltModeConfig(0, configuration) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setAltModeConfig(0, configuration) [cpp] [python] [NET] [CCA] [REST]
```

where configuration is an integer value defined below. Details of the pin mapping and data direction are also depicted below.

Index	Alt Mode Configuration
0	USB 3.1 Disabled
1	USB 3.1 Enabled
2	4 Lane DisplayPort Host on Common Port
3	4 Lane DisplayPort Host on Mux Port
4	2 Lane DisplayPort with USB 3.1 – Host on Common Port
5	2 Lane DisplayPort with USB 3.1 – Host on Mux Port
6	2 Lane DisplayPort Host on Common Port with USB 3.1 Inverted
7	2 Lane DisplayPort Host on Mux Port with USB 3.1 Inverted

Common Port Pin								Mux Port Pin Normal	Mux Port Pin Flipped
Redriver Config	USB 3.1	4 Lane DisplayPort Host on Common	4 Lane DisplayPort Host on Mux	2 Lane DisplayPort Host on Mux with USB3.1	2 Lane DisplayPort Host on Common with USB3.1	2 Lane DisplayPort Host on Common with USB 3.1 Inverted	2 Lane DisplayPort Host on Mux with USB 3.1 Inverted	Color Key	
								USB HS	
								USB SS	
								DisplayPort (alt-mode)	
A2	←	→	←	←	→	→	←	B11	A11
A3	←	→	←	←	→	→	←	B10	A10
A10	→	→	←	←	→	→	←	B3	A3
A11	→	→	←	←	→	→	←	B2	A2
B2	←	→	←	←	←	→	→	A11	B11
B3	←	→	←	←	←	→	→	A10	B10
B10	→	→	←	→	→	←	←	A3	B3
B11	→	→	←	→	→	←	←	A2	B2
A8	↔	↔	↔	↔	↔	↔	↔	B8	A8
B8	↔	↔	↔	↔	↔	↔	↔	A8	B8

Cable Flip

The USB-C-Switch can simulate a cable flip by electrically switching the CC/VCONN and SBU lines between side-A and side-B of the USB-C female sockets. USB data lines are also swapped accordingly. This flip can be done with:

```
stem.usb.getCableFlip(setting) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setCableFlip(setting) [cpp] [python] [NET] [CCA] [REST]
```

where the parameter (setting) is an interger value of 0 or 1, where 0 is normal and 1 is full cable flip.

Individual functional groups of the USB connection can be flipped using the portMode option.

CC Manipulation

The automatic orientation detection and connection functionality is interfaced with:

```
stem.usb.setConnectMode(0, mode) [cpp] [python] [NET] [CCA] [REST]
```

where (mode) is a Boolean value of 0 or 1.

Manipulating the CC lines is done by calling:

```
stem.usb.setCC1Enable(0, enabled) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setCC2Enable(0, enabled) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getCC1Enable(0, enabled) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getCC2Enable(0, enable) [cpp] [python] [NET] [CCA] [REST]
```

where (enable) is a Boolean value of 0 or 1.

CC line current and voltage can be measured with:

```
stem.usb.getCC1Voltage(0,  $\mu$ V) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getCC2Voltage(0,  $\mu$ V) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getCC1Current(0,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getCC2Current(0,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

where positive current is power transfer from the common port to the mux port.

Channel Control

The usb entity provides a mechanism to control and monitor all USB functionality on the common port. Individual parts of the USB connection can be manipulated through the usb entity. For example, enable/disable USB data and Vbus lines, measure current and voltage on Vbus, VCONN, and CC. The USB-C-Switch has one usb entity class. It uses the mux entity to select one of the 4 mux channels to which to connect the enabled USB signals.

The usb entity splits the USB connection into tree going from most generic to most specific with usb entity options at each level. Higher levels of the tree can be used to cause simultaneous changes on the lower levels. The tree structure is port(Vbus, data(HS, SS), USB-C(CC1, CC2, SBU)).

The usb.setPortEnable/Disable entity option allows for manipulating all parts of the USB connection (HS data, SS data, both CC and SBU lines, and Vbus lines) simultaneously.

```
stem.usb.setPortEnable(channel) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortDisable(channel) [cpp] [python] [NET] [CCA] [REST]
```

Where channel is always 0 for the USB-C-Switch. Further examples of the usb entity will always show the channel option as 0.

Manipulating USB data lines (HS and SS) simultaneously is done by calling:

```
stem.usb.setDataEnable(0) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setDataDisable(0) [cpp] [python] [NET] [CCA] [REST]
```

Manipulating HS or SS data lines is done by calling:

```
stem.usb.setHiSpeedDataEnable(0 [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setHiSpeedDataDisable(0) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setSuperSpeedDataEnable(0) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setSuperSpeedDataDisable(0) [cpp] [python] [NET] [CCA] [REST]
```

Manipulating Vbus lines are done by calling:

```
stem.usb.setPowerEnable(0) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setPowerDisable(0) [cpp] [python] [NET] [CCA] [REST]
```

Port Manipulation

Vbus voltage and current through the switch's Vbus lines can be measured with:

```
stem.usb.getPortVoltage(0,  $\mu$ V) [cpp] [python] [NET] [CCA] [REST]
stem.usb.getPortCurrent(0,  $\mu$ A) [cpp] [python] [NET] [CCA] [REST]
```

where positive current is power transfer from the common port to the mux port.

Port Mode

The portMode option provides a bitmapped setting for granular control of the individual connections. The portMode option is the desired mode of the port. The companion option, portState, is used to provide the current state of the port.

```
stem.usb.getPortMode(0, mode) [cpp] [python] [NET] [CCA] [REST]
stem.usb.setPortMode(0, mode) [cpp] [python] [NET] [CCA] [REST]
```

where (mode) is a 32-bit word, defined below.

Bit	Port Mode Bit Map
0	Reserved
1	Reserved
2	Keep Alive Charging Enable
3	Reserved
4	HS Side A Data enable
5	HS Side B Data enable
6	Vbus enable
7	SS Lane 1 Data enable
8	SS Lane 2 Data enable
9:10	Reserved
11	Auto Connect enable
12	CC1 enable
13	CC2 enable
14	SBU enable
15	CC Flip enable
16	Super-Speed Flip enable
17	SBU Flip enable
18	Hi-Speed Flip enable
19	CC1 Current Injection enable
20	CC2 Current Injection enable
21:31	Reserved

Port Operational State

The portState option provide an interface to the state of the common port and internals of the USB-C-Switch system.

```
stem.usb.getPortState(0, state) [cpp] [python] [NET] [CCA] [REST]
```

where (state) is a 32-bit word, defined below.

Bit	Port State Bit Map
0	Vbus enable
1	HS Side A Data enable
2	HS Side B Data enable
3	SBU enable
4	SS Lane 1 Data enable
5	SS Lane 2 Data enable
6	CC1 enable
7	CC2 enable
13:8	Reserved
14	CC Flip enable
15	Super-Speed Flip enable
16	SBU Flip enable
17	High-Speed Flip enable
19:18	Daughter-Card status
25:20	Reserved
26	CC1 Current Injection
27	CC2 Current Injection
28	CC1 Pulse detect
29	CC2 Pulse detect
30	CC1 Logic state
31	CC2 Logic state

SBU Manipulation

```
stem.usb.setSBUEnable(0, enabled) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getSBUEnable(0, enabled) [cpp] [python] [NET] [CCA] [REST]
```

where (enable) is a Boolean vale of 0 or 1.

Complete List of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
App[0-3]	execute	
	return	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	

continues on next page

Table 23 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getTransferStore	
	setTransferStore	
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	
	setInt	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	getModel	
	getHardwareVersion	
	getModule	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getVersion	
	getSerialNumber	
	save	
	reset	
	getInputVoltage	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	getUptime	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
timer[0-8]	getExpiration	
	setExpiration	
usb[0]	setPortEnable	
	setPortDisable	
	setDataEnable	
	setDataDisable	
	setHiSpeedDataEnable	
	setHiSpeedDataDisable	
	setSuperSpeedDataEnable	
	setSuperSpeedDataDisable	
	setPowerEnable	

continues on next page

Table 23 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	setPowerDisable	
	getPortVoltage	
	getPortCurrent	
	getPortMode	
	setPortMode	
	getPortState	
	setCableFlip	
	getCableFlip	
	setConnectMode	
	getConnectMode	
	setCC1Enable	
	getCC1Enable	
	setCC2Enable	
	getCC2Enable	
	getCC1Voltage	
	getCC2Voltage	
	getCC1Current	
	getCC2Current	
	setSBUEnable	
	getSBUEnable	
mux[0]	setEnabled	
	getEnable	
	setChannel	
	getChannel	
	getConfiguration	
	setConfiguration	
	getSplitMode	
	setSplitMode	
	getVoltage	Channels 0-3
	setReceiverConfig	
equalizer[0-1]	getReceiverConfig	
	setTransmitterConfig	
	getTransmitterConfig	

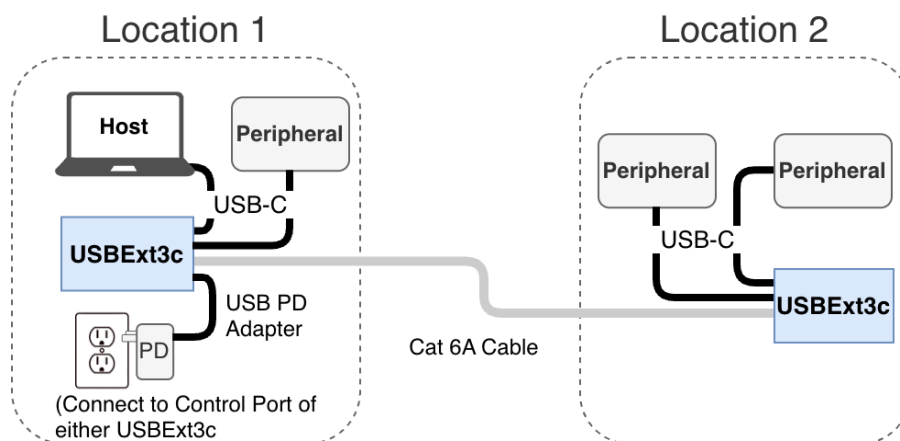
1.6 USBExt3c

1.6.1 Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)¹⁴ for your particular operating system and architecture.
- Download [HubTool](#)¹⁵ for your particular operating system and architecture.

2. Connect Devices



Extension Setup

- Position devices at the desired locations. Devices are identical and can be placed on either end of the extension.
- Connect their HDBaseT-USB3 ports with a Cat6A cable (not included in the kit).

Power Connection¹⁶

- Connect the included 100 W USB-PD power adapter to any available¹⁷ USB port on either USBExt3c. The other extender and attached devices will be powered over the extension link.

Host and Peripheral Connections

- Connect the host device to *Port 0* or *Port 1* via included USB-C cable.
 - The USBExt3c will automatically detect when a host-capable device has connected and set the corresponding port to be Upstream-Facing.
- Connect peripherals to other available ports 0 or 1 on either end of the extension link via USB-C .
- Confirm that peripherals are visible to the host computer.

¹⁴ <https://acroname.com/api>

¹⁵ <https://acroname.com/hubtool>

¹⁶ Several alternate power configurations are supported, including bus power and powering both devices. see [Flexible Power Routing](#).

¹⁷ If controlling the extender pair via a control port, keep the desired control port available by powering the extender pair through other ports.

3. Connect with HubTool

- Open HubTool on the host computer.
- On the bottom right side of the application, select the USBExt3c device to connect.
- Once the connection is established, the second USBExt3c device will also appear on the bottom right .

Note: *Linux users will need to run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the USBExt3c. For more information please take a look at our [Getting Started Guide](#)

1.6.2 Functionality

Feature Descriptions

Each USBExt3c is addressable and controllable from a host system via USB-C Control Port, Ethernet (TCP/IP), RS-232, or via a second USBExt3c over the HDBaseT-USB3 Link. Once connected, a BrainStem link is established to the onboard controller, enabling software control through the BrainStem API. This API provides full access to all module functionality from a host system.

USB Ports

The two full-featured ports (0 and 1) can be configured as upstream-facing host ports or downstream-facing device ports. The ports implement separate, independently switched USB HS and SS data, CC, V_{CONN} and current-limited V_{BUS} lines, supporting advanced USB testing applications.

The dedicated USB-C control port is a high-speed USB 2.0 connection for BrainStem interface and device power only. No other USB traffic can flow on this connection. Port-level features are controlled by the [Port Entity](#)

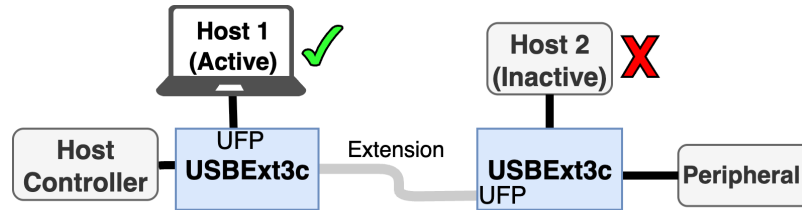
Port	Index
0	0
1	1
Control	2

Automatic Host Port Switching Behavior

By default, any time a host-capable device is connected to any port on either end of the extension, it will become the host. This enables simple host switching for BYOD/M conference rooms: a laptop can temporarily take control of room AV equipment to host a meeting. When the laptop disconnects, the dedicated room PC becomes the host and regains control of the peripherals.

Programmatic Host Switching

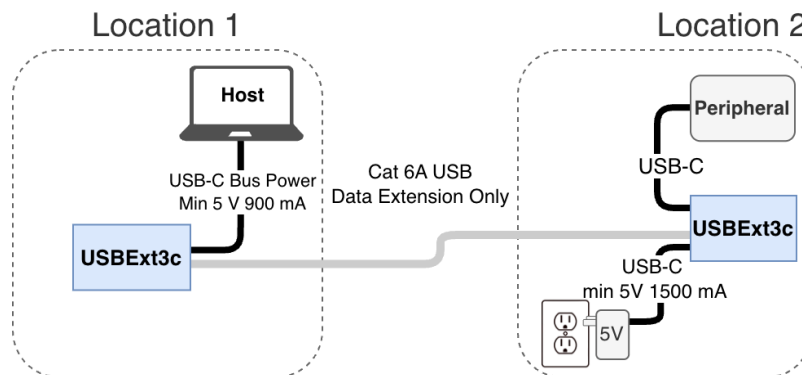
For static installations or when a controller is managing host selection, port roles can also be assigned directly. For each USBExt3, the upstream-facing port role can be assigned to either port or to the HDBaseT-USB3 extension link.



Programmatic host selection by assigning UFP port roles

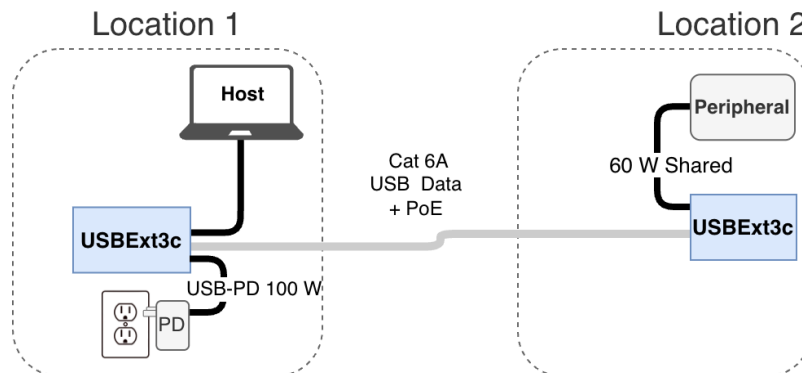
Flexible Power Routing

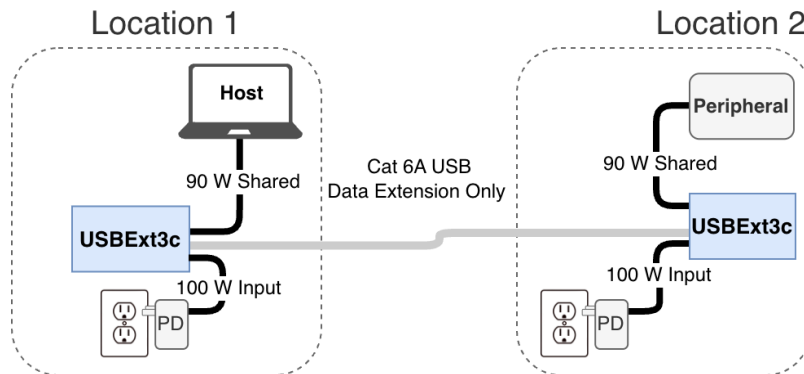
The USBExt3c accepts power through any USB-C port and will operate with as little as 5 V (900 mA) for data-only extension, allowing bus-powered applications. For example, the host-side USBExt3c could be bus-powered by the host, while the device-side extender receives 5 V 900 mA from a power adapter connected to the control port.



Minimal local power configuration

To power a pair of USBExt3c from a single power source (with PoE power provided to the second USBExt3c), the power supply must provide between 45 W and 100 W. When a single 100 W supply is connected to one USBExt3c, up to 60 W is available to devices on the second USBExt3c. For maximum device power, connect a 100 W USB-PD supply to each USBExt3c to provide 90 W shared per extender.



Power extension

Maximum device power is available when both USBExt3c extenders are powered

RS-232 Extension

By default, the RS-232 ports are set to extend signals in Oversampled Mode where the RX and TX lines are sampled at 1.5 MHz and reconstructed on the other end. This allows host and clients to negotiate data rates independent of the extension. It is recommended to limit data rates to 150 k baud (1/10 of Fs).

Ethernet Control

The USBExt3c can be managed over Ethernet using the HubTool application, BrainStem API, REST interface, or built-in web interface. Connections are made through the Ethernet jack using TCP/IP sockets and are supported on the local link segment only. For most setups, we recommend a direct Ethernet link between the host test machine and the USBExt3c.

When using a pair of USBExt3c connected by HDBaseT-USB3, only one Ethernet connection is required; the second device is available through Brainstem networking over the extension link.

By default, the USBExt3c acts as a DHCP client and will receive an IP address from a DHCP server. If no server is detected, the USBExt3c falls back to a static IP address of **192.168.44.42**. In static mode, the host computer interface IP must be set to an address in the **192.168.44.x** range. The DHCP client is limited to hosts on the local link and does not operate across network bridges or gateways.

The USBExt3c responds to ICMP “ping” requests including broadcast pings. The brainstem API interface performs a discovery process prior to establishing communication by sending a UDP multicast request on port 9888. The USBExt3c responds with a message to UDP port 9889. The USBExt3c listens for socket connections on TCP port 8000.

Host firewall rules must allow:

- Outgoing UDP multicast on port 9888
- Incoming UDP responses on port 9889
- Outgoing TCP connections to port 8000
- Incoming / Outgoing TCP connections on ports 9005 and 9006

HDBaseT-USB3 Configuration

The HDBaseT-USB3 port provides simultaneous extension of USB SuperSpeed, High-speed, RS232, and GPIO / frame sync data over Cat6A cable, along with bidirectional PoE++ Power. The HDBaseT-USB3 link direction normally operates in Automatic mode but can be manually configured for optimizing fixed installations.

Detailed HDBaseT-USB3 link parameters from either extender in the pair can be viewed in HubTool or accessed through the BrainStem API, including:

- Index (local or remote)
- State (Device present, Link role and status)
- Serial Number
- VS6320 Firmware Version
- Cable Length (m)
- Mean Squared Error (μ B)
- Retransmission rate (# messages between retransmission, 0 = no errors)
- Link Utilization (μ %)
- Encoding state (e.g. PAM 8)

PoE++ Configuration

The PoE++ system will automatically manage power extension based on power sources and loads on either end of the extension. However, for testing, diagnostics, and advanced applications, low-level control and monitoring is provided, including:

Control:

- Set Power Mode (PSE / PD / Auto / Off)
- Set Sourcing Class Signature for pairs 1-2 and 3-4

Monitoring:

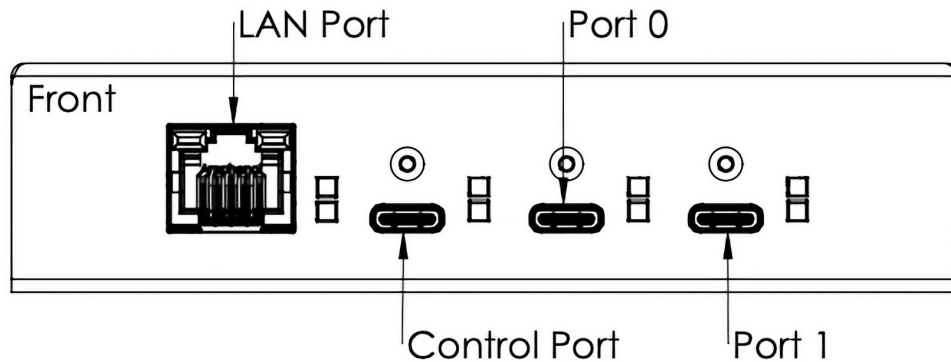
- Power State (PD / PSE / Off)
- Requested Class (Class requested from PD to PSE)

For pairs 1-2 or 3-4:

- Sourcing Class (PoE class PD is trying to source)
- Discovered Class (negotiated class)
- Detection Status (valid connection, short, open, Hi Z, Lo Z)
- Voltage (μ V)
- Current (μ A)
- Resistance ($m\Omega$)
- Capacitance (μ F)
- Power (mW)
- Accumulated Power (mWh)

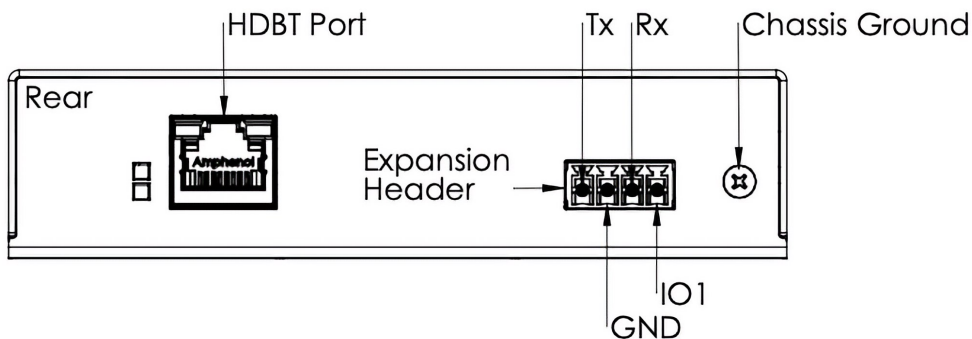
1.6.3 Indicators and Connections

Front Panel



The front of the USBExt3c contains three USB-C receptacles (Control, Port 0, and Port 1) and one Ethernet port. The Control and Ethernet ports are dedicated for control communication between a computer and the USBExt3c. Port 0 and Port 1 connectors are full-featured hub ports.

Rear Panel



The back of the USBExt3c contains an HDBaseT-USB3 port and a 4-pin expansion connector for RS-232, frame sync, and GPIO. Chassis ground is accessed through a grounding screw.

Expansion Connector

The USBExt3c expansion connector is a 4-pin, 1-row, shrouded male header block with 3.81mm pitch. This interface provides additional mechanisms for expandability and testing scenarios.

Table 24: Expansion connector pinout

Connection Name	Pin Number	Description
TX	1	RS-232 Serial Transmit (data from USBExt3c)
GND	2	Ground
RX	3	RS-232 Serial Receive (data to USBExt3c)
IO1	4	General Purpose Input / Selector

LED Indicators

Both the front and back panels of the USBExt3c have status LEDs to communicate device status and behavior. The names and meanings of these LEDs are explained in the image and table below.

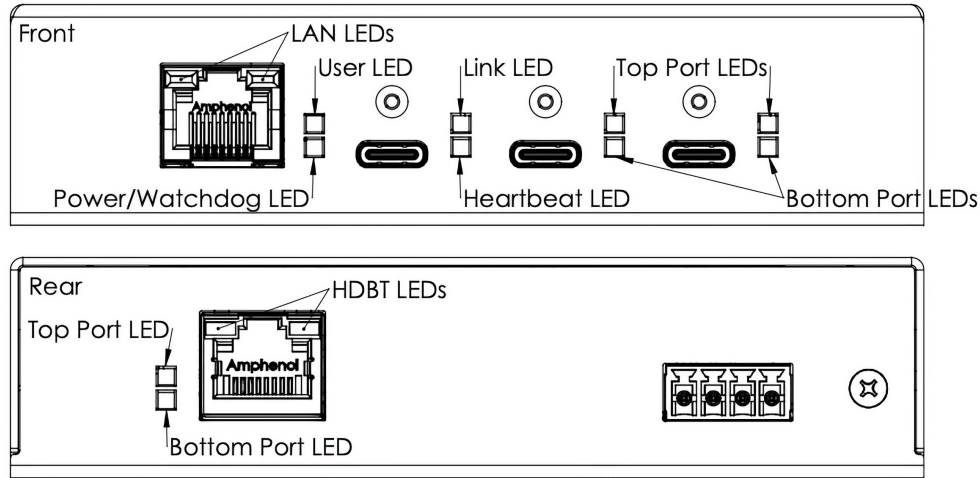


Table 25: LED Indicators and Function

Type	Icon	LED Name	Color	Description
System	Person	User LED	Blue	User-controllable LED
	Power Icon	Power / Watchdog	Red / Magenta	Alternating Red and Magenta when powered
Control	Chain Links	BrainStem Link	Yellow	Link present
	Heart	BrainStem Heartbeat	Green	Blink when Heartbeat received
HDBaseT	None	HDBaseT-USB3 LEDs	Green Amber	Solid Green: USB 3 link exists Solid Amber: HDBaseT-USB3 link exists Both flash during VS6320 firmware update
Port Status: USB-C and HD-BaseT-USB3	None	USB Enumeration Speed (Top LED)	Yellow	USB 2.0 Hi-Speed up to 480 Mbps
			Green	SuperSpeed 5 Gbps
		UFP/DFP (Bottom LED)	Blue	SuperSpeed+ 10 Gbps
			Magenta	Upstream (UFP) port
Ethernet	None	Ethernet Activity	Red	Downstream (DFP) port
			Green Amber	Green LED is solid when a link exists Amber LED blinks during activity

1.6.4 USBExt3c Software Features

To enable a newly-licensed software feature, it is necessary to *update the firmware* of the USBExt3c even if you are at on the current version.

PD Builder

The PD Builder feature allows the user to modify all *Power Data Objects (PDO)* presented by the USBExt3c. This includes both sourcing and sinking PDOs of the USBExt3c.

Use Cases

- Designing PDOs for you own device
- Exposing DUTs to PDOs
- Restricting DUTs from PDOs

Editable Items

PDO Types	PDO Flags	PDO Limits
Fixed	Unchuncked message support	Voltage
Variable	Dual role data	Current
Battery	USB communications possible	Power
APDO	External power	
	USB suspend	
	Dual role power	
	High capability	
	Fast role-swap current	

HubTool

HubTool allows the user to visually build a PDO without needing to know anything about the Power Delivery specification. Once created you can immediately apply it to the USBExt3c.

Port: 1 ⌵ Power Type: Local Source Rules ⌵

En	Rule	PDO Type	Min Voltage (mV)	Max Voltage (mV)	Power (mW)	Current (mA)	Raw	Set Rule	Set Default
<input checked="" type="checkbox"/>	1	Fixed	5000	5000		3000	0x3F01912C	Set	Reset
<input checked="" type="checkbox"/>	2	Fixed	9000	9000		3000	0x0002D12C	Set	Reset
<input checked="" type="checkbox"/>	3	Variable	3300	21000		5000	0x9A4109F4	Set	Reset
<input checked="" type="checkbox"/>	4	Battery	<input type="text" value="3400"/>	21000	100000		0x5A410990	Set	Reset
<input checked="" type="checkbox"/>	5	ADPO	3000	21000		5000	0xC1A41E64	Set	Reset
<input type="checkbox"/>	6	Fixed	0	0		0	0x00000000	Set	Reset
<input type="checkbox"/>	7	Fixed	0	0		0	0x00000000	Set	Reset

Example

C++

```

static const int TEST_PORT = 1;
static const uint32_t MY_CUSTOM_SOURCE_PDO = 0x0001912C;
static const uint32_t MY_CUSTOM_SINK_PDO = 0x0002D12C;

aUSBExt3c stem;
stem.discoverAndConnect(USB);

//Change local SOURCE PDO 1 to MY_CUSTOM_SOURCE_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSource, 1, MY_CUSTOM_
↳SOURCE_PDO)

//Change local SINK PDO 1 to MY_CUSTOM_SINK_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSink, 1, MY_CUSTOM_SINK_
↳PDO)

//Do Stuff

stem.disconnect()

```

Python

```

TEST_PORT = 1;
MY_CUSTOM_SOURCE_PDO = 0x0001912C;
MY_CUSTOM_SINK_PDO = 0x0002D12C;

stem = brainstem.stem.USBExt3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

#Change local SOURCE PDO 1 to MY_CUSTOM_SOURCE_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSource, 1, MY_CUSTOM_
↳SOURCE_PDO)

#Change local SINK PDO 1 to MY_CUSTOM_SINK_PDO
stem.pd[TEST_PORT].setPowerDataObject(powerdeliveryPowerRoleSink, 1, MY_CUSTOM_SINK_
↳PDO)

#Do Stuff

stem.disconnect()

```

Relevant API's

```

stem.hub.pd[x].setPowerDataObject() [cpp] [python] [NET] [LabVIEW]
stem.hub.pd[x].getPowerDataObject() [cpp] [python] [NET] [LabVIEW]

```

```

stem.hub.pd[x].setPowerDataObjectEnabled() [cpp] [python] [NET] [LabVIEW]
stem.hub.pd[x].getPowerDataObjectEnabled() [cpp] [python] [NET] [LabVIEW]

```

PD Logging

The PD Logging feature allows you to monitor Power Delivery communication on ports 0,1, and Control of the USBExt3c.

Use Cases

- Capture Power Delivery (PD) events across all USB-C ports
- Decode PD messages
- Export PD message logs to CSV
- Filter USB-PD traffic by message type.
- Clearly show PD message direction.
- Send arbitrary Vendor Defined Messages (VDMs)

HubTool

With HubTool you can easily visualize the Power Delivery log.

The screenshot displays the HubTool application window. At the top, 'System Information' shows details for device 0x4F7A0C15, including SN, Model (24), Firmware (2.9.6), Module (6), Voltage (20.0 VDC), Current (0.5 A), Temperature (30 C), Hardware Offset (0), Router (6), and SW Offset (0). Below this is a row of tabs for different system components: Summary, Port 0, Port 2, Port 3, Port 4, Port 5, Control, Power C, IO Expander, Power, and PD Logging. The 'PD Logging' tab is selected. Underneath the tabs are buttons for 'Start', 'Stop', and 'Export', followed by checkboxes for each component, all of which are checked. The main area contains a table of PD log entries with 25 rows. The table has columns for Timestamp (S:uS), Port, Direction, Spec, SOP, Power Role, Data Role, ID, Packet Type, Msg Type, and Raw data. The log shows various messages including Source Capabilities, Request, Accept, PS Ready, Vendor Defined, and DR Swap. At the bottom, there is a section for 'Software Feature' (Rail Enable, QC Mode, PDO Editing, VBUS Validation, PD Logging) and a 'Device Type' / 'Serial Number' section showing USBHub3c 4F7A0C15 and USBHub3p AF62130D.

	Timestamp (S:uS)	Port	Direction	Spec	SOP	Power Role	Data Role	ID	Packet Type	Msg Type	Raw
1	1429:635630	1	RX	V2.0	SOP	Source	DFP	0	Data	Source Capabilities	0x61 0x11 0x96 0x90 0x01 0x36
2	1429:645760	1	TX	V2.0	SOP	Sink	UFP	0	Data	Request	0x42 0x10 0x64 0x90 0x01 0x10
3	1429:654670	1	RX	V2.0	SOP	Source	DFP	1	Control	Accept	0x63 0x03
4	1429:680910	1	RX	V2.0	SOP	Source	DFP	2	Control	PS Ready	0x66 0x05
5	1429:690560	1	RX	V2.0	SOP	Source	DFP	3	Data	Vendor Defined	0x6F 0x17 0x01 0x80 0x00 0xFF
6	1429:701040	1	TX	V2.0	SOP	Sink	UFP	1	Data	Vendor Defined	0x4F 0x72 0x41 0x80 0x00 0xFF 0xFF 0x24 0xA...
7	1429:707240	1	RX	V2.0	SOP	Source	DFP	4	Data	Vendor Defined	0x6F 0x19 0x02 0x80 0x00 0xFF
8	1429:712110	1	TX	V2.0	SOP	Sink	UFP	2	Data	Vendor Defined	0x4F 0x24 0x42 0x80 0x00 0xFF 0x00 0x00 0xF...
9	1429:717770	1	RX	V2.0	SOP	Source	DFP	5	Data	Vendor Defined	0x6F 0x1B 0x02 0x80 0x00 0xFF
10	1429:724220	1	TX	V2.0	SOP	Sink	UFP	3	Data	Vendor Defined	0x4F 0x26 0x42 0x80 0x00 0xFF 0x00 0x00 0xF...
11	1429:948940	1	TX	V2.0	SOP	Sink	UFP	4	Control	VConn Swap	0x4B 0x08
12	1429:957240	1	RX	V2.0	SOP	Source	DFP	6	Control	Accept	0x63 0x0D
13	1430:18980	1	TX	V2.0	SOP	Sink	UFP	5	Control	PS Ready	0x46 0x0A
14	1430:72450	1	TX	V2.0	S...	Sink	UFP	0	Data	Vendor Defined	0x4F 0x10 0x01 0x80 0x00 0xFF
15	1430:158980	1	TX	V2.0	S...	Sink	UFP	1	Data	Vendor Defined	0x4F 0x12 0x01 0x80 0x00 0xFF
16	1430:168330	1	RX	V2.0	S...	Source	UFP	1	Data	Vendor Defined	0x4F 0x53 0x41 0x80 0x00 0xFF 0x11 0x07 0x00...
17	1430:262130	1	TX	V2.0	SOP	Sink	UFP	6	Control	DR Swap	0x49 0x0C
18	1430:269210	1	RX	V2.0	SOP	Source	DFP	7	Control	Accept	0x63 0x0F
19	1430:515440	1	TX	V2.0	SOP	Sink	DFP	7	Control	PR Swap	0x6A 0x0E
20	1430:523230	1	RX	V2.0	SOP	Source	UFP	0	Control	Accept	0x43 0x01
21	1430:625040	1	RX	V2.0	SOP	Sink	UFP	1	Control	PS Ready	0x46 0x02
22	1430:741540	1	TX	V2.0	SOP	Source	DFP	0	Control	PS Ready	0x66 0x01
23	1430:903350	1	TX	V2.0	SOP	Source	DFP	0	Data	Source Capabilities	0x61 0x51 0x2C 0x91 0x01 0x3E 0x2C 0xD1 0x02...
24	1430:915290	1	RX	V2.0	SOP	Sink	UFP	0	Data	Request	0x42 0x10 0x2C 0xB1 0x04 0x13
25	1430:919090	1	TX	V2.0	SOP	Source	DFP	1	Control	Accept	0x63 0x03

Software Feature: Rail Enable: Enabled
 Software Feature: QC Mode: Enabled
 Software Feature: PDO Editing: Enabled
 Software Feature: VBUS Validation: Enabled
 Software Feature: PD Logging: Enabled
 USBHub3c: 0x4F7A0C15, Error: 7 CMD: stem.hub.port[x].setVoltageSetpoint

Device Type: | Serial Number:
 USBHub3c 4F7A0C15
 USBHub3p AF62130D

Relevant API's

```
stem.pd[x].setUEIBytes() [cpp] [python]
```

VBus Validation

The VBus Validation software feature gives the user full control of current limit and voltage setpoint for ports 0,1, and Control. This feature can be used in two different applications; Within the Power Delivery specification or as a fully programmable power supply.

Use Cases

- Over-voltage testing
- Under-voltage testing
- 3-channel bench top power supply

Note: This feature has the ability to damage your device. Acroname is not responsible for any damage incurred by using this feature.

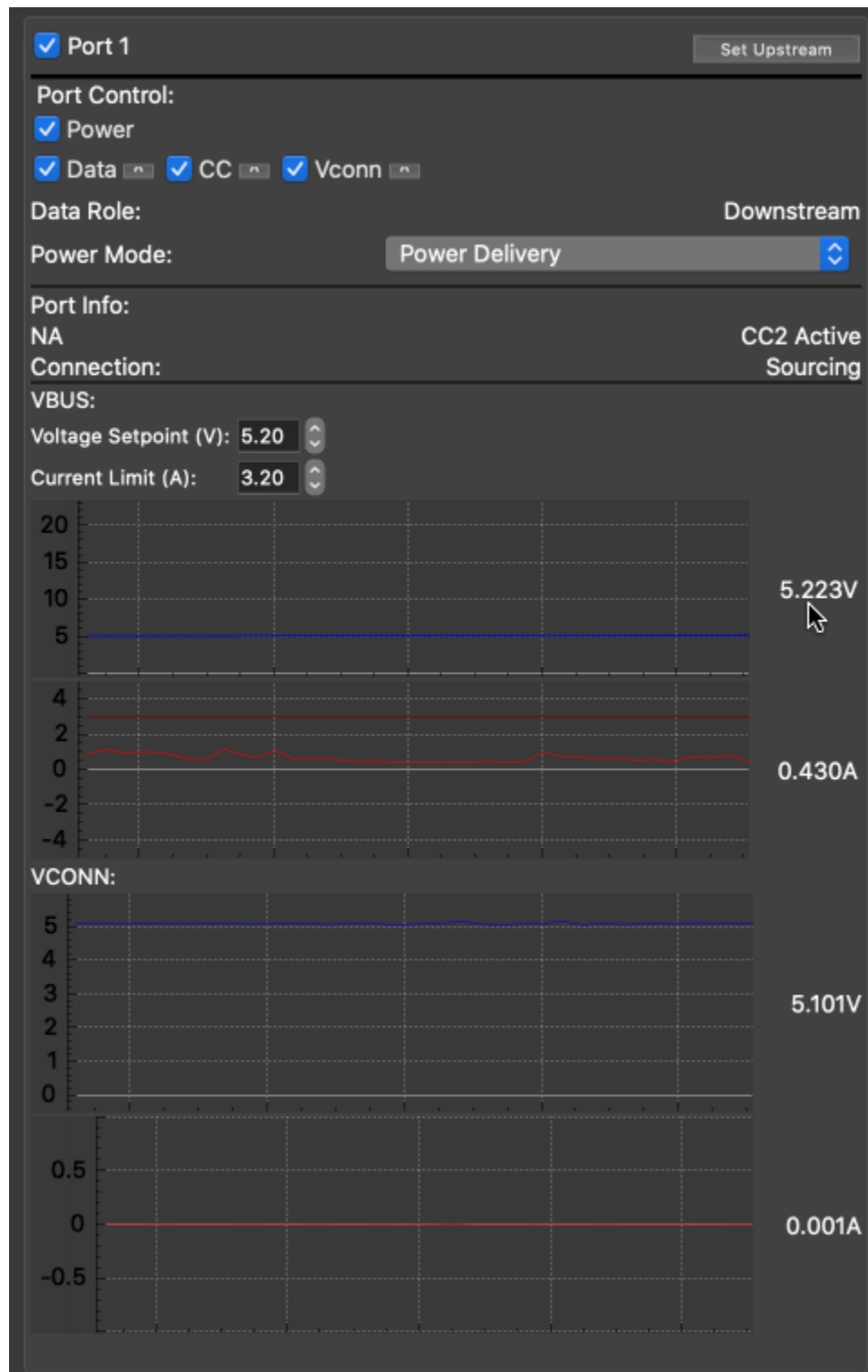
This software feature can be exploited through the *USBExt3c Port Entity*

VBus Validation - Power Delivery Mode

Using the VBus Validation feature within the power delivery mode allows the user to test if their device responds properly when a power source is behaving incorrectly or operating at the edge of specification.

It is important to remember that in this mode the USBExt3c 's power delivery engine also has access to these controls and therefore it is important to allow the USBExt3c and the device to finish negotiations before adjusting these values. Additionally, any PD events or errors can trigger re-negotiations which will replace any values the user has set.

This mode should only be used when the Power Delivery connection state is sourcing.



Example

C++

```
static const int TEST_PORT = 1;
```

(continues on next page)

(continued from previous page)

```

aUSBExt3c stem;
stem.discoverAndConnect(USB);

//Configure the Power mode: Power Delivery (default)
stem.hub.port[TEST_PORT].setPowerMode(portPowerMode_pd_Value);

//Check if we are sourcing power
uint8_t connectionState = 0;
stem.pd[TEST_PORT].getConnectionState(&connectionState);

//Ensure we have an RDO from the remote.
//This ensures we have finished negotiating.
uint32_t rdo = 0;
stem.pd[TEST_PORT].getRequestDataObject(powerdeliveryPartnerRemote, &rdo);

if((connectionState == powerdeliveryPowerRoleSource) &&
    (rdo > 0))
{
    //Do Stuff
    //Set desired port voltage and limit.
    stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000); //5.5VDC
    stem.hub.port[TEST_PORT].setCurrentLimit(500000); //500mA
    //Do Stuff
}

stem.disconnect()

```

Python

```

TEST_PORT = 1;

stem = brainstem.stem.USBExt3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

#Configure the Power mode: Power Delivery (default)
stem.hub.port[TEST_PORT].setPowerMode(_BS_C.portPowerMode_pd_Value);

#Check if we are sourcing power
connection_state_result = stem.pd[TEST_PORT].getConnectionState();

#Ensure we have an RDO from the remote.
#This ensures we have finished negotiating.
rdo_result = stem.pd[TEST_PORT].getRequestDataObject(_BS_C.
    powerdeliveryPartnerRemote);

if ((connection_state_result.value == _BS_C.powerdeliveryPowerRoleSource) and
    (rdo_result.value > 0)):

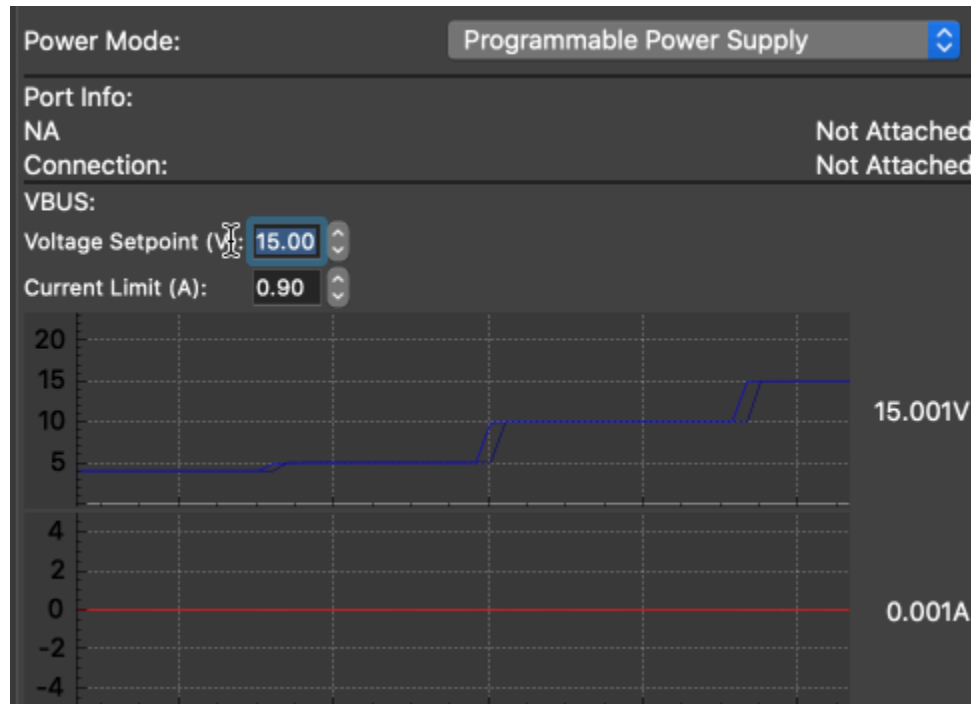
    #Do Stuff
    #Set desired port voltage and limit.
    stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000); #5.5VDC
    stem.hub.port[TEST_PORT].setCurrentLimit(500000); #500mA
    #Do Stuff

stem.disconnect()

```

VBus Validation - Programmable Power Supply Mode

In this mode the USBExt3c is transformed into a 6 channel programmable power supply capable of supplying 100 Watts per channel.



Example

C++

```
static const int TEST_PORT = 1;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

//Disable port while we configure
stem.hub.port[TEST_PORT].setEnabled(false);

//Configure the Power mode: Programmable Power Supply
stem.hub.port[TEST_PORT].setPowerMode(portPowerMode_ps_Value);

//Set desired port voltage and limit.
stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000); //5.5VDC
stem.hub.port[TEST_PORT].setCurrentLimit(500000); //500mA

//enable the port when ready.
stem.hub.port[TEST_PORT].setEnabled(true);

//Do Stuff

//Return port to safe state.
stem.hub.port[TEST_PORT].setEnabled(false)

stem.disconnect()
```


Python

```

TEST_PORT = 1;

stem = brainstem.stem.USBHub3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

#Disable port while we configure
stem.hub.port[TEST_PORT].setEnabled(false)

//Configure the Power mode: Programmable Power Supply
stem.hub.port[TEST_PORT].setPowerMode(_BS_C.portPowerMode_ps_Value);

#Set desired port voltage and limit.
stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000)      #5.5VDC
stem.hub.port[TEST_PORT].setCurrentLimit(500000)          #500mA

#Enable the port when ready.
stem.hub.port[TEST_PORT].setEnabled(true)

#Do Stuff

#Return port to safe state.
stem.hub.port[TEST_PORT].setEnabled(false)

stem.disconnect()

```

Relevant API's

```

stem.hub.port[x].setVoltageSetpoint() [cpp] [python] [NET] [LabVIEW] stem.hub.
port[x].getVoltageSetpoint() [cpp] [python] [NET] [LabVIEW]

stem.hub.port[x].setCurrentLimit() [cpp] [python] [NET] [LabVIEW] stem.hub.
port[x].getCurrentLimit() [cpp] [python] [NET] [LabVIEW]

stem.hub.port[x].setPowerMode() [cpp] [python] [NET] [LabVIEW] stem.hub.port[x].
getPowerMode() [cpp] [python] [NET] [LabVIEW]

stem.hub.pd[x].setRequestDataObject() [cpp] [python] [NET] [LabVIEW] stem.hub.
pd[x].getRequestDataObject() [cpp] [python] [NET] [LabVIEW]

stem.hub.pd[x].setConnectionState() [cpp] [python] [NET] [LabVIEW] stem.hub.
pd[x].getConnectionState() [cpp] [python] [NET] [LabVIEW]

```

RS232 Serial Communication

The RS232 Serial Communication feature allows one to send commands to control and configure USBExt3c.

Use Cases

- Affecting USBExt3c control.
- Audio/Video applications.

Configuration

The default configuration of the RS232 Serial Communication feature is:

- 8 Data bits
- No Parity
- No Flow Control
- 1 Stop bit
- 9600 Baudrate

This feature does have some configurability through the *USBExt3c UART Entity*

Extron Compatible Serial Commands

The RS232 Serial Communication feature is capable of changing the USBExt3c upstream port, requesting the current status of the upstream/downstream connections, enable/disable of ports, and the USBExt3c part number/firmware version queries. This can be accomplished with a protocol that is compatible with Extron's \$Simple Instruction Set\$ over RS232.

Commands

The following is a list of all commands the USBExt3c supports with their arguments, descriptions, and expected responses.

Error Codes

The following is a list of all error codes the USBExt3c supports with descriptions.

Code	Description
E01	invalid port number, check the port number and make sure it's valid.
E10	invalid command, verify that you formatted the command correctly.
E13	invalid value, verify that the value is within the acceptable range for this command.
E14	invalid configuration, verify the system is in a state it can accept this command.

General Notes

- All commands are ASCII strings.
- \r is the ASCII character for carriage return.
- \n is the ASCII character for new line.

Examples

Extron Compatible Serial Commands:

Upstream Change

Change Upstream to Port 1

Tx: 1!

Rx: Chn 1\r\n

Port Status

Get Port status with Upstream set to port 1, an upstream device on port 1 and Downstream device on port 2

Tx: I

Rx: Chn 1 InACT010000 OutACT001000\r\n

Port Disable

Disable Port 3

Tx: 3*0P

Rx: Port 3*0\r\n

API Configurations:

C++

```
static const int TEST_SERIAL = 0;

aUSBExt3c stem;
stem.discoverAndConnect(USB);

// Enable the port
stem.uart[TEST_SERIAL].setEnabled(true);

// Change baudrate to 115200 from default 9600
stem.uart[TEST_SERIAL].setBaudRate(115200);

// Change Protocol to Extron Compatible
// 0 - Disabled/Undefined
// 1 - Extron Compatible
// 2 - Brainstem Transport
// 6 - Loopback
stem.uart[TEST_SERIAL].setProtocol(1)

// Perform a system save so the changes persist
```

(continues on next page)

(continued from previous page)

```
// through power cycles
stem.system.save();

stem.disconnect();
```

Python

```
TEST_SERIAL = 0

stem = brainstem.stem.USBxt3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

# Enable the port
stem.uart[TEST_SERIAL].setEnabled(1)

# Change baudrate to 115200 from default 9600
stem.uart[TEST_SERIAL].setBaudRate(115200)

# Change Protocol to Extron Compatible
# 0 - Disabled/Undefined
# 1 - Extron Compatible
# 2 - Brainstem Transport
# 6 - Loopback
stem.uart[TEST_SERIAL].setProtocol(1)

# Perform a system save so the changes persist
# through power cycles
stem.system.save()

stem.disconnect()
```

Loopback Protocol

```
# Configure UART[0] for loopback testing
# Data sent to the UART will be echoed back
TEST_SERIAL = 0

stem = brainstem.stem.USBxt3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

stem.uart[TEST_SERIAL].setEnabled(1)
stem.uart[TEST_SERIAL].setBaudRate(115200)
stem.uart[TEST_SERIAL].setProtocol(6) # Loopback protocol

# Now any data sent to UART[0] will be echoed back
stem.system.save()
stem.disconnect()
```

VCOM to Hardware UART Link

```
# Link VCOM channel (UART[4]) to hardware RS232 (UART[0])
# This allows USB serial communication to be forwarded to RS232
stem = brainstem.stem.USBxt3c()
stem.discoverAndConnect(brainstem.link.Spec.USB)

# Enable both channels
```

(continues on next page)

(continued from previous page)

```

stem.uart[0].setEnabled(1)    # Hardware RS232
stem.uart[4].setEnabled(1)    # VCOM_0 channel

# Configure matching baud rates
stem.uart[0].setBaudRate(115200)
stem.uart[4].setBaudRate(115200)

# Link VCOM to hardware UART
stem.uart[0].setLinkChannel(4)
stem.uart[4].setLinkChannel(0)

# Data received on VCOM will now be forwarded to RS232,
# and data received on RS232 will be forwarded to VCOM
stem.system.save()
stem.disconnect()

```

Loopback Protocol

```

// Configure UART[0] for loopback testing
// Data sent to the UART will be echoed back
static const int TEST_SERIAL = 0;

aUSBExt3c stem;
stem.discoverAndConnect(USB);

stem.uart[TEST_SERIAL].setEnabled(true);
stem.uart[TEST_SERIAL].setBaudRate(115200);
stem.uart[TEST_SERIAL].setProtocol(6); // Loopback protocol

// Now any data sent to UART[0] will be echoed back
stem.system.save();
stem.disconnect();

```

VCOM to Hardware UART Link

```

// Link VCOM channel (UART[4]) to hardware RS232 (UART[0])
// This allows USB serial communication to be forwarded to RS232
aUSBExt3c stem;
stem.discoverAndConnect(USB);

// Enable both channels
stem.uart[0].setEnabled(true); // Hardware RS232
stem.uart[4].setEnabled(true); // VCOM_0 channel

// Configure matching baud rates
stem.uart[0].setBaudRate(115200);
stem.uart[4].setBaudRate(115200);

// Link VCOM to hardware UART
stem.uart[0].setLinkChannel(4);
stem.uart[4].setLinkChannel(0);

// Data received on VCOM will now be forwarded to RS232,
// and data received on RS232 will be forwarded to VCOM
stem.system.save();
stem.disconnect()

```

Relevant API's

```
stem.uart[x].setEnabled() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getEnable() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setBaudRate() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getBaudRate() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setProtocol() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getProtocol() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setLinkChannel() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getLinkChannel() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setStopBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getStopBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setParity() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getParity() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].setDataBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getDataBits() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getCapableProtocols() [cpp] [python] [NET] [LabVIEW]
stem.uart[x].getAvailableProtocols() [cpp] [python] [NET] [LabVIEW]
```

1.6.5 USBExt3c Module Entities

Ethernet Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Ethernet Entity provides control over network configuration and monitoring for devices with Ethernet connectivity. This includes IP address management, network configuration modes, and interface settings.

Ethernet Enable/Disable (Get/Set)

```
ethernet . getEnabled <= (unsigned char) enabled
ethernet . setEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Ethernet interface. When enabled, the Ethernet interface is active and can be used for network communication.

Network Configuration (Get/Set)

```
ethernet . getNetworkConfiguration <= (unsigned char) configuration
ethernet . setNetworkConfiguration => (unsigned char) configuration
```

Returns or sets the network configuration mode for the Ethernet interface. This controls whether the interface uses DHCP or static IP configuration.

Table 26: Network Configuration Options

Value	Name	Description
0	None	No network configuration
1	Static	Static IP configuration
2	DHCP	DHCP automatic configuration

Static IPv4 Address (Get/Set)

```
ethernet . getStaticIPv4Address <= (unsigned char[]) address
ethernet . setStaticIPv4Address => (unsigned char[]) address
```

Returns or sets the static IPv4 address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the IP address (e.g., [192, 168, 1, 100] for “192.168.1.100”).

Static IPv4 Netmask (Get/Set)

```
ethernet . getStaticIPv4Netmask <= (unsigned char[]) netmask
ethernet . setStaticIPv4Netmask => (unsigned char[]) netmask
```

Returns or sets the static IPv4 netmask for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the netmask (e.g., [255, 255, 255, 0] for “255.255.255.0”).

Static IPv4 Gateway (Get/Set)

```
ethernet . getStaticIPv4Gateway <= (unsigned char[]) gateway
ethernet . setStaticIPv4Gateway => (unsigned char[]) gateway
```

Returns or sets the static IPv4 gateway address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the gateway address (e.g., [192, 168, 1, 1] for “192.168.1.1”).

IPv4 Address (Get)

```
ethernet . getIPv4Address <= (unsigned char[]) address
```

Returns the current IPv4 address of the Ethernet interface as a 4-byte array. This reflects the actual address being used, whether obtained via DHCP or set statically. Each byte represents one octet of the IP address.

IPv4 Netmask (Get)

```
ethernet . getIPv4Netmask <= (unsigned char[]) netmask
```

Returns the current IPv4 netmask of the Ethernet interface as a 4-byte array. This reflects the actual netmask being used, whether obtained via DHCP or set statically. Each byte represents one octet of the netmask.

IPv4 Gateway (Get)

```
ethernet . getIPv4Gateway <= (unsigned char[]) gateway
```

Returns the current IPv4 gateway address of the Ethernet interface as a 4-byte array. This reflects the actual gateway being used, whether obtained via DHCP or set statically. Each byte represents one octet of the gateway address.

Static IPv4 DNS Address (Get/Set)

```
ethernet . getStaticIPv4DNSAddress <= (unsigned char[]) dns  
ethernet . setStaticIPv4DNSAddress => (unsigned char[]) dns
```

Returns or sets the static IPv4 DNS server address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the DNS address.

IPv4 DNS Address (Get)

```
ethernet . getIPv4DNSAddress <= (unsigned char[]) dns
```

Returns the current IPv4 DNS server address of the Ethernet interface as a 4-byte array. This reflects the actual DNS server being used, whether obtained via DHCP or set statically. Each byte represents one octet of the DNS address.

Hostname (Get/Set)

```
ethernet . getHostname <= (unsigned char[]) hostname  
ethernet . setHostname => (unsigned char[]) hostname
```

Returns or sets the hostname for the Ethernet interface. The hostname is used for network identification and can be up to 63 characters long.

MAC Address (Get)

```
ethernet . getMACAddress <= (unsigned char[]) mac
```

Returns the MAC address of the Ethernet interface. This is a unique hardware identifier for the network interface.

Interface Port (Get/Set)

```
ethernet . getInterfacePort (unsigned char) service <= (unsigned short) port
ethernet . setInterfacePort => (unsigned char) service, (unsigned short) port
```

Returns or sets the interface port for specific services on the Ethernet interface. The service parameter specifies which service to configure, and the port parameter sets the port number for that service.

Table 27: Interface Port Services

Name	Value	Description
RestServer_HTTP	0	HTTP REST API server
RestServer_HTTPS	1	HTTPS REST API server
BrainStem_TCP	2	BrainStem TCP communication
Brain-Stem_DiscoveryRequest	3	BrainStem discovery request port
Brain-Stem_DiscoveryReply	4	BrainStem discovery reply port

Examples

C++

```
// Enable Ethernet interface
ethernet.setEnabled(1);

// Set static IP configuration
ethernet.setNetworkConfiguration(1); // Static mode
unsigned char ip[4] = {192, 168, 1, 100};
unsigned char netmask[4] = {255, 255, 255, 0};
unsigned char gateway[4] = {192, 168, 1, 1};
ethernet.setStaticIPv4Address(ip, 4);
ethernet.setStaticIPv4Netmask(netmask, 4);
ethernet.setStaticIPv4Gateway(gateway, 4);

// Get current IP address
unsigned char current_ip[4];
ethernet.getIPv4Address(current_ip, 4);

// Set hostname
ethernet.setHostname("brainstem-device");
```

Python

```
# Enable Ethernet interface
stem.ethernet.setEnabled(1)

# Set static IP configuration
stem.ethernet.setNetworkConfiguration(1) # Static mode
ip = [192, 168, 1, 100]
netmask = [255, 255, 255, 0]
gateway = [192, 168, 1, 1]
stem.ethernet.setStaticIPv4Address(ip)
stem.ethernet.setStaticIPv4Netmask(netmask)
stem.ethernet.setStaticIPv4Gateway(gateway)

# Get current IP address
current_ip = stem.ethernet.getIPv4Address()
print(current_ip.value)

# Set hostname
stem.ethernet.setHostname("brainstem-device")
```

HDBaseT Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The **HDBaseT** entity provides link telemetry, remote-end USB topology discovery, and role control for the HDBaseT-USB3 link.

Identification

`stem.system.getSerialNumber()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Fills a caller-provided buffer with up to 6 bytes and reports the actual length unloaded.

`stem.hdbaset.getFirmwareVersion()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Returns a packed 32-bit value (Major: Bits 24-31; Minor: Bits 16-23; Patch: Bits 8-15; Build: Bits 0-7)

Link Status and Health

`stem.hdbaset.getState()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Returns a packed integer describing current link state.

`stem.hdbaset.getCableLength()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Perceived cable length in μm .

`stem.hdbaset.getMSEA()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

`stem.hdbaset.getMSEB()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Mean Squared Error for channel A or B in μdB .

`stem.hdbaset.getRetransmissionRate()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Number of successful messages between retransmission; 0 = no errors, otherwise higher is better.

`stem.hdbaset.getLinkUtilization()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Link utilization in milli-percent (1000 = saturation).

`stem.hdbaset.getEncodingState()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Encoding state (e.g. PAM 8).

Remote USB Topology (Device Trees)

```
stem.hdbaset.getUSB2DeviceTree() [cpp] [python] [.NET] [LabVIEW]
stem.hdbaset.getUSB3DeviceTree() [cpp] [python] [.NET] [LabVIEW]
```

These calls return bytes describing the USB2/USB3 device trees at the HDBaseT endpoint.

Link Role Control

```
stem.hdbaset.getLinkRole() [cpp] [python] [.NET] [LabVIEW]
stem.hdbaset.setLinkRole() [cpp] [python] [.NET] [LabVIEW]
```

Role values are product/firmware specific (e.g., *Auto Detect*, *Leader*, *Follower*).

When in *Auto Detect*, the active role can be read using `getState()`.

Legacy Support

Previous Acroname USB products made use of the *USB Entity* for measurements, configuration and control; however, the USBExt3c aims to organize these capabilities in a cleaner fashion through the use of the all new *Port* and *USBSysytem* Entities.

In efforts to ease this transition we have added Legacy support to the USBExt3c by implementing limited USB Entity functionally. The following functions of the *USB Entity* will still behave as they did in previous Acroname USB products.

Port

```
stem.usb.setPortEnable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setPortDisable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.getPortState(channel) [cpp] [python] [NET] [LabVIEW] [REST]
```

Port State Item	Bit	Value
VBus is enabled	6	0/1
USB2A is enabled	4	0/1
USB2B is enabled	5	0/1
SuperSpeedA is enabled	7	0/1
SuperSpeedB is enabled	8	0/1
CC1 is enabled	12	0/1
CC2 is enabled	13	0/1

Power

```
stem.usb.setPowerEnable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setPowerDisable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
```

Voltage

```
stem.usb.getPortVoltage(channel,  $\mu$ V) [cpp] [python] [NET] [LabVIEW] [REST]
```

Current

```
stem.usb.getPortCurrent(channel,  $\mu$ A) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.getPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setPortCurrentLimit(channel,  $\mu$ A) [cpp] [python] [NET] [LabVIEW] [REST]
```

Data

```
stem.usb.setDataEnable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setDataDisable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setHiSpeedDataEnable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setHiSpeedDataDisable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setSuperSpeedDataEnable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setSuperSpeedDataDisable(channel) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setCC1Enable(channel, enable) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.getCC1Enable(channel, enable) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.setCC2Enable(channel, enable) [cpp] [python] [NET] [LabVIEW] [REST]
stem.usb.getCC2Enable(channel, enable) [cpp] [python] [NET] [LabVIEW] [REST]
```

PoE Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The **PoE** entity provides Power over Ethernet control and telemetry on supported products. Use this entity to enable/disable PoE pairs, select power modes, inspect negotiated classes, and read per-pair electrical measurements and accumulated power.

- *pair* = 0 = wire pair 1/2
- *pair* = 1 = wire pair 3/4

Pair Enable/Disable

```
stem.poe.getPairEnabled(pair) [cpp] [python] [.NET] [LabVIEW]
stem.poe.setPairEnabled(pair, enable) [cpp] [python] [.NET] [LabVIEW]
```

1 = Enabled, 0 = Disabled.

Power Mode and Power State

```
stem.poe.getPowerMode() [cpp] [python] [.NET] [LabVIEW]
stem.poe.setPowerMode(value) [cpp] [python] [.NET] [LabVIEW]
stem.poe.getPowerState() [cpp] [python] [.NET] [LabVIEW]
```

Power Mode (commanded): PD, PSE, Auto, Off.
Power State (actual): PD, PSE, Off.

Classification and Detection

```
stem.poe.getPairSourcingClass(pair) [cpp] [python] [.NET] [LabVIEW]
stem.poe.setPairSourcingClass(pair, value) [cpp] [python] [.NET] [LabVIEW]
```

Sourcing class: The PoE class offered by the device when acting as a PSE.

```
stem.poe.getPairRequestedClass(pair) [cpp] [python] [.NET] [LabVIEW]
```

Requested class: The PoE class requested by the device when acting as a PD.

```
stem.poe.getPairDiscoveredClass(pair) [cpp] [python] [.NET] [LabVIEW]
```

Discovered class (PSE/PD): The negotiated PoE class result.

```
stem.poe.getPairDetectionStatus(pair) [cpp] [python] [.NET] [LabVIEW]
```

Current per-pair detection status.

Table 28: Detection status values

Unknown
Short circuit
Open circuit
Low resistance
High resistance
Valid
Switch Failure

Electrical Measurements (per pair)

```
stem.poe.getPairVoltage(pair) [cpp] [python] [.NET] [LabVIEW]
```

Voltage: μV

```
stem.poe.getPairCurrent(pair) [cpp] [python] [.NET] [LabVIEW]
```

Current: μA

```
stem.poe.getPairResistance(pair) [cpp] [python] [.NET] [LabVIEW]
```

Resistance: $\text{m}\Omega$

```
stem.poe.getPairCapacitance(pair) [cpp] [python] [.NET] [LabVIEW]
```

Capacitance: nF

```
stem.poe.getPairPower(pair) [cpp] [python] [.NET] [LabVIEW]
```

Power: mW (approx. Voltage * Current for the pair)

Total Power (instantaneous)

```
stem.poe.getTotalPower() [cpp] [python] [.NET] [LabVIEW]
```

Returns the sum of both pairs' instantaneous power (mW).

Accumulated Power (energy counters)

```
stem.poe.getPairAccumulatedPower(pair) [cpp] [python] [.NET] [LabVIEW]
```

```
stem.poe.setPairAccumulatedPower(pair, power) [cpp] [python] [.NET] [LabVIEW]
```

```
stem.poe.getTotalAccumulatedPower() [cpp] [python] [.NET] [LabVIEW]
```

```
stem.poe.setTotalAccumulatedPower(power) [cpp] [python] [.NET] [LabVIEW]
```

* All values are in mWh

* Setting total or pair accumulated power resets the corresponding accumulator.

Port Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Port Entity provides control over the most basic items related to a USB Port. This includes actions ranging from a complete port enable and disable to the individual interface control. Voltage and current measurements are also included for devices which support the Port Entity.

Port Control

USBExt3c has a Port Entity for every Type C port on the device; however, not all ports have the same capabilities. These ports can be referenced by their instance (port[x]) index.

Port Label	Index (port[x])
0	0
1	1
Control	2

One of the most powerful features of USBExt3c is its ability to turn Ports 0 and 1 entirely on and off.

```
stem.hub.port[x].setEnabled() [cpp] [python] [.NET] [LabVIEW] [REST]
```

```
stem.hub.port[x].getEnabled() [cpp] [python] [.NET] [LabVIEW] [REST]
```

USB V_{BUS} vsn be enabled or disabled per-port on ports 0, 1, and control.

```
stem.hub.port[x].setPowerEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getPowerEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Data lines (Hi-Speed data and SuperSpeed) can be enabled together per-port while not affecting V_{BUS} lines (ports 0 and 1).

```
stem.hub.port[x].setDataEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

USB 2.0 Hi-Speed data lines can be independently enabled / disabled per-port (ports 0 and 1).

```
stem.hub.port[x].setDataHSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataHSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Further granularity can be achieved by independently controlling Hi-Speed 1 and Hi-Speed 2 lines (Ports 0 and 1).

```
stem.hub.port[x].setDataHS1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataHS1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].setDataHS2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataHS2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

USB SuperSpeed data lines can be independently enabled / disabled per-port (ports 0 and 1).

```
stem.hub.port[x].setDataSSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataSSEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Just as with the Hi-Speed lines, USBExt3c also has granular control of the SuperSpeed 1 and SuperSpeed 2 lines.

```
stem.hub.port[x].setDataSS1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataSS1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].setDataSS2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getDataSS2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

CC lines can be enabled and disabled on all USB ports.

```
stem.hub.port[x].setCCEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCCEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

CC1 and CC@ can be controlled independently.

```
stem.hub.port[x].setCC1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].setCC2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCC2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Finally we come to V_{CONN} control; however, this one overlaps with CC control because V_{CONN} is what the unused CC line becomes (if needed). i.e. if CC1 is used for orientation and/or PD communication then CC2 will become V_{CONN} (V_{CONN} 2) if it is enabled. If you are unaware of which pin is being used for V_{CONN} you can simply call:

```
stem.hub.port[x].setVconnEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconnEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

and USBExt3c will take care of the guess work for you. If you are aware of which line is being used for V_{CONN} you can use the granular control just as we have outlined above.

```
stem.hub.port[x].setVconn1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconn1Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].setVconn2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconn2Enabled() [cpp] [python] [NET] [LabVIEW] [REST]
```

Voltage and Current Measurements

USBExt3c provides Voltage and Current measurements for both the V_{BUS} and V_{CONN} lines. These values can be acquired for all ports through the following APIs

```
stem.hub.port[x].getVbusVoltage() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVbusCurrent() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconnVoltage() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconnCurrent() [cpp] [python] [NET] [LabVIEW] [REST]
```

Power Modes

The USB-C ports of USBExt3c are capable of providing power in multiple formats. The default is Power Delivery (PD), but that can be changed to: Standard Downstream Port (SDP), Charging Downstream Port (CDP) / Dedicated Charging Port (DCP), or even Qualcomm Quick Charge (QC) 3 and 4. These modes can be set through:

```
stem.hub.port[x].setPowerMode() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getPowerMode() [cpp] [python] [NET] [LabVIEW] [REST]
```


Power Mode	Value	Define
None	0	portPowerMode_none_Value
SDP	1	portPowerMode_sdp_Value
CDP/DCP	2	portPowerMode_cdp_dcp_Value
QC	3	portPowerMode_qc_Value
PD	4	portPowerMode_pd_Value
PS	5	portPowerMode_ps_Value
USB-C	6	portPowerMode_usbc_Value

Note: The Power Modes can only be changed when the port power is disabled.

Port Mode

As outlined in the “Port Control” section USBExt3c can individually manipulate almost every pin on the Type-C connector; however, depending on your application that might require multiple function calls in order to configure the port how you want it. Port Mode on the other hand is a one stop shop that allows you to pick and choose which lines you want enabled or disabled through a single call. Additionally, it has a few other features tucked away inside of it.

```
stem.hub.port[x].setMode() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getMode() [cpp] [python] [NET] [LabVIEW] [REST]
```

Port Mode Item	Bit	Value	Define
Power Enable	0	0/1	portPortMode_powerEnabled_Bit
HS 1 Enable	1	0/1	portPortMode_HS1Enabled_Bit
HS 2 Enable	2	0/1	portPortMode_HS2Enabled_Bit
SS 1 Enable	3	0/1	portPortMode_SS1Enabled_Bit
SS 2 Enable	4	0/1	portPortMode_SS2Enabled_Bit
CC 1 Enable	5	0/1	portPortMode_CC1Enabled_Bit
CC 2 Enable	6	0/1	portPortMode_CC2Enabled_Bit
Vconn 1 Enable	7	0/1	portPortMode_Vconn1Enabled_Bit
Vconn 2 Enable	8	0/1	portPortMode_Vconn2Enabled_Bit
Power Mode: Offset		16	portPortMode_portPowerMode_Offset
Power Mode: Mask		0x7	portPortMode_portPowerMode_Mask
Power Mode: None	16-18	0	portPortMode_portPowerMode_none_Value
Power Mode: SDP	16-18	1	portPortMode_portPowerMode_sdp_Value
Power Mode: CDP/DCP	16-18	2	portPortMode_cdp_dcp_Value
Power Mode: QC	16-18	3	portPortMode_portPowerMode_qc_Value
Power Mode: PD	16-18	4	portPortMode_portPowerMode_pd_Value
Power Mode: PS	16-18	5	portPortMode_portPowerMode_ps_Value
Power Mode: USB-C	16-18	6	portPortMode_portPowerMode_usbc_Value

Data Role

The data role describes the current configuration of the port in regards to its data direction. In most cases this evaluates to an Upstream Facing Port (UFP) or a Downstream Facing Port (DFP). Upstream in this case means the host side of the port and Downstream refers to the device side. The Data Role can be aquired through:

```
stem.hub.port[x].getDataRole() [cpp] [python] [NET] [LabVIEW] [REST]
```

Data Role	Value	Define
Disabled	0	portDataRole_Disabled_Value
Upstream	1	portDataRole_Upstream_Value
Downstream	2	portDataRole_Downstream_Value
Control	3	portDataRole_Control_Value

Port Limits and Modes

At the Port level the user has the ability to define current limit and/or a power limit.

```
stem.hub.port[x].setCurrentLimit() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCurrentLimit() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].setPowerLimit() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getPowerLimit() [cpp] [python] [NET] [LabVIEW] [REST]
```

If either of these values are exceed then USBExt3c will then apply on of the following modes

```
stem.hub.port[x].setCurrentLimitMode() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getCurrentLimitMode() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].setPowerLimitMode() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getPowerLimitMode() [cpp] [python] [NET] [LabVIEW] [REST]
```

Available Power

One of the unique features of USBExt3c is its ability to manage input and output power. Because of smart charging technologies like PD we know exactly how much power we have access too. That input power must then be shared across all of the ports. The following function allows the user to request the amount of power that is currently allocated to the port in question.

```
stem.hub.port[x].getAvailablePower() [cpp] [python] [NET] [LabVIEW] [REST]
```

Accumulated Power

USBExt3c is capable of monitoring the accumulated power (energy) it has sank or sourced on both the V_{BUS} and V_{CONN} lines of each port. This value is presented as mWh.

```
stem.hub.port[x].getVbusAccumulatedPower() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.port[x].getVconnAccumulatedPower() [cpp] [python] [NET] [LabVIEW] [REST]
```

The accumulated power is set to zero and the accumulation period is restarted with these commands.

```
stem.hub.port[x].resetVbusAccumulatedPower() [cpp] [python] [NET] [LabVIEW]
[REST]
stem.hub.port[x].resetVconnAccumulatedPower() [cpp] [python] [NET] [LabVIEW]
[REST]
```

Downstream Data Speed

The USBExt3c can detect if a device has been enumerated. Additionally, it can detect at what speed a device has enumerated at.

```
stem.hub.port[x].getDataSpeed() [cpp] [python] [NET] [CCA] [REST]
```

Data Speed	Bit	Value	Define
1.5 Mbit/s	0	0/1	portDataSpeed_ls_1p5M_Bit
12 Mbit/s	1	0/1	portDataSpeed_fs_12M_Bit
480 Mbit/s	2	0/1	portDataSpeed_hs_480M_Bit
5 Gbit/s	3	0/1	portDataSpeed_ss_5G_Bit
10 Gbit/s	4	0/1	portDataSpeed_ss_10G_Bit
USB 2.0	6	0/1	portDataSpeed_Connected_2p0_Bit
USB 3.0	7	0/1	portDataSpeed_Connected_3p0_Bit

Power Delivery Entity

Manipulating PDO's and RDO's

The Power Delivery specification defines a large number of Data Objects and the USBExt3c is capable of manipulating and modifying most of them in some fashion. The most common are PDO's and RDO's which were defined above. Direct manipulation is quite a complex process and is a feature of the Pro version only. It is highly recommended that users of these features first experiment with the Power Rule Editor within HubTool. Once you know the values you want to use manipulation can be done through:

```
stem.pd[x].setPowerDataObject() [cpp] [python] [NET] [LabVIEW] [REST]  
stem.pd[x].getPowerDataObject() [cpp] [python] [NET] [LabVIEW] [REST]
```

```
stem.pd[x].setRequestDataObject() [cpp] [python] [NET] [LabVIEW] [REST]  
stem.pd[x].getRequestDataObject() [cpp] [python] [NET] [LabVIEW] [REST]
```

Power Roles Preferred

Preferred Power Role	Value	Define
None	0	pdPowerRolePreferred_None
Source	1	pdPowerRolePreferred_Source
Sink	2	pdPowerRolePreferred_Sink

Connection State

Connection State	Value	Define
None	0	pdConnectionState_None
Source	1	pdConnectionState_Source
Sink	2	pdConnectionState_Sink
Powered Cable	3	pdConnectionState_PoweredCable
Powered Cable with Sink	4	pdConnectionState_PoweredCableWithSink

Requests

Given the nature of Power Delivery there are only so many things that are within the direct control of the local USBExt3c. Many of the items on the remote side of the USBExt3c are merely request. In other words items in this category not guaranteed to happen.

```
stem.pd[x].request() [cpp] [python] [NET] [LabVIEW] [REST]
```

Request	Value	Define
Hard Reset	0	pdRequestHardReset
Soft Reset	1	pdRequestSoftReset
Data Reset	2	pdRequestDataReset
Power Role Swap	3	pdRequestPowerRoleSwap
Power Fast Role Swap	4	pdRequestPowerFastRoleSwap
Data Role Swap	5	pdRequestDataRoleSwap
Vconn Swap	6	pdRequestVconnSwap
Sink Go to Min	7	pdRequestSinkGoToMinimum
Remote Source PDOs	8	pdRequestRemoteSourcePowerDataObjects
Remote Sink PDOs	9	pdRequestRemoteSinkPowerDataObjects
Remote Source Extended Capabilities	10	pdRequestRemoteSourceExtendedCapabilities
Remote Sink Extended Capabilities	11	pdRequestRemoteSinkExtendedCapabilities
Status	12	pdRequestStatus
PPS Status	13	pdRequestPPSStatus
Battery Capabilities	14	pdRequestBatteryCapabilities
Battery Status	15	pdRequestBatteryStatus
Manufacturer Info Sop	16	pdRequestManufacturerInfoSop
Manufacturer Info Sopp	17	pdRequestManufacturerInfoSopp
Manufacturer Inof Soppp	18	pdRequestManufacturerInfoSoppp
Discover Identity Sop	19	pdRequestDiscoverIdentitySop
Discover Identity Sopp	20	pdRequestDiscoverIdentitySopp
Discover Identity Soppp	21	pdRequestDiscoverIdentitySoppp
Revision	22	pdRequestRevision
Source Info	23	pdRequestSourceInfo
Country Code	24	pdRequestCountryCode
Country Info	25	pdRequestCountryInfo

Errors return from this function call only indicate the success of sending the request and do not reflect the success of the actual request. To find the status of the request you can investigate the outcome of the connection or check the most recent status of the PD stack.

```
stem.pd[x].requestStatus() [cpp] [python] [NET] [LabVIEW] [REST]
```

Cable Orientation

Although the Type C connector has no visible orientation the connector does have electrical orientation which directly correlates to the Communications Chanel (CC) strapping internal to the cable. The orientation be be obtained via:

```
stem.pd[x].getCableOrientation() [cpp] [python] [NET] [LabVIEW] [REST]
```

Orientation	Value	Define
Invalid	0	pdCableOrientation_Invalid
CC1/A Side	1	pdCableOrientation_CC1
CC2/B Side	2	pdCableOrientation_CC2

Cable Type

Cable Type	Value	Define
Invalid	0	pdCableType_Invalid
Passive	1	pdCableType_Passive
Active	2	pdCableType_Active

Override

The USB C connector by default follows rules around maximum cable current and budgetted power. In some test applications, including ones with a Universal Orientation Cable, the port should ignore those rules which is why we have exposed override bits to allow for disabling of specific behavior. Below are the get/set routines for overrides and the bit definitions.

```
stem.pd[x].getOverride() [cpp] [python] [NET] [LabVIEW] [REST]
stem.pd[x].setOverride() [cpp] [python] [NET] [LabVIEW] [REST]
```

Name	Bit	Definition
Cable Current	0	overrides the cable current limiting to 3A unless it's an emarked cable
Port Power	1	Overrides the port power budgetting and just allows full power always
Auto Discovery	2	Overrides the auto discovery feature. With override true the hub will only establish a basic power connection and wont ask for additional vendor information.
Data path disable	3	Override normal disabling of the data path when the PD Data Role and port configured data role differ

System Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every USBExt3c is assigned a unique serial number at the factory. This facilitates an arbitrary number of USBExt3c devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber() \[cpp\] \[python\] \[NET\] \[LabVIEW\] \[REST\]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the USBExt3c away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. Use the following command to save changes to system settings before reboot:

```
stem.system.save() \[cpp\] \[python\] \[NET\] \[LabVIEW\] \[REST\]
```

System Power

The USBExt3c is designed to accept power from either a Type-C PD port or over PoE.

This value represents the maximum amount of power the USBExt3c can budget for all of its sinking devices. This value can be acquired through:

```
stem.system.getPowerLimit() \[cpp\] \[python\] \[NET\] \[LabVIEW\] \[REST\]
```

It is possible that the limit of your PD source may not match that of what is being advertised by the function above. The contributing factors can be determined by checking the state of the power limit.

```
stem.system.getPowerLimitState() \[cpp\] \[python\] \[NET\] \[LabVIEW\] \[REST\]
```

```
stem.system.getInputPowerSource() \[cpp\] \[python\] \[NET\] \[LabVIEW\] \[REST\]
```

Power Budgeting and Behavior

As we alluded to in the System power section the USBExt3c has to be a bit clever about accounting for all input and output power. The method by which an input power source is selected and used is referred to as Input Power Behavior. It can be configured through

```
stem.system.setInputPowerBehavior() [cpp] [python] [NET] [LabVIEW] [REST]
stem.system.getInputPowerBehavior() [cpp] [python] [NET] [LabVIEW] [REST]
```

Some behaviors require additional configuration. In most cases this is a list of port numbers that define which ports should be prioritized for input power.

```
stem.system.setInputPowerBehaviorConfig() [cpp] [python] [NET] [LabVIEW] [REST]
stem.system.getInputPowerBehaviorConfig() [cpp] [python] [NET] [LabVIEW] [REST]
```

Temperature

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Certain modules have a temperature measurement available. The temperature entity gives access to these measurements. Check your module datasheet to see if your module has a temperature entity.

System Temperature

The temperature of the USBExt3c can be measured with:

```
stem.temperature[0].getValue(μC) [cpp] [python] [.NET] [LabVIEW] [REST]

stem.temperature[0].getValueMin(μC) [cpp] [python] [.NET] [LabVIEW] [REST]

stem.temperature[0].getValueMax(μC) [cpp] [python] [.NET] [LabVIEW] [REST]
```

where temperature is in micro-degrees Celsius.

The USBExt3c exposes seven temperature sensors (indices 0 through 6):

Index	Sensor
0	Channel 0 buck-boost converter
1	Channel 1 buck-boost converter
2	Control buck-boost converter
3	PoE PD stage
4	Ambient
5	PoE PSE controller
6	HDBaseT extension chip

UART

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The UART entity is a class which allows a user to configure and control an arbitrary stream of serial data. These streams can represent a number of different transports, including a RS232 external interface, a virtual COM port, and onboard UART interfaces from system components. This entity allows each transport to be configured as either an endpoint, or as a passthrough between transports, similar to a switchboard of a telephone operator.

UART Control

By default, USBExt3c extends RS-232 connections on the expansion connector. However, USBExt3c also has the ability to be controlled via RS-232. The USBExt3c also supports VCOM (Virtual COM port) channels for USB-based serial communication, with three VCOM channels available. For additional information about the serial protocol, reference *USBExt3c Serial Communication Feature*

UART Entity Indices

The USBExt3c provides the following UART entity indices:

Table 29: UART Entity Indices

Index	Description
0	Hardware RS232 (External Expansion Connector)
3	Internal Extender Channel (TA)
4	VCOM_0 (Virtual COM port channel)
5	VCOM_1 (Virtual COM port channel)
6	VCOM_2 (Virtual COM port channel)

Note: Indices 1-2 are reserved for internal use. User applications typically use indices 0, 3, 4, 5, and 6.

UART Protocols

USBExt3c has four protocol values enumerated.

Value	Description
0	Disabled/Undefined
1	Extron Compatible Protocol
2	Brainstem Transport
6	Loopback

UART APIs

UARTs are controlled through the *following APIs*:

```
stem.uart[x].setEnabled() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getEnable() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setBaudRate() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getBaudRate() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setProtocol() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getProtocol() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setLinkChannel() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getLinkChannel() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setStopBits() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getStopBits() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setParity() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getParity() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setDataBits() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getDataBits() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].setFlowControl() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getFlowControl() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getCapableProtocols() [cpp] [python] [NET] [LabVIEW] [REST]
stem.uart[x].getAvailableProtocols() [cpp] [python] [NET] [LabVIEW] [REST]
```

USB System

Upstream Control

The USBExt3c has the unique ability to designate either of its full featured hub ports (0,1) or the HDBaseT-USB3 link as the upstream connection. This is very useful for moving devices between hosts or testing dual-role port functionality.

```
stem.hub.setUpstream() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.getUpstream() [cpp] [python] [NET] [LabVIEW] [REST]
```

Enumeration Delay

Once a USB device is detected by the USBExt3c it is possible to delay its connection to an upstream host computer and subsequent enumeration on the USB bus. The enumeration delay can mitigate or eliminate host kernel instabilities by forcing devices to enumerate in slow succession, allowing a focus on validation of drivers and software. The enumeration delay is configured in milliseconds, representing the time delay between enabling each successive port. Enumeration delay is applied when the hub powers on or when a new upstream connection is made.

```
stem.hub.setEnumerationDelay() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.getEnumerationDelay() [cpp] [python] [NET] [LabVIEW] [REST]
```

Power Behavior

Ports 0,1, and control of the USBExt3c are each capable of sourcing 60 watts based on available system power. The method in which power is allocated across these ports is called the power behavior and it can be configured through

```
stem.hub.setPowerBehavior() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.getPowerBehavior() [cpp] [python] [NET] [LabVIEW] [REST]
```

Some behaviors require additional configuration. In most cases this is a list of port numbers that define which ports should be prioritized for power allocation.

```
stem.hub.setPowerBehaviorConfig() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.getPowerBehaviorConfig() [cpp] [python] [NET] [LabVIEW] [REST]
```

Data Behavior

Many devices are now capable of being Dual-Role Ports (DRP) meaning that they can be both a device (downstream) and a host (upstream). These devices can request to become a host at anytime which may or may not contradict the users desired upstream setting. The method in which these events are handled is referred to as data behavior. Just as power behavior it can be configured with a similar set of APIs

Table 30: List of Available Data Behaviors for USBExt3c

Behavior	Value	Define
Hard Coded	0	usbsystemDataBehavior_HardCoded
Reserved	1	usbsystemDataBehavior_Reserved
Port Priority	2	usbsystemDataBehavior_PortPriority

Hard Coded

The Hard-Coded data behavior is used to fix the Upstream port to a single port and not allow it to move except for a command through the [Set Upstream](#) API or via the *Serial Communication Feature*.

Newest Connection (default)

The newest connection data behavior assigns the Upstream port role to the port with the most recently connected host-capable device. This means a USB-C connection that's a sink or a USB PD connection that has described itself as USB Coms Capable and can act as a Host.

Relevant API's

```
stem.hub.setDataRoleBehavior() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.getDataRoleBehavior() [cpp] [python] [NET] [LabVIEW] [REST]
```

Some behaviors require additional configuration. In most cases this is a list of port numbers that define which ports should be prioritised for power allocation.

```
stem.hub.setDataRoleBehaviorConfig() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.getDataRoleBehaviorConfig() [cpp] [python] [NET] [LabVIEW] [REST]
```

Override

The USB system by default follows compliant behavior around Vbus toggling and upstream detection. In some test applications you may want to override specific behaviors which is why we have exposed override bits. Below are the get/set routines for overrides and the bit definitions.

```
stem.hub.getOverride() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.setOverride() [cpp] [python] [NET] [LabVIEW] [REST]
```

Name	Bit	Definition
Auto Vbus Toggle Disable	0	Disables the auto vbus toggle behavior on re-enumeration of the upstream port
Vbus Detect Disable	1	Disables the requirement of an upstream connection for enabling the hub chip

High-Level Control of the Port Entity

The USBSystem Entity and the *Port Entity* are capable of doing many of the same things its merely their perspective. The PortClass acts on individual elements where as the USBSystemClass acts on all of the ports. For instance if you wanted to enable all of the ports of the USBExt3c you would need to loop through each index and individually enable each port. With the USBSystem class you can do the exact same thing, but with a single API call with each bit representing a given port.

```
//USBSystem Entity method.
stem.hub.setEnabledList(0x03); //0b0000 0011 bits 0-1 set high = ports 0-1

//Port Entity method.
for(int x = 0; x <= 1; x++) {
    stem.hub.port[x].setEnabled(true);
}
```

```
stem.hub.setEnabledList() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.setEnabledList() [cpp] [python] [NET] [LabVIEW] [REST]
```

The same logic can also be applied to Data Role, Mode, and State elements, but with slightly different interfaces depending on the size of the data.

```
stem.hub.setModeList() [cpp] [python] [NET] [LabVIEW] [REST]
stem.hub.getModeList() [cpp] [python] [NET] [LabVIEW] [REST]
```

Data Role and State are slightly different in that there are only get calls for these functions:

```
stem.hub.getDataRoleList() [cpp] [python] [NET] [LabVIEW] [REST]

stem.hub.getStateList() [cpp] [python] [NET] [LabVIEW] [REST]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
digital[0-1]	getConfiguration	
	getLinkChannel	
	getState	
	getValue	
	setConfiguration	
	setLinkChannel	
ethernet[0]	setValue	
	getIPv4Address	
	getIPv4DNSAddress	
	getIPv4Gateway	

continues on next page

Table 31 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
hdbaset[0-1]	getIPv4Netmask	
	getInterfacePort	
	getMACAddress	
	getNetworkConfiguration	
	getStaticIPv4Address	
	getStaticIPv4Gateway	
	getStaticIPv4Netmask	
	setHostname	
	setInterfacePort	
	setNetworkConfiguration	
	setStaticIPv4DNSAddress	
	getCableLength	
	getEncodingState	
	getFirmwareVersion	
	getLinkRole	
	getLinkUtilization	
	getMSEA	
	getMSEB	
	getRetransmissionRate	
	getSerialNumber	
poe[0]	getState	
	getUSB2DeviceTree	
	getUSB3DeviceTree	
	setLinkRole	
	getPairAccumulatedPower	
	getPairCapacitance	
	getPairCurrent	
	getPairDetectionStatus	
	getPairDiscoveredClass	
	getPairEnabled	
	getPairPower	
	getPairRequestedClass	
	getPairResistance	
	getPairSourcingClass	
	getPairVoltage	
	getPowerMode	
system[0]	getPowerState	
	getTotalAccumulatedPower	
	getTotalPower	
	setPairEnabled	
	setPairSourcingClass	
	setPowerMode	
	getModule	
	getModuleBaseAddress	
	setRouter	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	

continues on next page

Table 31 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getVersion	
	getModel	
	getHardwareVersion	
	getBuild	
	getLinkInterface	
	getSerialNumber	
	save	
	reset	
	logEvents	
	getUptime	
	getTemperature	
	getMinimumTemperature	
	getMaximumTemperature	
	getInputPowerSource	
	getInputVoltage	
	getInputCurrent	
	getModuleHardwareOffset	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	routeToMe	
	getUnregulatedVoltage	
	getUnregulatedCurrent	
	getPowerLimit	
	getPowerLimitMax	
	setPowerLimitMax	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
temperature[0-6]	getValue	
	getValueMax	
	getValueMin	
	resetLoggedValues	
port[0-3]	getVbusVoltage	
	getVbusCurrent	
	getVconnVoltage	
	getVconnCurrent	
	getPowerEnabled	
	setPowerEnabled	
	getPowerMode	
	setPowerMode	
	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getDataHS1Enabled	
	setDataHS1Enabled	
	getDataHS2Enabled	
	setDataHS2Enabled	

continues on next page

Table 31 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getDataSSEnabled	
	setDataSSEnabled	
	getDataSS1Enabled	
	setDataSS1Enabled	
	getDataSS2Enabled	
	setDataSS2Enabled	
	getVconnEnabled	
	setVconnEnabled	
	getVconn1Enabled	
	setVconn1Enabled	
	getVconn2Enabled	
	setVconn2Enabled	
	getDataRole	
	getDataSpeed	
	getCCEnabled	
	setCCEnabled	
	getCC1Enabled	
	setCC1Enabled	
	getCC2Enabled	
	setCC2Enabled	
	getCCBias	
	setCCBias	
	getMode	
	setMode	
	getState	
	getName	
	setName	
	getCurrentLimit	
	setCurrentLimit	
	getAllocatedPower	
	getAvailablePower	
	getPowerLimit	
	setPowerLimit	
	getVbusAccumulatedPower	
	resetVbusAccumulatedPower	
	getVconnAccumulatedPower	
	resetVconnAccumulatedPower	
	getHSBoost	
	setHSBoost	
	getDataHSRoutingBehavior	
	setDataHSRoutingBehavior	
	getDataSSRoutingBehavior	
	setDataSSRoutingBehavior	
	getCC1State	
	getCC2State	
	getCCCurentLimit	
	setCCCurentLimit	
	getVoltageSetpoint	
	setVoltageSetpoint	
powerdelivery[0-2]	getConnectionState	

continues on next page

Table 31 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getNumberOfPowerDataObjects	
	setPowerDataObject	
	getPowerDataObject	
	resetPowerDataObjectToDefault	
	getPowerDataObjectList	
	setPowerDataObjectEnabled	
	getPowerDataObjectEnabled	
	getPowerDataObjectEnabledList	
	setRequestDataObject	
	getRequestDataObject	
	getPowerRole	
	setPowerRole	
	getPowerRolePreferred	
	setPowerRolePreferred	
	getCableVoltageMax	
	getCableCurrentMax	
	getCableSpeedMax	
	getCableType	
	getCableOrientation	
	setOverrides	
	getOverrides	
	request	
	setCurrentLimitBehavior	
	getCurrentLimitBehavior	
	getPeakCurrentConfiguration	
	setPeakCurrentConfiguration	
	getFastRoleSwapCurrent	
	setFastRoleSwapCurrent	
	getAttachTimeElapsed	
	getDataRoleCapabilities	
	getFlagMode	
	getLinkState	
	getPowerRoleCapabilities	
	getProtocolRevisionMax	
	setEnabled	
	setProtocolRevisionMax	
	vDM	
	resetEntityToFactoryDefaults	
usbsystem[0]	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
	getDataHSMaxDatarate	
	getDataSSMaxDatarate	
	getOverride	
	getSelectorMode	
	getUpstreamHS	
	getUpstreamSS	
	setDataHSMaxDatarate	
	setDataSSMaxDatarate	

continues on next page

Table 31 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
UART[0-6]	setOverride	
	setSelectorMode	
	setUpstreamHS	
	setUpstreamSS	
	setEnabled	
	getEnable	
	setBaudRate	
	getBaudRate	
	setProtocol	
	getProtocol	
	setLinkChannel	
	getLinkChannel	
	setStopBits	
	getStopBits	
	setParity	
	getParity	
	setDataBits	
	getDataBits	
	setFlowControl	
	getFlowControl	
	getCapableProtocols	
	getAvailableProtocols	
usb[0]	getCC1Enable	Legacy USB entity; channel sub-indices 0-3
	getCC2Enable	
	getPortCurrent	
	getPortCurrentLimit	
	getPortError	
	getPortMode	
	getPortState	
	getPortVoltage	
	setCC1Enable	
	setCC2Enable	
	setDataDisable	
	setDataEnable	
	setHiSpeedDataDisable	
	setHiSpeedDataEnable	
	setSuperSpeedDataDisable	
	setSuperSpeedDataEnable	
	setPortCurrentLimit	
	setPortDisable	
	setPortEnable	
	setPortMode	
	setPowerDisable	
	setPowerEnable	

1.7 MTM Products

Manufacturing Test Module (MTM) Series instrumentation from Acroname is the platform you need to free your production testers from the burdens of validation test equipment. Mix and match modules to create a complete tester within a fixture frame, eliminating benchtop and rack equipment - all without sacrificing the robustness and reusability you demand from your equipment. And when you are ready to scale your production line, MTM enables rapid replication of inexpensive testers that are repeatable. So mass-production testers behave the same way as the ramp station.

1.7.1 MTM-Relay



As part of Acroname's MTM series, the MTM-Relay module is a key component to automated manufacturing test systems requiring switching of industrial control voltages. The MTM-Relay module features four software-controlled solid-state relays (SSR). Each relay can handle up to 60V and 6ADC/RMS. The powerful BrainStem API allows simple networking of multiple modules into one system, and can report each channel's voltage.

The module also provides four digital (3.3V) GPIO to support module integration and provide additional capability, as necessary.

To get up to speed with the MTM Relay and quickly learn about its functionality follow the [quick start guide](#).

Have a look at the [basic example](#) or dive into the [Programming interface](#) of the MTM Relay for a more in depth view.

Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)¹⁸ for your particular operating system and architecture.
- Download [HubTool](#)¹⁹ for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM Relay by either connecting to the:
 - Onboard USB connection
 - Card edge USB input
 - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM Relay device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the MTM Relay. For more information please take a look at our [Getting Started Guide](#)

Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a MTM-RELAY module.
    aMTMRelay mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
    }
```

(continues on next page)

¹⁸ <https://acroname.com/api>

¹⁹ <https://acroname.com/hubtool>

(continued from previous page)

```

    return 1;

} else { printf("Connected to BrainStem module.\n"); }

//Basic initialization (Get LEDs turned off).
mtm.system.setLED(0);

//Ready for testing
//Enable LED
mtm.system.setLED(1);

//Turn LED off
mtm.system.setLED(0);

//Disconnect
mtm.disconnect();
}

```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-RELAY module.
mtm = brainstem.stem.MTMRelay();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

Indicators and Connections

LEDs

The MTM Relay board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below.

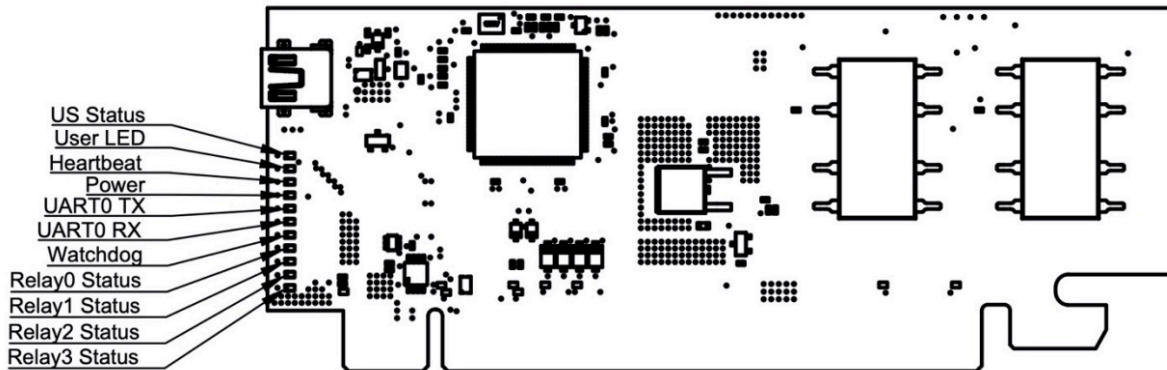


Figure 3: MTM-Relay LED Indicators

Programming Interface

The MTM-Relay is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-Relay.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-Relay. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-Relay can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-Relay is attached to a host machine, a user can connect to it via software API:

```
stem.discoverAndConnect(linkType, serialNumber)
```

The MTM-Relay can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I2C. Each MTM-Relay is uniquely addressable via hardware or software to avoid communication conflicts on the I2C bus.

Upstream USB Connectivity Options

The MTM-Relay supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the miniB connector if 5V is present on Vbus at the mini-B connector.

MTM-RELAY Module Entities

Digital

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

Set/Get Configurations

Gets or Sets the digital pin configuration. Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware. Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

```
stem.digital[index].setConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The mode parameter is an integer that correlates to the following:

Value	Configuration
0	Input
1	Output
4	Hi Impedance (Hi-Z)
5	Input with Pull Down

Set/Get State

Gets or Sets the digital I/O Value. For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet. For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

```
stem.digital[index].setState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

If a digital pin is configured in Hi-Z mode its internal circuitry has been disconnected to create a high impedance. There are no functions that can act on this configuration

Digital	Input	Output	HighZ	RCServo	Signal
DIO0	Yes	Yes	Yes	None	None
DIO 1	Yes	Yes	Yes	None	None
DIO 2	Yes	Yes	Yes	None	None
DIO 3	Yes	Yes	Yes	None	None

I2C

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read/Write Data

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's. The MTM-RELAY includes access to a single I2C bus operating at a set 1Mbit/s rate.

Note: *The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.*

```
stem.i2c[index].read(address, length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

```
stem.i2c[index].write(address, length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

The maximum data size for individual read and write operations on an I2C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Pullup

Each I2C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-RELAY in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I2C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I2C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c[index].setPullup(bEnable) \[cpp\] \[python\] \[NET\] \[CCA\]
```


Relay

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Relay entity is a simple class which allows the enabling and disabling of a specified relay.

Get/Set Enable

The MTM-Relay has four (4) optically isolated solid-state relays controlled by the relay entity. Each relay is controllable via software and capable of 60V and 6A continuous current load.

Enables the relay channel for the specified index.

```
stem.relay[index].setEnabled() \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

```
stem.relay[index].getEnabled() \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

Get Voltage

Returns the voltage of the specified index.

```
stem.relay[index].getVoltage() \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

Store

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Every BrainStem module includes several Store entities and onboard memory slots.

The MTM-RELAY has store slots [0-1].

Store Slot	Storage Type
0	RAM
1	Internal

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

```
stem.store[index].getSlotState(slot) [cpp] [python] [NET] [CCA] [REST]
```

Load/Unload Slot

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

```
stem.store[index].loadSlot(slot, data, _=None) [cpp] [python] [NET] [CCA]
```

This command reads the slot in the given store into the byte buffer. The length will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

```
stem.store[index].unloadSlot(slot) [cpp] [python] [NET] [CCA]
```

Enable/Disable Slot

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotEnable(slot) [cpp] [python] [NET] [CCA]
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotDisable(slot) [cpp] [python] [NET] [CCA]
```

Slot Capacity

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

```
stem.store[index].slotCapacity(slot) [cpp] [python] [NET] [CCA]
```

Slot Size

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

```
stem.store[index].slotSize(slot) [cpp] [python] [NET] [CCA]
```

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every MTM-RELAY is assigned a unique serial number at the factory. This facilitates an arbitrary number of MTM-RELAY devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for MTM-RELAY is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-RELAY away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Reset

Reset the system.

```
stem.system.reset() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set LED

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

```
stem.system.getLED(value) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Boot Slot

Get the store slot which is mapped when the module boots. Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

```
stem.system.getBootSlot() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Input Voltage

Get the module's input voltage.

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Get Version

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

```
stem.system.getVersion() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Module Software Offset

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on the BrainStem network. You must save and reset the module for this change to become effective.

```
stem.system.getModuleSoftwareOffset() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set HB Interval

Get the delay between heartbeat packets. Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Increments valid values are 1-255; default is 10 (256 milliseconds).

```
stem.system.getHBInterval() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Router

Get the router address the module uses to communicate with the host. Set the router address the module uses to communicate with the host.

```
stem.system.getRouter() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Router Address Setting

Get the router address setting saved in the module. This setting may be different from the effective router if the router has been set and saved but no reset has been made.

```
stem.system.getRouterAddressSetting() [cpp] [python] [NET] [CCA] [REST]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [CCA] [REST]
```

Get Model

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

```
stem.system.getModel() [cpp] [python] [NET] [CCA] [REST]
```

Route to Me

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

```
stem.system.routeToMe(value) [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

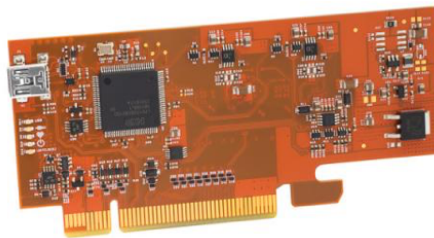
Entity Class	Entity Option	Variable(s) Notes
digital[0-3]	setConfiguration	
	getConfiguration	
	setState	
	getState	
i2c[0]	write	
	read	
relay[0-3]	setEnabled	
	getEnable	

continues on next page

Table 32 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
store[0-1]	getVoltage	
	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
system[0]	slotSize	
	save	
	reset	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

1.7.2 MTM-DAQ-2



The MTM-DAQ-2 module, part of Acroname's Manufacturing Test Module (MTM) instrumentation series, is a modular, software-controlled analog data acquisition (DAQ) module, designed for precision measurements of analog voltages in manufacturing or R&D test.

The MTM-DAQ-2 has 14 channels of differential bi-polar analog inputs with individually adjustable ranges.

Precision voltage measurements can be made through the powerful and cross-platform BrainStem API.

The MTM-DAQ-2 is optimized specifically for precision analog measurement of voltages for sensor and current-sense (through shunt resistors) applications in high-throughput manufacturing test environments.

To get up to speed with the MTM DAQ-2 and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [Programming interface](#) of the MTM Power Module for a more in depth view.

Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)²⁰ for your particular operating system and architecture.
- Download [HubTool](#)²¹ for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM DAQ-2 by either connecting to the:
 - Onboard USB connection
 - Card edge USB input
 - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM DAQ-2 device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the MTM Relay. For more information please take a look at our [Getting Started Guide](#)

Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a MTM-DAQ-2 module.
    aMTMDAQ2 mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
```

(continues on next page)

²⁰ <https://acroname.com/api>

²¹ <https://acroname.com/hubtool>

(continued from previous page)

```

    printf("Error %d encountered connecting to BrainStem module\n", err);
    return 1;

} else { printf("Connected to BrainStem module.\n"); }

//Basic initialization (Get LEDs turned off).
mtm.system.setLED(0);

//Ready for testing
//Enable LED
mtm.system.setLED(1);

//Turn LED off
mtm.system.setLED(0);

//Disconnect
mtm.disconnect();
}

```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-DAQ-2 module.
mtm = brainstem.stem.MTMDAQ2();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

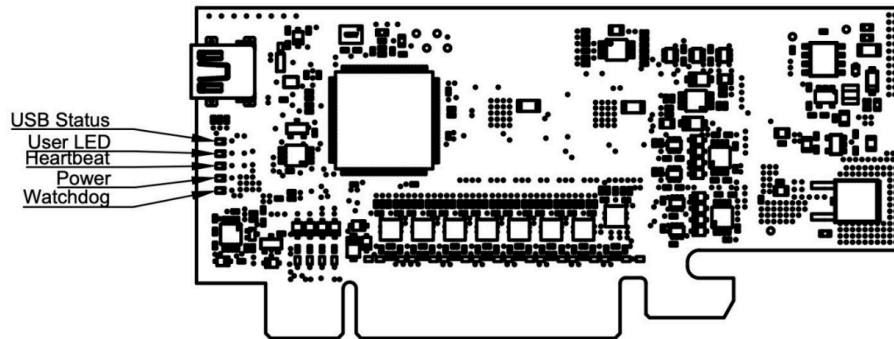
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

Indicators and Connections

LEDs

The MTM-DAQ-2 board has five LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below



Programming Interface

The MTM-DAQ-2 is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-DAQ-2.

A complete list of all entities and functions can be found in the *Module Entities* page.

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-DAQ-2. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-DAQ-2 can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-DAQ-2 is attached to a host machine, a user can connect to it via software API:

```
stem.discoverAndConnect(linkType, serialNumber)
```

The MTM-DAQ-2 can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I2C. Each MTM-DAQ-2 is uniquely addressable via hardware or software to avoid communication conflicts on the I2C bus.

Upstream USB Connectivity Options

The MTM-DAQ-2 supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the miniB connector if 5V is present on Vbus at the mini-B connector.

MTM-DAQ-2 Module Entities

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every MTM-DAQ-2 is assigned a unique serial number at the factory. This facilitates an arbitrary number of MTM-DAQ-2 devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for MTM-DAQ-2 is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-DAQ-2 away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Reset

Reset the system.

```
stem.system.reset() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set LED

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

```
stem.system.getLED(value) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Boot Slot

Get the store slot which is mapped when the module boots. Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

```
stem.system.getBootSlot() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Input Voltage

Get the module's input voltage.

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Get Version

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

```
stem.system.getVersion() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Module Software Offset

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on the BrainStem network. You must save and reset the module for this change to become effective.

```
stem.system.getModuleSoftwareOffset() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set HB Interval

Get the delay between heartbeat packets. Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Increments valid values are 1-255; default is 10 (256 milliseconds).

```
stem.system.getHBInterval() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Router

Get the router address the module uses to communicate with the host. Set the router address the module uses to communicate with the host.

```
stem.system.getRouter() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Router Address Setting

Get the router address setting saved in the module. This setting may be different from the effective router if the router has been set and saved but no reset has been made.

```
stem.system.getRouterAddressSetting() [cpp] [python] [NET] [CCA] [REST]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [CCA] [REST]
```

Get Model

Get the module's model enumeration.

A subset of the possible model enumerations is defined in `aProtocolDefs.h` under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

```
stem.system.getModel() [cpp] [python] [NET] [CCA] [REST]
```

Route to Me

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

```
stem.system.routeToMe(value) [cpp] [python] [NET] [CCA] [REST]
```

Store

API Documentation: [cpp] [python] [NET] [CCA] [REST]

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the `:doc:`System <system>`` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Every BrainStem module includes several Store entities and onboard memory slots.

The MTM-DAQ-2 has store slots [0-1].

Store Index	Storage Type	Number of Slots
0	Internal	12
1	RAM	1

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

```
stem.store[index].getSlotState(slot) [cpp] [python] [NET] [CCA] [REST]
```

Load/Unload Slot

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

```
stem.store[index].loadSlot(slot, data, _=None) [cpp] [python] [NET] [CCA]
```

This command reads the slot in the given store into the byte buffer. The length will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

```
stem.store[index].unloadSlot(slot) [cpp] [python] [NET] [CCA]
```

Enable/Disable Slot

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotEnable(slot) [cpp] [python] [NET] [CCA]
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotDisable(slot) [cpp] [python] [NET] [CCA]
```

Slot Capacity

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

```
stem.store[index].slotCapacity(slot) [cpp] [python] [NET] [CCA]
```

Slot Size

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

```
stem.store[index].slotSize(slot) [cpp] [python] [NET] [CCA]
```

Digital

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

Set/Get Configurations

Gets or Sets the digital pin configuration. Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware. Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

```
stem.digital[index].setConfiguration(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.digital[index].getConfiguration(mode) [cpp] [python] [NET] [CCA] [REST]
```

The mode parameter is an integer that correlates to the following:

Digital	Input	Output	HighZ	RCServo	Signal
DIO 0	Yes	Yes	Yes	.	.
DIO 1	Yes	Yes	Yes	.	.
DIO 2	Yes	Yes	Yes	.	.
DIO 3	Yes	Yes	Yes	.	.

Set/Get State

Gets or Sets the digital I/O Value. For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet. For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

```
stem.digital[index].setState(level) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.digital[index].getState(level) [cpp] [python] [NET] [CCA] [REST]
```


If a digital pin is configured in Hi-Z mode its internal circuitry has been disconnected to create a high impedance. There are no functions that can act on this configuration

Value	Configuration
0	Input with Pullup
1	Output
4	Hi Impedance (Hi-Z)
5	Input with Pull Down

Analog

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read an analog voltage (ADC) and convert this into a discrete digitized value or output a voltage value based on a desired discrete value (DAC). Analog voltage capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of analog entities and details for their capacities will be described in that module's datasheet.

Set/Get Range

The MTM-DAQ-2 has sixteen (16) analog inputs (ADC) and two (2) analog outputs (DAC) all controlled by the analog entity. Each analog is controllable via software and is independently current-limited for both source and sink currents. The analog inputs are connected to a 16-bit ADC, and return a voltage value in microvolts. The full ranges are in the table below.

Analog	Input	Input	Input	Input	Input	Output	Output	Output
0-13	(+/-) 10.24V	(+/-) 5.12V	(+/-) 2.56V	(+/-) 1.28V	(+/-) 0.64V	.	.	.
14-15	.	.	.	(+/-) 1.28V	(+/-) 0.64V	.	.	.
16-17	(+/-) 10.24V	(+/-) 4.096V	(+/-) 2.048V

```
stem.analog[index].setRange(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.analog[index].getRange(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Get/Set Enable [16, 17]

These outputs default to having their outputs disabled, so `setEnabled(1)` must be called before their voltage will be present on their respective pins.

```
stem.analog[index].setEnabled(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].setEnabled(mode) [cpp] [python] [NET] [CCA] [REST]
```

Get Voltage/Value

A BrainStem's A2D reading will always return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the measurement will get propagated up to 16 bits wide.

For example, if a 12-bit A2D engine exists in the target module's hardware, the reading will get promoted in the firmware layer by shifting up 4 bits to fill out the 16 bit value ($0x0FFF \ll 4 = 0xFFF0$) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

`setValue` and `setVoltage` is only applicable for Analog[16, 17]:

```
stem.analog[index].setValue() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].getValue() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].setVoltage(microvolts) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].getVoltage() [cpp] [python] [NET] [CCA] [REST]
```

I2C

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read/Write Data

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's. The MTM-DAQ-2 includes access to a single I2C bus operating at a set 1Mbit/s rate.

Note: *The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.*

```
stem.i2c[index].read(address,length) [cpp] [python] [NET] [CCA]
```

```
stem.i2c[index].write(address,length) [cpp] [python] [NET] [CCA]
```

The maximum data size for individual read and write operations on an I2C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Pullup

Each I2C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-DAQ-2 in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I2C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I2C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c[index].setPullup(bEnable) [cpp] [python] [NET] [CCA]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
digital[0-1]	setConfiguration	
	getConfiguration	
	setState	
	getState	
i2c[0]	write	
	read	
analog[0-15]	getVoltage	
	getValue	
	setRange	
	getRange	
analog[16-17]	setVoltage	
	getVoltage	
	setValue	
	getValue	
	setRange	
	getRange	
	setEnabled	
	getEnable	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
system[0]	slotCapacity	
	slotSize	
	save	
	reset	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	

continues on next page

Table 33 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

1.7.3 MTM-PM-1



The MTM-PM-1, part of Acroname's Manufacturing Test Module (MTM) system, is a modular power supply designed for powering devices during manufacturing or R&D testing. The MTM-PM-1 is a one-channel software controlled, voltage and current limiting power supply. While it can provide stable, consistent and robust power to a wide range of devices, it is optimized for devices using LiPo or similar batteries; in particular, it excels at powering devices needing stable power under large transient loads, such as cellular radios (GSM, UMTS, LTE, CDMA, etc.). Accurate voltage, temperature and current measurements can be made through the powerful and cross platform BrainStem API.

To get up to speed with the MTM Power Module and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [Programming interface](#) of the MTM Power Module for a more in depth view.

Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)²² for your particular operating system and architecture.
- Download [HubTool](#)²³ for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM Power Module by either connecting to the:
 - Onboard USB connection
 - Card edge USB input
 - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM Power Module device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the MTM Relay. For more information please take a look at our [Getting Started Guide](#)

Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a MTM-PM-1 module.
    aMTMPM1 mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
        return 1;
    } else { printf("Connected to BrainStem module.\n"); }
```

(continues on next page)

²² <https://acroname.com/api>

²³ <https://acroname.com/hubtool>

(continued from previous page)

```

//Basic initialization (Get LEDs turned off).
mtm.system.setLED(0);

//Ready for testing
//Enable LED
mtm.system.setLED(1);

//Turn LED off
mtm.system.setLED(0);

//Disconnect
mtm.disconnect();
}

```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-PM-1 module.
mtm = brainstem.stem.MTMPM1();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

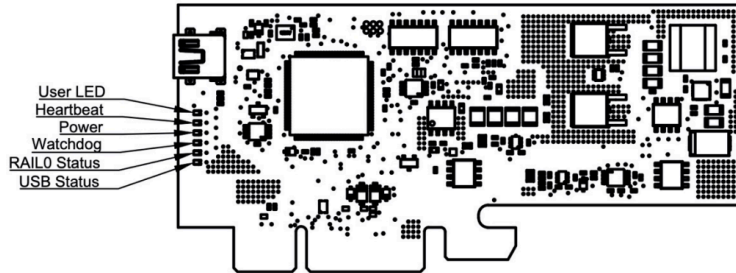
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

Indicators and Connections

LEDs

The MTM-PM-1 board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below.



Programming Interface

The MTM-PM-1 is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-PM-1.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-PM-1. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-PM-1 can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-PM-1 is attached to a host machine, a user can connect to it via software API:

```
stem.discoverAndConnect(linkType, serialNumber)
```

The MTM-PM-1 can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I2C. Each MTM-PM-1 is uniquely addressable via hardware or software to avoid communication conflicts on the I2C bus.

Upstream USB Connectivity Options

The MTM-PM-1 supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the miniB connector if 5V is present on Vbus at the mini-B connector.

MTM-PM-1 Module Entities

Digital

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

Set/Get Configurations

Gets or Sets the digital pin configuration. Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware. Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

```
stem.digital[index].setConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The mode parameter is an integer that correlates to the following:

Value	Configuration
0	Input
1	Output
4	Hi Impedance (Hi-Z)
5	Input with Pull Down

Set/Get State

Gets or Sets the digital I/O Value. For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet. For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

```
stem.digital[index].setState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

If a digital pin is configured in Hi-Z mode its internal circuitry has been disconnected to create a high impedance. There are no functions that can act on this configuration

Digital	Input	Output	HighZ	RCServo	Signal
DIO 0	Yes	Yes	Yes	.	.
DIO 1	Yes	Yes	Yes	.	.

I2C

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read/Write Data

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's. The MTM-PM-1 includes access to a single I2C bus operating at a set 1Mbit/s rate.

Note: *The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.*

```
stem.i2c[index].read(address,length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

The maximum data size for individual read and write operations on an I2C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Pullup

Each I2C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-PM-1 in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I2C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I2C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c[index].setPullup(bEnable) \[cpp\] \[python\] \[NET\] \[CCA\]
```

Rail

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Rail entity provides power control to connected devices on some modules. Check the module datasheet to determine if the module has this capability.

the Rail entity controls power provided to downstream devices, it has the ability to enable and disable power, can read voltage on the rail, and provides current consumption information on some modules. There are additional capabilities that certain modules provide which enhance basic power delivery through Kelvin sensing, or by bringing online separate power management functionality.

Certain modules may provide more than one power rail. These are independently controlled and can be accessed via the entity index.

Rails on the MTM-PM-1 module are powerful (no pun intended); they allow other devices and peripherals to consume power from the MTM-PM-1 module in a precisely controlled fashion. Two (2) different rails are available for use: a software-adjustable voltage rail (rail), and input voltage pass-through rail (rail1). These rails are accessed through an array of BrainStem rail class entities. The MTM-PM-1 module implements a subset of the BrainStem rail class for each of these rails.

Enable

All three rails can be switched on or off through using the API

```
stem.rail[index].setEnabled(state) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Rail Operational Mode

RAIL can be configured to use two different regulation stages: linear (LDO) or switch-mode power supply (SMPS)

```
stem.rail[index].setOperationalMode(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.rail[index].getOperationalMode(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Mode	Operational Mode Description
0	railOperationalModeAuto (default)
1	railOperationalModeLinear
3	railOperationalModeSwitcherLinear

Rail Operational State

Auto configuration chooses the switch-mode power supply if an input voltage greater than 7.25V is applied, and the linear regulator otherwise. The API can be used to read the actual operational state

```
stem.rail[index].getOperationalState() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Bits	Operational State Field Description	Rails
0	Initializing (railOperationalState_Initializing)	rail[0-1]
1	Enabled (railOperationalState_Enabled)	rail[0-1]
2	Fault (railOperationalState_Fault)	rail[0-1]
3-15	Reserved	.
8-15	Hardware Configuration (railOperationalState_HardwareConfiguration)	rail0
16-17	Reserved	.
18	Overcurrent Fault “OC” (railOperationalStateOverCurrentFault)	rail1
19-20	Reserved	.
21	Overtemperature Fault “OT” (railOperationalStateOverTemperatureFault)	rail0
22-31	Reserved	.

Rail Temperature

The printed circuit board (PCB) temperature can be monitored at the 5.0V rail (RAIL) linear regulation stage. Reading this value is possible through the API

```
stem.rail[index].getTemperature() [cpp] [python] [NET] [CCA] [REST]
```

Temperature monitoring is also used internally to prevent the power regulation stage from overheating and self-preserving the power stage. If an overtemperature condition occurs, then the MTM-IO-Serial module will disable the linear regulator until safe operating temperatures are reached.

Rail Voltage Setting

RAIL always uses linear regulators to generate an adjustable voltage. They can be set or read using the API

```
stem.rail[index].setVoltageSetpoint(microvolts) [cpp] [python] [NET] [CCA] [REST]
```

Rail and Rail1 Current Limits

The current limit for each rail can be configured in software from 0A to 3A

```
stem.rail[index].setCurrentLimit(microamps) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.rail[index].getCurrentLimit(microamps) [cpp] [python] [NET] [CCA] [REST]
```

Note that the behavior following an overcurrent event differs between rails:

- rail will simply reduce the output voltage to drive the specified current. No fault bits will be set in software.
- rail1 will be turned off by the hardware if the output current goes above the set limit. The rail1 Fault and Overcurrent Fault bits will be set and must be cleared before re-enabling the rail.

Rail and Rail1 Current and Voltage

The API command to measure what the current and voltages are

```
stem.rail[index].getCurrent(microamps) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.rail[index].getVoltage(microamps) [cpp] [python] [NET] [CCA] [REST]
```

Rail Kelvin Sensing

Remote sensing can be applied to compensate for line loss in a system often found in high transient load applications. The MTM-PM-1 provides a “3-wire” interface to provide feedback to the MTM-PM-1 power supply to adjust appropriately and dynamically

```
stem.rail[index].setKelvinSensingEnable(bEnable) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.rail[index].getKelvinSensingEnable(bEnable) [cpp] [python] [NET] [CCA] [REST]
```

bEnable parameter is an integer that correlates to the following:

- 0: kelvinSensingOff
- 1: kelvinSensingOn

Determine whether kelvin sensing is enabled or disabled. Kelvin sensing can be disabled if the power stage incurs a fault on the rail power stage.

```
stem.rail[index].getKelvinSensingState(state) [cpp] [python] [NET] [CCA] [REST]
```

Store

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Every BrainStem module includes several Store entities and onboard memory slots.

The MTM-PM-1 has store slots [0-1].

Store Slot	Storage Type
0	RAM
1	Internal

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

```
stem.store[index].getSlotState(slot) [cpp] [python] [NET] [CCA] [REST]
```

Load/Unload Slot

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

```
stem.store[index].loadSlot(slot, data, _=None) [cpp] [python] [NET] [CCA]
```

This command reads the slot in the given store into the byte buffer. The length will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

```
stem.store[index].unloadSlot(slot) [cpp] [python] [NET] [CCA]
```

Enable/Disable Slot

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotEnable(slot) [cpp] [python] [NET] [CCA]
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotDisable(slot) [cpp] [python] [NET] [CCA]
```

Slot Capacity

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

```
stem.store[index].slotCapacity(slot) [cpp] [python] [NET] [CCA]
```

Slot Size

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

```
stem.store[index].slotSize(slot) [cpp] [python] [NET] [CCA]
```

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every MTM-PM-1 is assigned a unique serial number at the factory. This facilitates an arbitrary number of MTM-PM-1 devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) [cpp] [python] [NET] [CCA] [REST]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for MTM-PM-1 is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) [cpp] [python] [NET] [CCA] [REST]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-PM-1 away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Reset

Reset the system.

```
stem.system.reset() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set LED

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

```
stem.system.getLED(value) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Boot Slot

Get the store slot which is mapped when the module boots. Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

```
stem.system.getBootSlot() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [CCA] [REST]
```


Get Input Voltage

Get the module's input voltage.

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Get Version

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

```
stem.system.getVersion() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Module Software Offset

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on the BrainStem network. You must save and reset the module for this change to become effective.

```
stem.system.getModuleSoftwareOffset() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set HB Interval

Get the delay between heartbeat packets. Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Increments valid values are 1-255; default is 10 (256 milliseconds).

```
stem.system.getHBInterval() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Router

Get the router address the module uses to communicate with the host. Set the router address the module uses to communicate with the host.

```
stem.system.getRouter() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Router Address Setting

Get the router address setting saved in the module. This setting may be different from the effective router if the router has been set and saved but no reset has been made.

```
stem.system.getRouterAddressSetting() [cpp] [python] [NET] [CCA] [REST]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [CCA] [REST]
```

Get Model

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

```
stem.system.getModel() [cpp] [python] [NET] [CCA] [REST]
```

Route to Me

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

```
stem.system.routeToMe(value) [cpp] [python] [NET] [CCA] [REST]
```

Temperature

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Certain modules have a temperature measurement available. The temperature entity gives access to these measurements. Check your module datasheet to see if your module has a temperature entity.

System Temperature

The temperature of the MTM-PM-1 can be measured with:

```
stem.temperature[0].getValue(μC) \[cpp\] \[python\] \[.NET\] \[CCA\] \[REST\]
```

where temperature is in micro-degrees Celcius.

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
digital[0-1]	setConfiguration	
	getConfiguration	
	setState	
	getState	
i2c[0]	write	
	read	
rail[0-1]	setEnabled	
	setCurrentLimit	
	getCurrent	
	getCurrentLimit	
rail[0]	getVoltage	
	setVoltage	
	setOperationalMode	
	getOperationalMode	
	getOperationalState	
	getTemperature	
	getKelvinSensingEnable	
	setKelvinSensingEnable	
store[0-1]	getKelvinSensingState	
	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
system[0]	slotSize	
	save	

continues on next page

Table 34 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	
temperature[0]	getTemperature	

1.7.4 MTM-Load-1



The MTM-Load-1, part of Acroname's Manufacturing Test Module (MTM) system, is a single-channel software-controlled electronic load for automated functional testing in production or validation test environments.

With its ultra-small footprint, MTM-Load-1 is the load you need for lean, miniaturized production testing. MTM-Load-1 is ideal for load-testing of battery chargers, amplifiers, USB power outputs or motor driver circuits.

Each MTM-Load-1 can dissipate a DC load of 50W continuous power up to 30V or 10A and features constant-current operation. Multiple MTM-Load-1 modules can be used in parallel for higher demand load applications.

To get up to speed with the MTM Load and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [Programming interface](#) of the MTM Power Module for a more in depth view.

Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)²⁴ for your particular operating system and architecture.
- Download [HubTool](#)²⁵ for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM Load by either connecting to the:
 - Onboard USB connection
 - Card edge USB input
 - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM Load device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the MTM Load. For more information please take a look at our [Getting Started Guide](#)

Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a MTM-LOAD module.
    aMTMLoad mtm;
```

(continues on next page)

²⁴ <https://acroname.com/api>

²⁵ <https://acroname.com/hubtool>

(continued from previous page)

```

//Connect to the hardware.
err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if (err != aErrNone) {
    printf("Error %d encountered connecting to BrainStem module\n", err);
    return 1;

} else { printf("Connected to BrainStem module.\n"); }

//Basic initialization (Get LEDs turned off).
mtm.system.setLED(0);

//Ready for testing
//Enable LED
mtm.system.setLED(1);

//Turn LED off
mtm.system.setLED(0);

//Disconnect
mtm.disconnect();
}

```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-LOAD module.
mtm = brainstem.stem.MTMLoad1();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

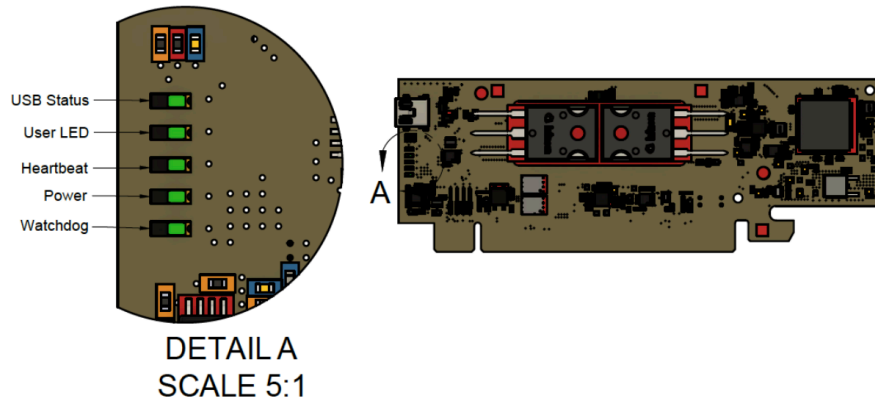
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

Indicators and Connections

LEDs

The MTM-Load-1 board has five LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below.



Programming Interface

The MTM-Load is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-Load.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-Load. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-Load can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-Load is attached to a host machine, a user can connect to it via software API:

```
stem.discoverAndConnect(linkType, serialNumber, modelNumber)
```

The MTM-Load can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I2C. Each MTM-Load is uniquely addressable via hardware or software to avoid communication conflicts on the I2C bus.

Upstream USB Connectivity Options

The MTM-Load supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the miniB connector if 5V is present on Vbus at the mini-B connector.

MTM-LOAD-1 Module Entities

Digital

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

Set/Get Configurations

Gets or Sets the digital pin configuration. Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware. Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

```
stem.digital[index].setConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The mode parameter is an integer that correlates to the following:

Value	Configuration
0	Input
1	Output
4	HiZ

Set/Get State

Gets or Sets the digital I/O Value. For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet. For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

```
stem.digital[index].setState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

If a digital pin is configured in Hi-Z mode its internal circuitry has been disconnected to create a high impedance. There are no functions that can act on this configuration

Digital	Input	Output	Hi-Z	RCServo	Signal
DIO0	Yes	Yes	Yes	.	.
DIO1	Yes	Yes	Yes	.	.
DIO2	Yes	Yes	Yes	.	.
DIO3	Yes	Yes	Yes	.	.

I2C

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read/Write Data

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's. The MTM-LOAD includes access to a single I2C bus operating at a set 1Mbit/s rate.

Note: *The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.*

```
stem.i2c[index].read(address,length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

The maximum data size for individual read and write operations on an I2C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Pullup

Each I2C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-LOAD in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I2C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I2C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c[index].setPullup(bEnable) \[cpp\] \[python\] \[NET\] \[CCA\]
```

Rail

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Rail entity provides power control to connected devices on some modules. Check the module datasheet to determine if the module has this capability.

the Rail entity controls power provided to downstream devices, it has the ability to enable and disable power, can read voltage on the rail, and provides current consumption information on some modules. There are additional capabilities that certain modules provide which enhance basic power delivery through Kelvin sensing, or by bringing online separate power management functionality.

Certain modules may provide more than one power rail. These are independently controlled and can be accessed via the entity index.

Rail 0 on the MTM-Load-1 module is powerful (no pun intended); it allows other devices and peripherals to provide power to the MTM-Load-1 module where it is precisely loaded. The rail is a software-adjustable constant current sink. This rail is accessed through a BrainStem rail class entity. The MTM-Load-1 module implements a subset of the BrainStem rail class for the load rail.

Enable

All three rails can be switched on or off through using the API

```
stem.rail[index].setEnabled(state) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Rail Operational Mode

RAIL can be configured to use two different regulation stages: linear (LDO) or switch-mode power supply (SMPS)

```
stem.rail[index].setOperationalMode(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.rail[index].getOperationalMode(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Value	Define
Hardware Mode - Bits [0-3]	
0	railOperationalModeAuto_Value
1	railOperationalModeLinear_Value
2	railOperationalModeSwitcher_Value
3	railOperationalModeSwitcherLinear_Value
Operational Mode - Bits [4-7]	
0	railOperationalConstantCurrent_Value

Operational State

Auto configuration chooses the switch-mode power supply if an input voltage greater than 7.25V is applied, and the linear regulator otherwise. The API can be used to read the actual operational state

```
stem.rail[index].getOperationalState(state) [cpp] [python] [NET] [CCA] [REST]
```

Bits	RAIL Operational State Description
0	Initializing (railOperationalState_Initializing)
1	Enabled (railOperationalState_Enabled)
2	Fault (railOperationalState_Fault)
3-15	Reserved
8-15	Hardware Configuration (railOperationalState_HardwareConfiguration)
16	Overvoltage Fault "OV" (railOperationalStateOverVoltageFault)
17	Undervoltage Fault "UV" (railOperationalStateUnderVoltageFault)
18	Overcurrent Fault "OC" (railOperationalStateOverCurrentFault)
19	Overpower Fault "OP" (railOperationalStateOverPowerFault)
20	Reverse Polarity Fault "RV" (railOperationalStateReversePolarityFault)
21	Overtemperature Fault "OT" (railOperationalStateOverTemperatureFault)
22-23	Reserved
24-31	Operating Mode (railOperationalStateOperatingMode)

Rail Temperature

The printed circuit board (PCB) temperature can be monitored at the 5.0V rail (RAIL0) linear regulation stage. Reading this value is possible through the API

```
stem.rail[index].getTemperature() [cpp] [python] [NET] [CCA] [REST]
```

Temperature monitoring is also used internally to prevent the power regulation stage from overheating and self-preserving the power stage. If an overtemperature condition occurs, then the MTM-IO-Serial module will disable the linear regulator until safe operating temperatures are reached.

Rail Current Setting

The current setpoint for the rail can be configured in software from 0A to 10A. Setting values outside the allowable range will return an error (aErrRange 13). The rail will attempt to maintain the specified current through all input voltage variations once the rail is enabled with the operational mode set to constant current.

```
stem.rail[index].setCurrentSetpoint(microvolts) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.rail[index].getCurrentSetpoint(microvolts) [cpp] [python] [NET] [CCA] [REST]
```

Rail Current Limit

The current limit for the rail can be configured in software from 0A to 12A. The rail will operate normally if the measured current is below the specified current. If the limit is crossed, the load will automatically disable the rail and set the corresponding overcurrent fault bit in the Operational State variable. If the current limit is below the current setpoint the rail will still disable itself when the current limit is exceeded

```
stem.rail[index].setCurrentLimit (microamps) [cpp] [python] [NET] [CCA] [REST]
```

Rail Voltage Min/Max Setting

The voltage limits for the rail can be configured in software from -0.7V to 35V. The rail will operate normally between the minimum and maximum voltage limits. If the upper or lower limit is crossed, the load will automatically disable the rail and set the corresponding over/under voltage fault bit in the Operational State variable.

```
stem.rail[index].setVoltageMinLimit (microvolts) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.rail[index].setVoltageMaxLimit (microvolts) [cpp] [python] [NET] [CCA] [REST]
```

Rail Power Limit Setting

The power limit for the rail can be configured in software from 0W to 150W. The rail will operate normally below this limit. If the limit is crossed, the load will automatically disable the rail and set the corresponding overpower fault bit in the Operational State variable.

```
stem.rail[index].setPowerLimit (milliwatts) [cpp] [python] [NET] [CCA] [REST]
```

Store

API Documentation: *[cpp] [python] [.NET] [CCA] [REST]*

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Every BrainStem module includes several Store entities and onboard memory slots.

The MTM-LOAD has store slots [0-1].

Store Slot	Storage Type
0	RAM
1	Internal

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

```
stem.store[index].getSlotState(slot) [cpp] [python] [NET] [CCA] [REST]
```

Load/Unload Slot

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

```
stem.store[index].loadSlot(slot, data, _=None) [cpp] [python] [NET] [CCA]
```

This command reads the slot in the given store into the byte buffer. The length will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

```
stem.store[index].unloadSlot(slot) [cpp] [python] [NET] [CCA]
```

Enable/Disable Slot

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotEnable(slot) [cpp] [python] [NET] [CCA]
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotDisable(slot) [cpp] [python] [NET] [CCA]
```

Slot Capacity

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

```
stem.store[index].slotCapacity(slot) [cpp] [python] [NET] [CCA]
```

Slot Size

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

```
stem.store[index].slotSize(slot) [cpp] [python] [NET] [CCA]
```

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every MTM-LOAD is assigned a unique serial number at the factory. This facilitates an arbitrary number of MTM-LOAD devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) [cpp] [python] [NET] [CCA] [REST]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for MTM-LOAD is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) [cpp] [python] [NET] [CCA] [REST]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-LOAD away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Reset

Reset the system.

```
stem.system.reset() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set LED

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

```
stem.system.getLED(value) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Boot Slot

Get the store slot which is mapped when the module boots. Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

```
stem.system.getBootSlot() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Input Voltage

Get the module's input voltage.

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Get Version

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

```
stem.system.getVersion() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Module Software Offset

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on the BrainStem network. You must save and reset the module for this change to become effective.

```
stem.system.getModuleSoftwareOffset() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set HB Interval

Get the delay between heartbeat packets. Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Increments valid values are 1-255; default is 10 (256 milliseconds).

```
stem.system.getHBInterval() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [CCA] [REST]
```


Get/Set Router

Get the router address the module uses to communicate with the host. Set the router address the module uses to communicate with the host.

```
stem.system.getRouter() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Router Address Setting

Get the router address setting saved in the module. This setting may be different from the effective router if the router has been set and saved but no reset has been made.

```
stem.system.getRouterAddressSetting() [cpp] [python] [NET] [CCA] [REST]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [CCA] [REST]
```

Get Model

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

```
stem.system.getModel() [cpp] [python] [NET] [CCA] [REST]
```

Route to Me

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

```
stem.system.routeToMe(value) [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
digital[0-3]	setConfiguration	
	getConfiguration	
	setState	
	getState	
i2c[0]	write	
	read	
rail[0]	setPullup	Disabled by default.
	setCurrent	
	getCurrent	
	getCurrentSetpoint	
	setCurrentLimit	
	getCurrentLimit	
	getTemperature	
	setEnabled	
	getEnable	
	setVoltage	
	getOperationalState	
	getVoltageSetpoint	
	setVoltageMinLimit	
	getVoltageMinLimit	
	setVoltageMaxLimit	
	getVoltageMaxLimit	
	setPower	
	getPower	
	getPowerSetpoint	
	setPowerLimit	
	getPowerLimit	
	setResistance	
	getResistance	
	getResistanceSetpoint	
	setOperationalMode	
	getOperationalMode	
	getOperationalState	
	clearFaults	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
system[0]	slotSize	
	Reset	
	save	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	

continues on next page

Table 35 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	

1.7.5 MTM-IO-Serial



As part of Acroname's MTM series, the MTM-IO-Serial module is a key component to manufacturing test systems for electronic devices using a standard USB 2.0 interface, serial UARTs and one or more interface voltages. The MTM-IO-Serial module features a software controlled USB hub (USB 2.0 high-speed) with four controllable channels. Each channel has switched data and 500mA current-limited power lines. With dedicated USB downstream and upstream channels, MTM-IO-Serial modules can scale with simple PCB daisy-chaining; only one cable needed to connect up to 100 devices.

The module also provides four high speed serial UART interfaces which require no specialized driver or kernel extensions.

To get up to speed with the MTM IO-Serial and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [Programming interface](#) of the MTM Power Module for a more in depth view.

Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)²⁶ for your particular operating system and architecture.
- Download [HubTool](#)²⁷ for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM IO Serial by either connecting to the:
 - Onboard USB connection
 - Card edge USB input
 - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM IO Serial device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the MTM IO Serial. For more information please take a look at our [Getting Started Guide](#)

Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a MTM-IO-SERIAL module.
    aMTMIOSerial mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
        return 1;
    } else { printf("Connected to BrainStem module.\n"); }
```

(continues on next page)

²⁶ <https://acroname.com/api>

²⁷ <https://acroname.com/hubtool>

(continued from previous page)

```

//Basic initialization (Get LEDs turned off).
mtm.system.setLED(0);

//Ready for testing
//Enable LED
mtm.system.setLED(1);

//Turn LED off
mtm.system.setLED(0);

//Disconnect
mtm.disconnect();
}

```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-IO-SERIAL module.
mtm = brainstem.stem.MTMIOSerial();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

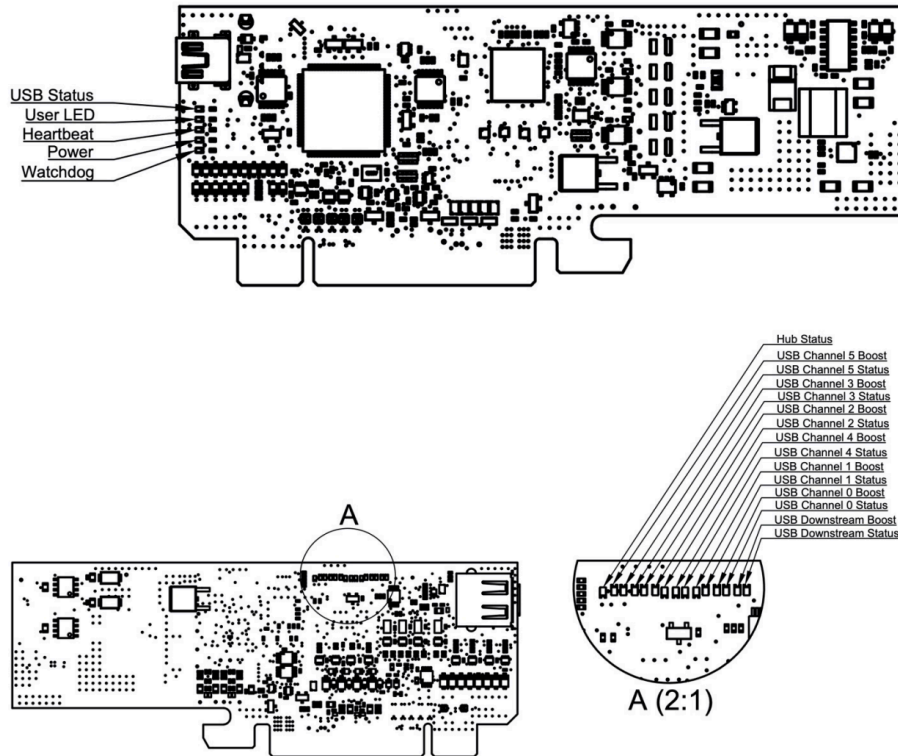
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

Indicators and Connections

LEDs

The MTM-IO-Serial board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below.



Programming Interface

The MTM-IO-Serial is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-IO-Serial.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-IO-Serial. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-IO-Serial can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-IO-Serial is attached to a host machine, a user can connect to it via software API:

```
stem.discoverAndConnect(linkType, serialNumber)
```

The MTM-IO-Serial can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I2C. Each MTM-IO-Serial is uniquely addressable via hardware or software to avoid communication conflicts on the I2C bus.

Upstream USB Connectivity Options

The MTM-IO-Serial supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the miniB connector if 5V is present on Vbus at the mini-B connector.

MTM-IO-SERIAL Module Entities

Digital

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

Set/Get Configurations

Gets or Sets the digital pin configuration. Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware. Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

```
stem.digital[index].setConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The mode parameter is an integer that correlates to the following:

Value	Configuration
0	Input
1	Output
2	RC Servo Input
3	RC Servo Output
6	Signal Output
7	Signal Input

Set/Get State

Gets or Sets the digital I/O Value. For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet. For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

```
stem.digital[index].setState(level) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.digital[index].getState(level) [cpp] [python] [NET] [CCA] [REST]
```

RCServo

API Documentation: *[cpp] [python] [.NET] [CCA] [REST]*

The RCServo entity provides a pulsed signal based on the RC servo standard. This consist of a period lasting 20ms with a high pulse between 1-2ms. The time high corresponds to a specific position determined by the servo being used. For example if you are using a 90 degree servo a 1.5ms pulse will correspond to the 45 degrees. 1ms and 2ms pulses will correspond to 0 and 90 degree positions respectively.

The RCServo entity is an overload to the [Digital Entity](#) and therefor requires proper configuration of the Digital entity before the RCServo entity can be enabled.

The MTM-IO-SERIAL board is equipped with 4 RC servo inputs and 4 RC servo outputs. The RC Servo entity is an overload of the Digital Entity and thus requires proper configuration before this entity can be enabled.

With the RC servo entity, digital output pins generate pulsed signal based on the RC Servo standard consisting of a period lasting 20ms and high pulse time between 1-2ms. The high time corresponds to a specific position determined by the specific servo being used. RC servo inputs, measure this high time and return the corresponding position for a servo.

Set Configuration

The table below aligns the Digital entities and the RCServo entities for configurations

digital[0]	servo[0]	Pin 0	RCServo Input
digital[1]	servo[1]	Pin 1	RCServo Input
digital[2]	servo[2]	Pin 2	RCServo Input
digital[3]	servo[3]	Pin 3	RCServo Input
digital[4]	servo[4]	Pin 4	RCServo Output
digital[5]	servo[5]	Pin 5	RCServo Output
digital[6]	servo[6]	Pin 6	RCServo Output
digital[7]	servo[7]	Pin 7	RCServo Output

```
stem.digital[index].setConfiguration(digitalConfigurationSignalInput) [cpp]
[python] [NET] [CCA] [REST]
```

Get/Set Enable

This functions gets/sets the RCServo function for a given pin (pending, it has been properly configured in the digital entity). At a firmware level this enables/disables the timers.

```
stem.servo[index].setEnabled(bEnable) [cpp] [python] [NET] [CCA] [REST]
stem.servo[index].getEnable() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/
(serial_num)/servo/(index)/enable
```

Get/Set Position

This functions gets/sets the RCServo position. For outputs this will return the currently set position; however, for inputs it will return the value seen at the pin pending the pulse is valid. If the pulse or period are invalid a zero will be returned along with the error code aErrRange. Only index 4-7.

```
stem.servo[index].setPosition(position) [cpp] [python] [NET] [CCA] [REST]
stem.servo[index].getPosition() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/
(serial_num)/servo/(index)/position
```

Get/Set Reverse

This functions gets/sets the reverse (invert) option in the RCServo Class. Only index 4-7.

```
stem.servo[index].setReverse(reverse) [cpp] [python] [NET] [CCA] [REST]
stem.servo[index].getReverse() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/
(serial_num)/servo/(index)/reverse
```

I2C

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read/Write Data

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's. The MTM-IO-SERIAL includes access to a single I2C bus operating at a set 1Mbit/s rate.

Note: *The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.*

```
stem.i2c[index].read(address,length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

The maximum data size for individual read and write operations on an I2C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Pullup

Each I2C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-IO-SERIAL in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I2C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I2C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c[index].setPullup(bEnable) \[cpp\] \[python\] \[NET\] \[CCA\]
```

USB

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USB Entity provides the software control interface for USB related features. This entity is supported by BrainStem products which have programmatically controlled USB features.

The usb entity manages the software-controllable downstream USB 2.0 channels of the MTM-IO-Serial (there are also two non-software-controllable USB channels on the module, one through the edge connector and the other through the onboard type-A connector, which are always on), as well as the upstream USB connection mode. All downstream USB ports are configured as SDP (Standard Data Port).

Downstream

Each of the four software-controllable USB channels (ports 0-3) can be individually manipulated using the usb entity. The API individually controls port power, data, or both together

Power

```
stem.usb.setPowerEnable(port) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPowerDisable(port) [cpp] [python] [NET] [CCA] [REST]
```

Data

```
stem.usb.setDataEnable(port) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setDataDisable(port) [cpp] [python] [NET] [CCA] [REST]
```

Port

```
stem.usb.setPortEnable(port) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setPortDisable(port) [cpp] [python] [NET] [CCA] [REST]
```

Upstream Mode

The MTM-IO-Serial has two (2) upstream USB connection options: through the edge connector or via the mini-B connector on the board itself. The upstream mode can be set or read using the usb entity

```
stem.usb.setUpstreamMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getUpstreamMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

The mode parameter is an integer that correlates to the following:

Mode	Result
0	Edge Connector
1	Mini-B Connector
2	Auto

Auto configuration chooses the upstream connection based on the presence or absence of VBUS power at the mini-B connector; if VBUS is present, the mini-B connector is used, otherwise the edge connector is used.

Upstream State

Gets the upstream switch state for the USB upstream ports. Returns none if no ports are plugged in, port 0 if the mode is set correctly and a cable is plugged into port 0, and port 1 if the mode is set correctly and a cable is plugged into port 1

```
stem.usb.getUpstreamState() [cpp] [python] [NET] [CCA] [REST]
```

Hub Mode

In addition to targeting individual downstream USB ports, a bit-mapped hub state interface is also available. This interface allows the reading or setting of all USB downstream ports in one functional call

```
stem.usb.setHubMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.getHubMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

Bit	Hub Operational Mode Result Bitwise Description
0	USB Channel 0 USB Hi Speed Data Enabled
1	USB Channel 0 USB VBUS Enabled
2	USB Channel 1 USB Hi Speed Data Enabled
3	USB Channel 1 USB VBUS Enabled
4	USB Channel 2 USB Hi Speed Data Enabled
5	USB Channel 2 USB VBUS Enabled
6	USB Channel 3 USB Hi Speed Data Enabled
7	USB Channel 3 USB VBUS Enabled
8:31	Reserved

Port State

Each downstream port reports information regarding its operating state represented in bit-packed results from

```
stem.usb.getPortState(mode) [cpp] [python] [NET] [CCA] [REST]
```

where channel can be [0-3], and the value status is 32-bit word, defined as the following:

Bit	Port State Result Bitwise Description
0	USB VBUS Enabled
1	USB2 Data Enabled
2:18	Reserved
19	USB Error Flag
20	USB2 Boost Enabled
21:31	Reserved

Hi-Speed Data

Enables or Disables Hi Speed data only for given downstream channel. This call enables or disables the usb data (+) and data (-) lines for the given channel.

```
stem.usb.setHiSpeedDataEnable(port) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.usb.setHiSpeedDataDisable(port) [cpp] [python] [NET] [CCA] [REST]
```

Hub and Port Error Status

Errors can be cleared on each individual channel (0, 1, 2 or 3) by calling the following method:

```
stem.usb.getPortError(channel) [cpp] [python] [NET] [CCA] [REST]
```

Details about the hub error status 32-bit word are as follows:

Bit	Hub Error Status Result Bitwise Description
0	USB CH0 overcurrent limit exceeded[1]
1	USB CH0 VBUS back drive[2]
2	USB CH0 hub power system failure
3	USB CH0 VBUS discharge[3]
4:7	Reserved
8	USB CH1 overcurrent limit exceeded4
9	USB CH1 VBUS back drive5
10	USB CH1 hub power system failure
11	USB CH1 VBUS discharge6
12:15	Reserved
16	USB CH2 overcurrent limit exceeded4
17	USB CH2 VBUS back drive5
18	USB CH2 hub power system failure
19	USB CH2 VBUS discharge6
20:23	Reserved
24	USB CH3 overcurrent limit exceeded
25	USB CH3 VBUS back drive5
26	USB CH3 hub power system failure
27	USB CH3 VBUS discharge6
28:31:00	Reserved
[1] Current limit value is defined by API settings section on USB Downstream Channels.	
[2] VBUS exceeds 5.150V for longer than 5ms.	
[3] VBUS discharge circuitry is activated. At the end of the 200ms the hub will confirm that VBUS was discharged if the VBUS voltage is not below 0.750V.	

To clear a Port error status

```
stem.usb.clearPortErrorStatus(channel) [cpp] [python] [NET] [CCA] [REST]
```

UART

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The UART entity is a class which allows a user to configure and control an arbitrary stream of serial data. These streams can represent a number of different transports, including a RS232 external interface, a virtual COM port, and onboard UART interfaces from system components. This entity allows each transport to be configured as either an endpoint, or as a passthrough between transports, similar to a switchboard of a telephone operator.

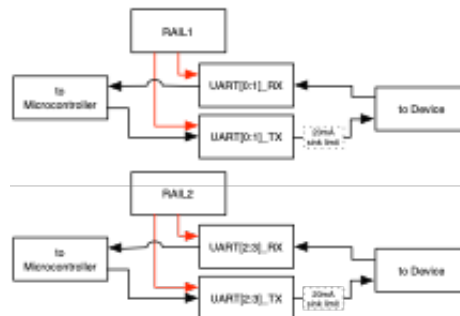
UART entities provide a mechanism to enable and disable UART data lines. All the UARTs that are passed down from the MTM-IO-Serial module can be turned on/off through software control. If a voltage is applied that is higher than the current rail voltage setpoint, each UART transmit line is current limited to 20mA sinking. Therefore, only a small amount of current will flow into the device, preventing any damage to the MTM-IO-SERIAL module's hardware

Get/Set Enable

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.uart[index].setEnabled(bEnable) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.uart[index].getEnable() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```



Rail

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Rail entity provides power control to connected devices on some modules. Check the module datasheet to determine if the module has this capability.

the Rail entity controls power provided to downstream devices, it has the ability to enable and disable power, can read voltage on the rail, and provides current consumption information on some modules. There are additional capabilities that certain modules provide which enhance basic power delivery through Kelvin sensing, or by bringing online separate power management functionality.

Certain modules may provide more than one power rail. These are independently controlled and can be accessed via the entity index.

Rails allow other devices and peripherals to consume power from the MTM-IO-SERIAL module in a controlled fashion. Three (3) different rails are available for use in a variety of application: a single fixed 5.0V rail (RAIL0)

and 2 adjustable voltage rails (RAIL1, RAIL2). These rails are accessed through an array of BrainStem rail class entities. The MTM-IO-SERIAL module implements a subset of the BrainStem rail class for each of these rails. The implemented rail entity options for each entity index are summarized below.

Enable

All three rails can be switched on or off through using the API

```
stem.rail[index].setEnabled(state) [cpp] [python] [NET] [CCA] [REST]
```

Rail0 Operational Mode

RAIL0 can be configured to use two different regulation stages: linear (LDO) or switch-mode power supply (SMPS)

```
stem.rail[index].setOperationalMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.rail[index].getOperationalMode(mode) [cpp] [python] [NET] [CCA] [REST]
```

Mode	Result
0	Auto
1	Linear
2	Switcher

Operational State

Auto configuration chooses the switch-mode power supply if an input voltage greater than 7.25V is applied, and the linear regulator otherwise. The API can be used to read the actual operational state

```
stem.rail[index].getOperationalState(state) [cpp] [python] [NET] [CCA] [REST]
```

State	Result
0	Linear
1	Switcher

Rail0 Temperature

The printed circuit board (PCB) temperature can be monitored at the 5.0V rail (RAIL0) linear regulation stage. Reading this value is possible through the API

```
stem.rail[index].getTemperature() [cpp] [python] [NET] [CCA] [REST]
```

Temperature monitoring is also used internally to prevent the power regulation stage from overheating and self-preserving the power stage. If an overtemperature condition occurs, then the MTM-IO-Serial module will disable the linear regulator until safe operating temperatures are reached.

Rail1 and Rail2 Voltage Setting

RAIL1 and RAIL2 always use linear regulators to generate their adjustable voltages. They can be set or read using the API

```
stem.rail[index].setVoltageSetpoint(microvolts) [cpp] [python] [NET] [CCA] [REST]
stem.rail[index].getVoltageSetpoint(microvolts) [cpp] [python] [NET] [CCA] [REST]
```

Rail Voltage

Getting the rail voltage from any of the rails can be used by implementing

```
stem.rail[index].getVoltage(microvolts) [cpp] [python] [NET] [CCA] [REST]
```

Rail Protection

Each rail is current limited in hardware to 100mA and will operate in constant-current mode upon reaching 100mA. Extended operation in constant-current mode is discouraged and may result in thermal shutdown of the rail.

Each rail is automatically disconnected when an overvoltage condition is detected and automatically reconnected if the overvoltage condition ceases. Overvoltage detection is implemented in hardware and based on the rail's voltage setpoint.

Signal

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

SignalClass. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Signal entity to Digital entity mapping varies from device to device. Please refer to the datasheet.

The MTM-IO-SERIAL board has 5 Signal Input and 4 Signal outputs. The Signal entity allows you to generate square waves by supplying a period and a time high value. The Signal inputs can also be used as counters. The Signal entity is an overload of the Digital entity and must first be properly configured before it can be enabled.

Set Configurations

The configurations for the signal inputs are found in the table below.

```
stem.digital[index].setConfiguration(digitalConfigurationSignalInput) [cpp]
[python] [NET] [CCA] [REST]
```

Pin	Input	Output	Rail	Hi-Z	Servo	Signal
DIO 0	Yes	Yes	1	No	Input	Input
DIO 1	Yes	Yes	1	No	Input	Input / Out- put
DIO 2	Yes	Yes	1	No	Input	Input / Out- put
DIO 3	Yes	Yes	1	No	Input	Input / Out- put
DIO 4	Yes	Yes	2	No	Output	Input / Out- put
DIO 5	Yes	Yes	2	No	Output	.
DIO 6	Yes	Yes	2	No	Output	.
DIO 7	Yes	Yes	2	No	Output	.

Get/Set Enable

```
stem.signal[index].setEnabled(bEnable) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.signal[index].getEnable() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set T3 Time

The T3 time defines the period of the waveform in nano seconds.

```
stem.signal[index].setT3Time(period) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.signal[index].getT3Time() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set T2 Time

The T2 time defines the high period of the waveform in nano seconds.

```
stem.signal[index].setT2Time(timeHigh) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.signal[index].getT2Time() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Invert

Inverts the meaning of the T2 time. When inverted the T2 time will represent the time in nano seconds that the waveform is low.

```
stem.signal[index].setInvert() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.signal[index].getInvert() [cpp] [python] [NET] [CCA] [REST]
```

Store

API Documentation: [cpp] [python] [NET] [CCA] [REST]

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Every BrainStem module includes several Store entities and onboard memory slots.

The MTM-IO-SERIAL has store slots [0-1].

Store Slot	Storage Type
0	RAM
1	Internal

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

```
stem.store[index].getSlotState(slot) [cpp] [python] [NET] [CCA] [REST]
```

Load/Unload Slot

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

```
stem.store[index].loadSlot(slot, data, _=None) [cpp] [python] [NET] [CCA]
```

This command reads the slot in the given store into the byte buffer. The length will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

```
stem.store[index].unloadSlot(slot) [cpp] [python] [NET] [CCA]
```

Enable/Disable Slot

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotEnable(slot) [cpp] [python] [NET] [CCA]
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotDisable(slot) [cpp] [python] [NET] [CCA]
```

Slot Capacity

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

```
stem.store[index].slotCapacity(slot) [cpp] [python] [NET] [CCA]
```

Slot Size

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

```
stem.store[index].slotSize(slot) [cpp] [python] [NET] [CCA]
```

System

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every MTM-IO-SERIAL is assigned a unique serial number at the factory. This facilitates an arbitrary number of MTM-IO-SERIAL devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) [cpp] [python] [NET] [CCA] [REST]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for MTM-IO-SERIAL is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) [cpp] [python] [NET] [CCA] [REST]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-IO-SERIAL away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Reset

Reset the system.

```
stem.system.reset() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set LED

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

```
stem.system.getLED(value) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Boot Slot

Get the store slot which is mapped when the module boots. Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

```
stem.system.getBootSlot() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Input Voltage

Get the module's input voltage.

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Get Version

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

```
stem.system.getVersion() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Module Software Offset

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on the BrainStem network. You must save and reset the module for this change to become effective.

```
stem.system.getModuleSoftwareOffset() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set HB Interval

Get the delay between heartbeat packets. Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Increments valid values are 1-255; default is 10 (256 milliseconds).

```
stem.system.getHBInterval() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Router

Get the router address the module uses to communicate with the host. Set the router address the module uses to communicate with the host.

```
stem.system.getRouter() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Router Address Setting

Get the router address setting saved in the module. This setting may be different from the effective router if the router has been set and saved but no reset has been made.

```
stem.system.getRouterAddressSetting() [cpp] [python] [NET] [CCA] [REST]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [CCA] [REST]
```

Get Model

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

```
stem.system.getModel() [cpp] [python] [NET] [CCA] [REST]
```

Route to Me

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

```
stem.system.routeToMe(value) [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
digital[0-7]	setConfiguration	
	getConfiguration	
	setState	
	getState	
rcservo[0-7]	setEnabled	
	getEnable	
	setPosition	Index 4-7 only
	getPosition	
	setReverse	Index 4-7 only
	getReverse	
i2c[0]	write	
	read	
usb[0]	setPortEnable	
	setPortDisable	
	setDataEnable	
	setDataDisable	
	setHiSpeedDataEnable	

continues on next page

Table 36 – continued from previous page

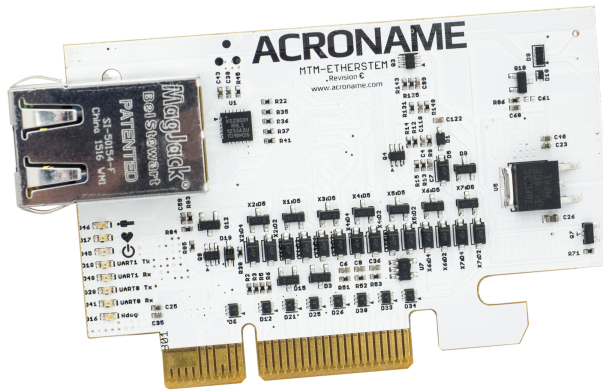
Entity Class	Entity Option	Variable(s) Notes
	setHiSpeedDataDisable	
	setPowerEnable	
	setPowerDisable	
	getPortError	
	clearPortErrorStatus	
	getSystemTemperature	In microcelsius
	setUpstreamMode	
	getUpstreamMode	
	getUpstreamState	
	getDownstreamDataSpeed	
	getHubMode	
	setHubMode	
	getPortState	
UART[0-3]	setEnabled	
	getEnable	
rail[0]	setEnabled	
	getEnable	
	getTemperature	In microcelsius
	setOperationalMode	
	getOperationalMode	
	getOperationalState	
	getVoltage	In microvolts
rail[1-2]	setEnabled	
	getEnable	
	setVoltageSetpoint	In microvolts
	getVoltageSetpoint	In microvolts
	getVoltage	In microvolts
signal[0-5]	setEnabled	
	getEnable	
	setInvert	
	getInvert	
	setT3Time	
	getT3Time	
	setT2Time	
	getT2Time	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	

continues on next page

Table 36 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	
port[0-3]	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getPowerEnabled	
	setPowerEnabled	
	getName	
	setName	
	getDataRole	
	getErrors	
port[4]	getDataRole	
USBSysytem [0]	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	

1.7.6 MTM-EtherStem



The MTM EtherStem is a BrainStem link module and part of Acroname's Manufacturing Test Module (MTM) system. It allows a TCP/IP based Ethernet connection to a host PC which may be used for test direction, control and data collection. It may also be used to load reflex programs to MTM or BrainStem modules for PC-free operation.

This module provides a TCP/IP Ethernet link to a host PC or network for test automation control and data collection. Using this link and the BrainStem API, any host based application can interact with a device under test, other MTM module(s), test station hardware and custom peripherals, as well as log and store data from a test.

To get up to speed with the MTM EtherStem and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [Programming interface](#) of the MTM Power Module for a more in depth view.

Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)²⁸ for your particular operating system and architecture.
- Download [HubTool](#)²⁹ for your particular operating system and architecture.

²⁸ <https://acroname.com/api>

²⁹ <https://acroname.com/hubtool>

2. Connect to Device

- Utilize the MTM EtherStem by either connecting to the:
 - Onboard USB connection
 - Card edge USB input
 - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM EtherStem device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the MTM EtherStem. For more information please take a look at our [Getting Started Guide](#)

Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a MTM-ETHERSTEM module.
    aMTMEtherStem mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
        return 1;
    } else { printf("Connected to BrainStem module.\n"); }

    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

    //Disconnect
    mtm.disconnect();
}
```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-ETHERSTEM module.
mtm = brainstem.stem.MTMEtherStem();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

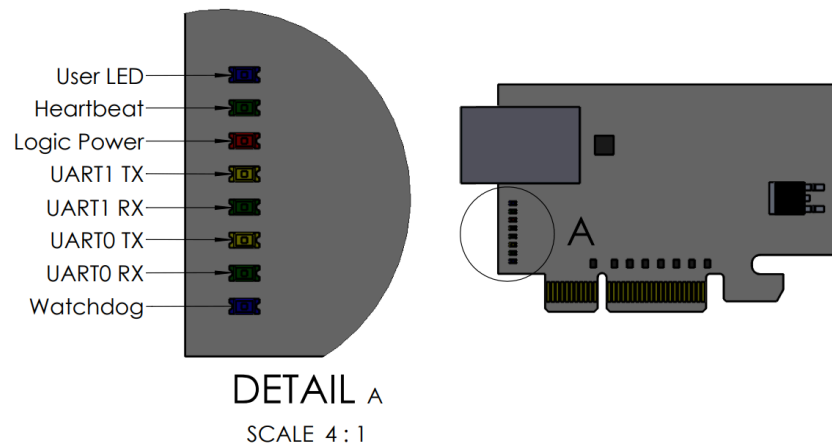
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

Indicators and Connections

LEDs

The MTM-EtherStem board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below



Programming Interface

The MTM-USBetherStem is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-USBetherStem.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-USBetherStem. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-USBetherStem can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-USBetherStem is attached to a host machine, a user can connect to it via software API:

```
stem.discoverAndConnect(linkType, serialNumber)
```

The MTM-USBetherStem can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I2C. Each MTM-USBetherStem is uniquely addressable via hardware or software to avoid communication conflicts on the I2C bus.

Link and TCP/IP Settings

The MTM-USBetherStem supports host computer connections over its Ethernet jack via TCP/IP sockets. The MTM-USBetherStem is designed to interact on the local network segment only. Typical setup is a direct Ethernet connection between a host test machine and the MTM-EtherStem. The host can run a DHCP server to provide an IP address to the module or, without a DHCP server, the MTM-USBetherStem will fall back to a static IP address of 192.168.44.42/24 when it does not receive a response to DHCP requests. In the fallback IP configuration, manually configuring the host machine interface to communicate on this subnet will enable communication to the module.

The module features a limited DHCP client which will not function across a network bridge or other gateway mechanism. The MTM-USBetherStem will respond to ICMP "ping" requests including broadcast requests. The brainstem API interface performs a discovery process prior to establishing communication with the MTM-EtherStem. Brainstem discovery over IP is accomplished using a UDP multicast request on port 9888, and a response on the stem to the host UDP port 9889. The MTM-USBetherStem listens for socket connections on TCP port 8000. Firewall rules will need to be configured allowing the outgoing multicast request on 9888 and incoming response on 9889, as well as outgoing socket TCP connections to port 8000.

MTM-ETHERSTEM Module Entities

Digital

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

Set/Get Configurations

Gets or Sets the digital pin configuration. Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware. Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

```
stem.digital[index].setConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The mode parameter is an integer that correlates to the following:

Value	Configuration
0	Input
1	Output
2	RC Servo Input
3	RC Servo Output
4	Hi Impedance (Hi-Z)
5	Input with Pull Down

Set/Get State

Gets or Sets the digital I/O Value. For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet. For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

```
stem.digital[index].setState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

If a digital pin is configured in Hi-Z mode its internal circuitry has been disconnected to create a high impedance. There are no functions that can act on this configuration

Pin	Input	Output	Hi-Z	RCServo
DIO 0	Yes	Yes	Yes	Input
DIO 1	Yes	Yes	Yes	Input
DIO 2	Yes	Yes	Yes	Input
DIO 3	Yes	Yes	Yes	Input
DIO 4	Yes	Yes	Yes	Output
DIO 5	Yes	Yes	Yes	Output
DIO 6	Yes	Yes	Yes	Output
DIO 7	Yes	Yes	Yes	Output
DIO 8	Yes	Yes	Yes	.
DIO 9	Yes	Yes	Yes	.
DIO 10	Yes	Yes	Yes	.
DIO 11	Yes	Yes	Yes	.
DIO 12	Yes	Yes	Yes	.
DIO 13	Yes	Yes	Yes	.
DIO 14	Yes	Yes	Yes	.

RCServo

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The RCServo entity provides a pulsed signal based on the RC servo standard. This consist of a period lasting 20ms with a high pulse between 1-2ms. The time high corresponds to a specific position determined by the servo being used. For example if you are using a 90 degree servo a 1.5ms pulse will correspond to the 45 degrees. 1ms and 2ms pulses will correspond to 0 and 90 degree positions respectively.

The RCServo entity is an overload to the [Digital Entity](#) and therefor requires proper configuration of the Digital entity before the RCServo entity can be enabled.

The MTM-ETHERSTEM board is equipped with 4 RC servo inputs and 4 RC servo outputs. The RC Servo entity is an overload of the Digital Entity and thus requires proper configuration before this entity can be enabled.

With the RC servo entity, digital output pins generate pulsed signal based on the RC Servo standard consisting of a period lasting 20ms and high pulse time between 1-2ms. The high time corresponds to a specific position determined by the specific servo being used. RC servo inputs, measure this high time and return the corresponding position for a servo.

Get/Set Enable

This functions gets/sets the RCServo function for a given pin (pending, it has been properly configured in the digital entity). At a firmware level this enables/disables the timers.

```
stem.servo[index].setEnabled(bEnable) [cpp] [python] [NET] [CCA] [REST]
stem.servo[index].getEnable() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/
(serial_num)/servo/(index)/enable
```

Get/Set Position

This functions gets/sets the RCServo position. For outputs this will return the currently set position; however, for inputs it will return the value seen at the pin pending the pulse is valid. If the pulse or period are invalid a zero will be returned along with the error code aErrRange. Only index 4-7.

```
stem.servo[index].setPosition(position) [cpp] [python] [NET] [CCA] [REST]
stem.servo[index].getPosition() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/
(serial_num)/servo/(index)/position
```

Get/Set Reverse

This functions gets/sets the reverse (invert) option in the RCServo Class. Only index 4-7.

```
stem.servo[index].setReverse(reverse) [cpp] [python] [NET] [CCA] [REST]
stem.servo[index].getReverse() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/
(serial_num)/servo/(index)/reverse
```

I2C

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read/Write Data

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's. The MTM-ETHERSTEM includes access to a single I2C bus operating at a set 1Mbit/s rate.

Note: *The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.*

```
stem.i2c[index].read(address,length) [cpp] [python] [NET] [CCA]
stem.i2c[index].write(address,length) [cpp] [python] [NET] [CCA]
```

The maximum data size for individual read and write operations on an I2C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Pullup

Each I2C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-ETHERSTEM in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I2C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I2C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c[index].setPullup(bEnable) [cpp] [python] [NET] [CCA]
```

Analog

API Documentation: *[cpp] [python] [.NET] [CCA] [REST]*

BrainStem modules may have the ability to read an analog voltage (ADC) and convert this into a discrete digitized value or output a voltage value based on a desired discrete value (DAC). Analog voltage capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of analog entities and details for their capacities will be described in that module's datasheet.

Initiate Bulk Capture

The system can capture any number of samples up the size of the RAM_STORE slot 0 (8191). The capture is then triggered with

```
stem.analog[index].initiateBulkCapture() [cpp] [python] [NET] [CCA] [REST]
```

Results of the capture are stored in the RAM_STORE slot 0. Results are always stored in ADC counts as two little-endian byte pairs with the second byte the most significant. Computing a sample value from the Store read out is:

Get/Set Bulk Capture Sample Rate and Number of Samples

The MTM-EtherStem's ADC's are also capable of being captured in bulk based on a user defined sample rate. For additional information on sample rate settings. Configuring and triggering the bulk capture is accomplished by setting the number of samples and the sample rate, then triggering the capture.

To set the number of samples and the sample rate

```
stem.analog[index].setBulkCaptureSampleRate() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].getBulkCaptureSampleRate() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].setBulkCaptureNumberOfSamples() [cpp] [python] [NET] [CCA]  
[REST]
```

```
stem.analog[index].getBulkCaptureNumberOfSamples() [cpp] [python] [NET] [CCA]  
[REST]
```

Get Bulk Capture State

```
stem.analog[index].getBulkCaptureState() [cpp] [python] [NET] [CCA] [REST]
```

Get Voltage/Value

A BrainStem's A2D reading will always return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the measurement will get propagated up to 16 bits wide.

For example, if a 12-bit A2D engine exists in the target module's hardware, the reading will get promoted in the firmware layer by shifting up 4 bits to fill out the 16 bit value ($0x0FFF =: 0x0FFF \ll 4 = 0xFFF0$) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

setValue and setVoltage is only applicable for Analog[3]:

```
stem.analog[index].setValue() [cpp] [python] [NET] [CCA] [REST]
stem.analog[index].getValue() [cpp] [python] [NET] [CCA] [REST]
stem.analog[index].setVoltage(microvolts) [cpp] [python] [NET] [CCA] [REST]
stem.analog[index].getVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Signal

API Documentation: [cpp] [python] [.NET] [CCA] [REST]

SignalClass. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Signal entity to Digital entity mapping varies from device to device. Please refer to the datasheet.

The MTM-ETHERSTEM board has 5 Signal Input and 4 Signal outputs. The Signal entity allows you to generate square waves by supplying a period and a time high value. The Signal inputs can also be used as counters. The Signal entity is an overload of the Digital entity and must first be properly configured before it can be enabled.

Set Configurations

Digital IO Pin Configurations, configure the correct digital pin as a Signal input:

```
stem.digital[index].setConfiguration(digitalConfigurationSignalInput) [cpp]
[python] [NET] [CCA] [REST]
```

Get/Set Enable

```
stem.signal[index].setEnabled(bEnable) [cpp] [python] [NET] [CCA] [REST]
stem.signal[index].getEnable() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set T3 Time

The T3 time defines the period of the waveform in nano seconds.

```
stem.signal[index].setT3Time(period) [cpp] [python] [NET] [CCA] [REST]
stem.signal[index].getT3Time() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set T2 Time

The T2 time defines the high period of the waveform in nano seconds.

```
stem.signal[index].setT2Time(timeHigh) [cpp] [python] [NET] [CCA] [REST]
stem.signal[index].getT2Time() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Invert

Inverts the meaning of the T2 time. When inverted the T2 time will represent the time in nano seconds that the waveform is low.

```
stem.signal[index].setInvert() [cpp] [python] [NET] [CCA] [REST]
stem.signal[index].getInvert() [cpp] [python] [NET] [CCA] [REST]
```

Store

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Every BrainStem module includes several Store entities and onboard memory slots.

The MTM-ETHERSTEM has store slots [0-2].

Store Slot	Storage Type	Available Slots
0	RAM	12
1	Internal	1
2	SD	0-255

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

```
stem.store[index].getSlotState(slot) [cpp] [python] [NET] [CCA] [REST]
```

Load/Unload Slot

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

```
stem.store[index].loadSlot(slot, data, _=None) [cpp] [python] [NET] [CCA]
```

This command reads the slot in the given store into the byte buffer. The length will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

```
stem.store[index].unloadSlot(slot) [cpp] [python] [NET] [CCA]
```

Enable/Disable Slot

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotEnable(slot) [cpp] [python] [NET] [CCA]
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotDisable(slot) [cpp] [python] [NET] [CCA]
```

Slot Capacity

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

```
stem.store[index].slotCapacity(slot) [cpp] [python] [NET] [CCA]
```

Slot Size

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

```
stem.store[index].slotSize(slot) [cpp] [python] [NET] [CCA]
```

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every MTM-ETHERSTEM is assigned a unique serial number at the factory. This facilitates an arbitrary number of MTM-ETHERSTEM devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) [cpp] [python] [NET] [CCA] [REST]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for MTM-ETHERSTEM is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) [cpp] [python] [NET] [CCA] [REST]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-ETHERSTEM away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Reset

Reset the system.

```
stem.system.reset() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set LED

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

```
stem.system.getLED(value) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Boot Slot

Get the store slot which is mapped when the module boots. Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

```
stem.system.getBootSlot() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Input Voltage

Get the module's input voltage.

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Get Version

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

```
stem.system.getVersion() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Module Software Offset

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on the BrainStem network. You must save and reset the module for this change to become effective.

```
stem.system.getModuleSoftwareOffset() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set HB Interval

Get the delay between heartbeat packets. Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Increments valid values are 1-255; default is 10 (256 milliseconds).

```
stem.system.getHBInterval() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [CCA] [REST]
```


Get/Set Router

Get the router address the module uses to communicate with the host. Set the router address the module uses to communicate with the host.

```
stem.system.getRouter() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Router Address Setting

Get the router address setting saved in the module. This setting may be different from the effective router if the router has been set and saved but no reset has been made.

```
stem.system.getRouterAddressSetting() [cpp] [python] [NET] [CCA] [REST]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [CCA] [REST]
```

Get Model

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

```
stem.system.getModel() [cpp] [python] [NET] [CCA] [REST]
```

Route to Me

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

```
stem.system.routeToMe(value) [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

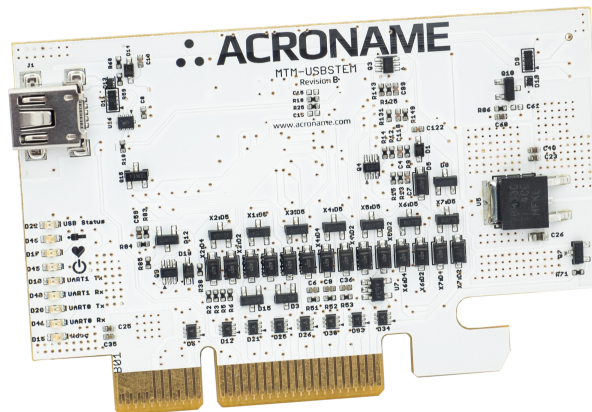
Entity Class	Entity Option	Variable(s)	Notes
digital[0-14]	setConfiguration		
	getConfiguration		
	setState		
	getState		
rcservo[0-7]	setEnable		
	getEnable		
	setPosition	Index 4-7 only	
	getPosition		
	setReverse	Index 4-7 only	
	getReverse		
i2c[0-1]	write		
	read		
analog[0-2]	getValue		
	getVoltage		
	setBulkCaptureSampleRate		
	getBulkCaptureSampleRate		
	setBulkCaptureNumberOfSamples		
	getBulkCaptureNumberOfSamples		
	initiateBulkCapture		
	getBulkCaptureState		
analog[3]	setValue		
	setVoltage		
signal[0-5]	setEnable		
	getEnable		
	setInvert		
	getInvert		
	setT3Time		
	getT3Time		
	setT2Time		
	getT2Time		
store[0-2]	getSlotState		
	loadSlot		
	unloadSlot		
	slotEnable		
	slotDisable		
	slotCapacity		
	slotSize		
system[0]	save		
	reset		
	setLED		
	getLED		
	setBootSlot		
	getBootSlot		
	getInputVoltage		
	getVersion		
	getModuleBaseAddress		
	getModuleSoftwareOffset		
	setModuleSoftwareOffset		

continues on next page

Table 37 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

1.7.7 MTM-USBStem



The MTM-USBStem is a BrainStem link module and part of Acroname's Manufacturing Test Module (MTM) system. It allows a USB connection to a host PC which may be used for test direction, control and data collection. It may also be used to load reflex programs to MTM or BrainStem modules for PC-free operation.

With the upstream USB connection on its industry-standard edge connector, the MTM-USBStem is designed to allow even complex and dense test stations to connect to a PC, using just one cable and daisy-chaining as many MTM modules as needed. This makes for rapid and error free station bring-up.

To get up to speed with the MTM USBStem and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [Programming interface](#) of the MTM Power Module for a more in depth view.

Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)³⁰ for your particular operating system and architecture.
- Download [HubTool](#)³¹ for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM USBStem by either connecting to the:
 - Onboard USB connection
 - Card edge USB input
 - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM USBStem device.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

Congratulations! You are now ready to start exploring the capabilities of the MTM USBStem. For more information please take a look at our [Getting Started Guide](#)

Basic Example

C++

```
#include <iostream>
#include "BrainStem2/BrainStem-all.h"

int main(int argc, const char * argv[]) {

    //Create an instance of a MTM-USBSTEM module.
    aMTMUSBStem mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
        return 1;
    } else { printf("Connected to BrainStem module.\n"); }
```

(continues on next page)

³⁰ <https://acroname.com/api>

³¹ <https://acroname.com/hubtool>

(continued from previous page)

```

//Basic initialization (Get LEDs turned off).
mtm.system.setLED(0);

//Ready for testing
//Enable LED
mtm.system.setLED(1);

//Turn LED off
mtm.system.setLED(0);

//Disconnect
mtm.disconnect();
}

```

Python

```

import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-USBSTEM module.
mtm = brainstem.stem.MTMUSBStem();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

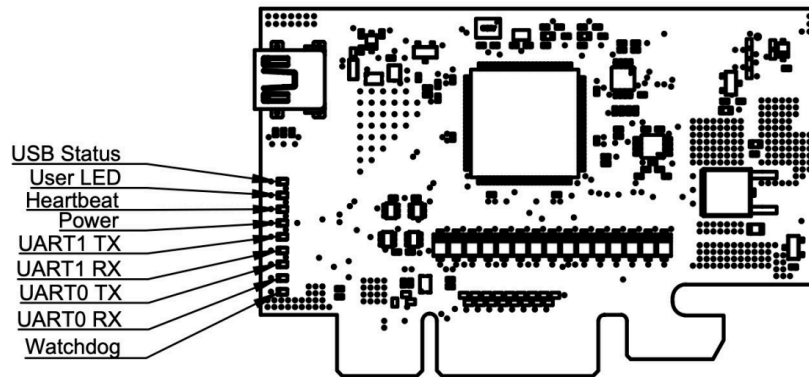
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

Indicators and Connections

LEDs

The MTM-USBStem board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below



Programming Interface

The MTM-USBStem is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-USBStem.

A complete list of all entities and functions can be found in the [Module Entities](#) page.

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-USBStem. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-USBStem can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-USBStem is attached to a host machine, a user can connect to it via software API:

```
stem.discoverAndConnect(linkType, serialNumber)
```

The MTM-USBStem can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I2C. Each MTM-USBStem is uniquely addressable via hardware or software to avoid communication conflicts on the I2C bus.

Upstream USB Connectivity Options

The MTM-USBStem supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the miniB connector if 5V is present on Vbus at the mini-B connector.

MTM-USBSTEM Module Entities

Digital

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

Set/Get Configurations

Gets or Sets the digital pin configuration. Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware. Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

```
stem.digital[index].setConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

The mode parameter is an integer that correlates to the following:

Value	Configuration
0	Input
1	Output
2	RC Servo Input
3	RC Servo Output
4	Hi Impedance (Hi-Z)
5	Input with Pull Down
6	Signal Output
7	Signal Input

Set/Get State

Gets or Sets the digital I/O Value. For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet. For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

```
stem.digital[index].setState(level) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

```
stem.digital[index].getState(level) [cpp] [python] [NET] [CCA] [REST]
```

If a digital pin is configured in Hi-Z mode its internal circuitry has been disconnected to create a high impedance. There are no functions that can act on this configuration.

Pin	Input	Output	Hi-Z	RCServo	Signal
DIO 0	Yes	Yes	Yes	Input	.
DIO 1	Yes	Yes	Yes	Input	.
DIO 2	Yes	Yes	Yes	Input	.
DIO 3	Yes	Yes	Yes	Input	.
DIO 4	Yes	Yes	Yes	Output	Input
DIO 5	Yes	Yes	Yes	Output	Input / Output
DIO 6	Yes	Yes	Yes	Output	Input / Output
DIO 7	Yes	Yes	Yes	Output	Input / Output
DIO 8	Yes	Yes	Yes	.	Input / Output
DIO 9	Yes	Yes	Yes	.	.
DIO 10	Yes	Yes	Yes	.	.
DIO 11	Yes	Yes	Yes	.	.
DIO 12	Yes	Yes	Yes	.	.
DIO 13	Yes	Yes	Yes	.	.
DIO 14	Yes	Yes	Yes	.	.

RCServo

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The RCServo entity provides a pulsed signal based on the RC servo standard. This consist of a period lasting 20ms with a high pulse between 1-2ms. The time high corresponds to a specific position determined by the servo being used. For example if you are using a 90 degree servo a 1.5ms pulse will correspond to the 45 degrees. 1ms and 2ms pulses will correspond to 0 and 90 degree positions respectively.

The RCServo entity is an overload to the [Digital Entity](#) and therefor requires proper configuration of the Digital entity before the RCServo entity can be enabled.

The MTM-USBSTEM board is equipped with 4 RC servo inputs and 4 RC servo outputs. The RC Servo entity is an overload of the Digital Entity and thus requires proper configuration before this entity can be enabled.

With the RC servo entity, digital output pins generate pulsed signal based on the RC Servo standard consisting of a period lasting 20ms and high pulse time between 1-2ms. The high time corresponds to a specific position determined by the specific servo being used. RC servo inputs, measure this high time and return the corresponding position for a servo.

Get/Set Enable

This functions gets/sets the RCServo function for a given pin (pending, it has been properly configured in the digital entity). At a firmware level this enables/disables the timers.

```
stem.servo[index].setEnabled(bEnable) [cpp] [python] [NET] [CCA] \[REST\]
```

```
stem.servo[index].getEnable() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/  
\(serial\_num\)/servo/\(index\)/enable
```

Get/Set Position

This functions gets/sets the RCServo position. For outputs this will return the currently set position; however, for inputs it will return the value seen at the pin pending the pulse is valid. If the pulse or period are invalid a zero will be returned along with the error code aErrRange. Only index 4-7.

```
stem.servo[index].setPosition(position) [cpp] [python] [NET] [CCA] \[REST\]
```

```
stem.servo[index].getPosition() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/  
\(serial\_num\)/servo/\(index\)/position
```

Get/Set Reverse

This functions gets/sets the reverse (invert) option in the RCServo Class. Only index 4-7.

```
stem.servo[index].setReverse(reverse) [cpp] [python] [NET] [CCA] \[REST\]
```

```
stem.servo[index].getReverse() [cpp] [python] [NET] [CCA] GET /api/v1/brainstem/  
\(serial\_num\)/servo/\(index\)/reverse
```

I2C

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read/Write Data

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's. The MTM-USBSTEM includes access to a single I2C bus operating at a set 1Mbit/s rate.

Note: *The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.*

```
stem.i2c[index].read(address, length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

```
stem.i2c[index].write(address, length) \[cpp\] \[python\] \[NET\] \[CCA\]
```

The maximum data size for individual read and write operations on an I2C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Pullup

Each I2C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-USBSTEM in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I2C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I2C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c[index].setPullup(bEnable) \[cpp\] \[python\] \[NET\] \[CCA\]
```

Analog

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read an analog voltage (ADC) and convert this into a discrete digitized value or output a voltage value based on a desired discrete value (DAC). Analog voltage capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of analog entities and details for their capacities will be described in that module's datasheet.

Initiate Bulk Capture

The system can capture any number of samples up the size of the RAM_STORE slot 0 (8191). The capture is then triggered with

```
stem.analog[index].initiateBulkCapture() [cpp] [python] [NET] [CCA] [REST]
```

Results of the capture are stored in the RAM_STORE slot 0. Results are always stored in ADC counts as two little-endian byte pairs with the second byte the most significant.

Get/Set Bulk Capture Sample Rate and Number of Samples

The MTM-EtherStem's ADC's are also capable of being captured in bulk based on a user defined sample rate. For additional information on sample rate settings. Configuring and triggering the bulk capture is accomplished by setting the number of samples and the sample rate, then triggering the capture.

To set the number of samples and the sample rate use:

```
stem.analog[index].setBulkCaptureSampleRate() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].getBulkCaptureSampleRate() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].setBulkCaptureNumberOfSamples() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].getBulkCaptureNumberOfSamples() [cpp] [python] [NET] [CCA] [REST]
```

Get Bulk Capture State

```
stem.analog[index].getBulkCaptureState() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Value and Voltage

A BrainStem's A2D reading will always return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the measurement will get propagated up to 16 bits wide.

For example, if a 12-bit A2D engine exists in the target module's hardware, the reading will get promoted in the firmware layer by shifting up 4 bits to fill out the 16 bit value ($0x0FFF \ll 4 = 0xFFF0$) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

setValue and setVoltage is only applicable for Analog[3]:

```
stem.analog[index].setValue() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].getValue() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].setVoltage(microvolts) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.analog[index].getVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Signal

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

SignalClass. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Signal entity to Digital entity mapping varies from device to device. Please refer to the datasheet.

The MTM-USBSTEM board has 5 Signal Input and 4 Signal outputs. The Signal entity allows you to generate square waves by supplying a period and a time high value. The Signal inputs can also be used as counters. The Signal entity is an overload of the Digital entity and must first be properly configured before it can be enabled.

Set Configurations

Digital IO Pin Configurations, configure the correct digital pin as a Signal input:

```
stem.digital[index].setConfiguration(digitalConfigurationSignalInput) \[cpp\]  
\[python\] \[NET\] \[CCA\] \[REST\]
```

Get/Set Enable

```
stem.signal[index].setEnabled(bEnable) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]  
stem.signal[index].getEnable() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Get/Set T3 Time

The T3 time defines the period of the waveform in nano seconds.

```
stem.signal[index].setT3Time(period) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]  
stem.signal[index].getT3Time() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Get/Set T2 Time

The T2 time defines the high period of the waveform in nano seconds.

```
stem.signal[index].setT2Time(timeHigh) \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]  
stem.signal[index].getT2Time() \[cpp\] \[python\] \[NET\] \[CCA\] \[REST\]
```

Get/Set Invert

Inverts the meaning of the T2 time. When inverted the T2 time will represent the time in nano seconds that the waveform is low.

```
stem.signal[index].setInvert() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.signal[index].getInvert() [cpp] [python] [NET] [CCA] [REST]
```

Store

API Documentation: [cpp] [python] [NET] [CCA] [REST]

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Every BrainStem module includes several Store entities and onboard memory slots.

The MTM-USBSTEM has store slots [0-2].

Store Slot	Storage Type	Available Slots
0	RAM	12
1	Internal	1
2	SD	0-255

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

```
stem.store[index].getSlotState(slot) [cpp] [python] [NET] [CCA] [REST]
```

Load/Unload Slot

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

```
stem.store[index].loadSlot(slot, data, _=None) [cpp] [python] [NET] [CCA]
```

This command reads the slot in the given store into the byte buffer. The length will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

```
stem.store[index].unloadSlot(slot) [cpp] [python] [NET] [CCA]
```

Enable/Disable Slot

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotEnable(slot) [cpp] [python] [NET] [CCA]
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

```
stem.store[index].slotDisable(slot) [cpp] [python] [NET] [CCA]
```

Slot Capacity

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

```
stem.store[index].slotCapacity(slot) [cpp] [python] [NET] [CCA]
```

Slot Size

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

```
stem.store[index].slotSize(slot) [cpp] [python] [NET] [CCA]
```

System

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

Serial Number

Every MTM-USBSTEM is assigned a unique serial number at the factory. This facilitates an arbitrary number of MTM-USBSTEM devices attached to a host computer. The following method call can retrieve the unique serial number for each device.

```
stem.system.getSerialNumber(serialNumber) [cpp] [python] [NET] [CCA] [REST]
```

Module Default Base Address

BrainStems are designed to be able to form a reactive, extensible network. All BrainStem modules come with a default network base address for identification on the BrainStem network bus. The default module base address for MTM-USBSTEM is factory-set as 6, and can be accessed with.

```
stem.system.getModule(module) [cpp] [python] [NET] [CCA] [REST]
```

Saved Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-USBSTEM away from the factory default settings. Saving system settings preserves the settings as the new default. Most changes to system settings require a save and reboot before taking effect. For example, upstream and downstream USB Boost settings will not take effect unless a system save operation is completed, followed by a reset or power cycle. Use the following command to save changes to system settings before reboot:

```
stem.system.save() [cpp] [python] [NET] [CCA] [REST]
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Reset

Reset the system.

```
stem.system.reset() [cpp] [python] [NET] [CCA] [REST]
```

Get/Set LED

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

```
stem.system.getLED(value) [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Boot Slot

Get the store slot which is mapped when the module boots. Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

```
stem.system.getBootSlot() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Input Voltage

Get the module's input voltage.

```
stem.system.getInputVoltage() [cpp] [python] [NET] [CCA] [REST]
```

Get Version

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

```
stem.system.getVersion() [cpp] [python] [NET] [CCA] [REST]
```


Get/Set Module Software Offset

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on the BrainStem network. You must save and reset the module for this change to become effective.

```
stem.system.getModuleSoftwareOffset() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set HB Interval

Get the delay between heartbeat packets. Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Increments valid values are 1-255; default is 10 (256 milliseconds).

```
stem.system.getHBInterval() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [CCA] [REST]
```

Get/Set Router

Get the router address the module uses to communicate with the host. Set the router address the module uses to communicate with the host.

```
stem.system.getRouter() [cpp] [python] [NET] [CCA] [REST]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [CCA] [REST]
```

Get Router Address Setting

Get the router address setting saved in the module. This setting may be different from the effective router if the router has been set and saved but no reset has been made.

```
stem.system.getRouterAddressSetting() [cpp] [python] [NET] [CCA] [REST]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [CCA] [REST]
```

Get Model

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

```
stem.system.getModel() [cpp] [python] [NET] [CCA] [REST]
```

Route to Me

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

```
stem.system.routeToMe(value) [cpp] [python] [NET] [CCA] [REST]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
digital[0-14]	setConfiguration	
	getConfiguration	
	setState	
	getState	
rcservo[0-7]	setEnable	
	getEnable	
	setPosition	Index 4-7 only
	getPosition	
	setReverse	Index 4-7 only
	getReverse	
i2c[0-1]	write	
	read	
analog[0-2]	getValue	
	getVoltage	
	setBulkCaptureSampleRate	
	getBulkCaptureSampleRate	
	setBulkCaptureNumberOfSamples	

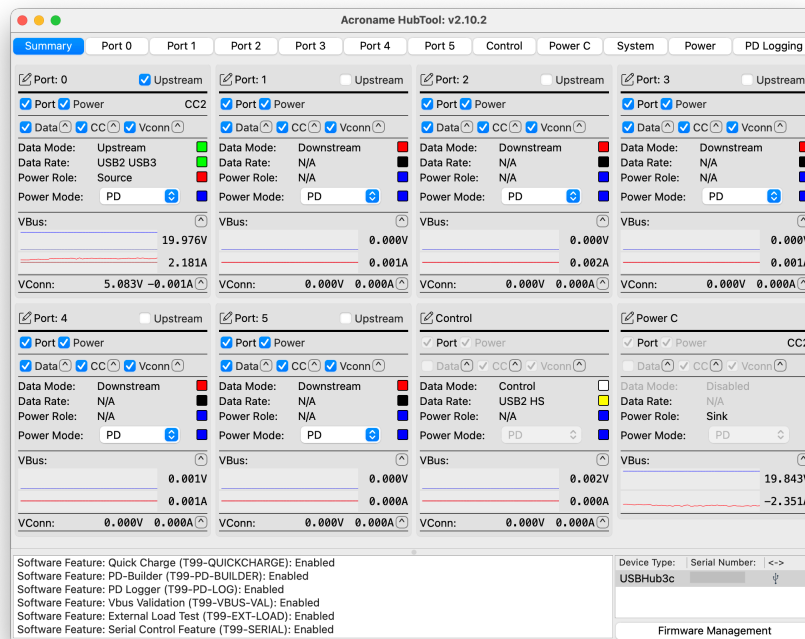
continues on next page

Table 38 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getBulkCaptureNumberOfSamples	
	initiateBulkCapture	
	getBulkCaptureState	
analog[3]	setValue	
	setVoltage	
signal[0-5]	setEnabled	
	getEnabled	
	setInvert	
	getInvert	
	setT3Time	
	getT3Time	
	setT2Time	
	getT2Time	
store[0-2]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

Here you can find software developed by Acroname.

2.1 HubTool



HubTool is a utility that lets users view detailed information and control settings of Acroname devices. Users can operate HubTool either on the USB host computer or a computer linked to the Acroname device's Control, ethernet, or RS-232 ports (if available).

2.1.1 What can it do?

HubTool provides a simple GUI to:

- Enable and disable ports (virtually plug and unplug)
 - Selectively toggle port data and power connections such as USB 3, USB 2, and V_{BUS} independently
- View connection speed
- Visualize per-port current and voltage
- Select automatic and manual upstream host port switching
- Set PD power rules (USBHub3c, USBExt3c) and port power modes
- Control Acroname devices over the local network

2.1.2 Which Acroname devices work with HubTool?

USB hubs and switches ³²	USBExt3c ³³
	USBHub3c ³⁴
	USBHub3p ³⁵
	USBHub2x4 ³⁶
	USB-C-Switch Pro ³⁷
	USB-C-Switch ³⁸
MTM manufacturing test modules ³⁹	MTM-Load-1 ⁴⁰
	MTM-DAQ-2 ⁴¹
	MTM-PM-1 ⁴²
	MTM-IO-Serial ⁴³
	MTM-Relay ⁴⁴
	MTM-EtherStem ⁴⁵
	MTM-USBStem ⁴⁶

³² https://acroname.com/store-grid/field_category/programmable-usb

³³ <https://acroname.com/store/491>

³⁴ <https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro>

³⁵ <https://acroname.com/store/programmable-industrial-hub-s79-usbhub-3p>

³⁶ <https://acroname.com/store/industrial-intelligent-4-port-hub-s77-usbhub-2x4>

³⁷ <https://acroname.com/store/s105-usbc-switch-pro>

³⁸ <https://acroname.com/store/programmable-industrial-switch-s85-rdvr-usbcsw>

³⁹ <https://acroname.com/manufacturing-test-modules-mtm>

⁴⁰ <https://acroname.com/store/s96-mtm-load-1>

⁴¹ <https://acroname.com/store/s92-mtm-daq-2>

⁴² <https://acroname.com/products/ACRONAME-MTM-1-CHANNEL-POWER-MODULE>

⁴³ <https://acroname.com/products/ACRONAME-MTM-IO-SERIAL-SOFTWARE-CONTROLLED-USB-HUB>

⁴⁴ <https://acroname.com/store/s78-mtm-relay>

⁴⁵ <https://acroname.com/store/s67-mtm-etherstem>

⁴⁶ <https://acroname.com/store/s69-mtm-usbstem>

2.1.3 Host system requirements

HubTool binaries are available for these platforms:

- Windows 10 and higher
- Intel and Apple Silicon Macs running macOS 10.15 or higher
- Linux:
 - x86_64 Ubuntu LTS 16.04, 18.04, 20.04, 22.04, 24.04 - Required dependencies: *apt install xcb**
 - x86_64 Red Hat 8 - Required dependencies: *dnf install qt5-qtbase-gui*
 - x86_64 Red Hat 9 - Required dependencies: *dnf install qt6-qtbase-gui*
 - arm64v8 Ubuntu LTS 16.04, 18.04, 20.04, 22.04, 24.04 - Required dependencies: *apt install xcb**
 - i686 Ubuntu LTS 16.04 - Required dependencies: *apt install xcb**

2.1.4 Installation

1. Download [HubTool](#)⁴⁷ and [BrainStem Development Kit \(BDK\)](#)⁴⁸
2. Open the .dmg / .zip / .tgz
3. Move the contents to a folder, or move just the HubTool application to your preferred location.

Note: *Linux users will need run the script labeled “udev.sh” located in the “BrainStem_linux_Driverless” folder before they will be able communicate with a BrainStem device.*

2.1.5 Usage

Click on the HubTool app to launch it. The lower right panel shows any Acroname devices that are connected. The lower left panel is the console. On launch, HubTool writes default settings to the console.



Startup messages in HubTool Console

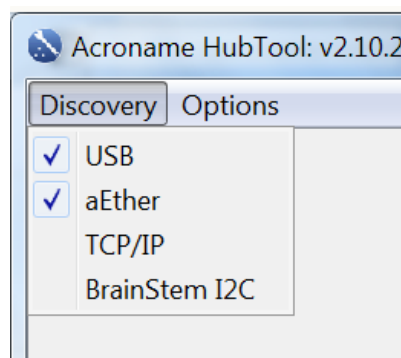
⁴⁷ <https://acroname.com/hubtool>

⁴⁸ <https://acroname.com/api>

Table 1: *Startup messages*

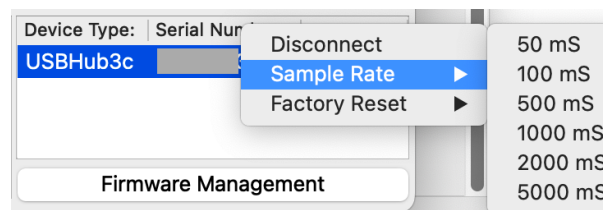
Console message	Meaning
BrainStem I2C discovery: Disabled	Discover Acroname devices connected to a USB-connected MTM module's I2C bus
USB discovery: Enabled	Discover local USB-connected Acroname devices
BrainStem aEther discovery: Enabled	Discover Acroname devices connected through other applications (e.g. BrainD), local only by default
TCP/IP discovery: Disabled	Discover MTM-EtherStem modules connected to the host by ethernet
Port Mapping: Enabled	Get USB descriptors for each downstream device, E.G. vendor ID, serial number

Discovery methods (USB, aEther, TCP/IP, and Brainstem I2C) are enabled and disabled using the Discovery menubar dropdown. Port mapping is enabled in the options menu. Both settings reset to default values with every launch of HubTool.

*Discovery menu*

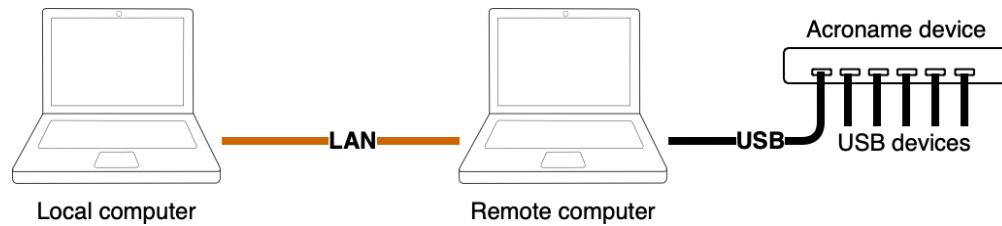
2.1.6 Connecting devices

When an Acroname device is connected, the device type, serial number, and an icon indicating the connection type will appear in the lower right pane. If the device is connected by USB to the local host, the USB icon will appear (🔌). If the device is connected to a remote host, the aEther icon will be shown instead (🌐). Click on the device serial number to view the [device interface](#). Right-click to bring up a contextual menu to *disconnect*, *set sample rate*, and perform a *factory reset*.

*Device context menu*

2.1.7 Control devices on remote hosts

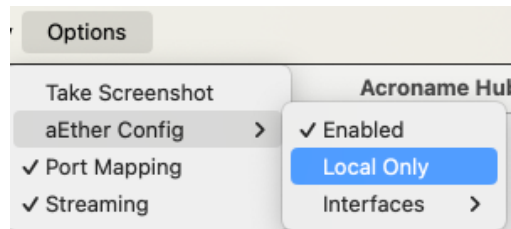
HubTool can also view and control shared Acroname devices connected to other hosts on the same network (WiFi or ethernet). To enable remote control of devices, follow these steps:



Controlling remote devices over the local network

On the remote host:

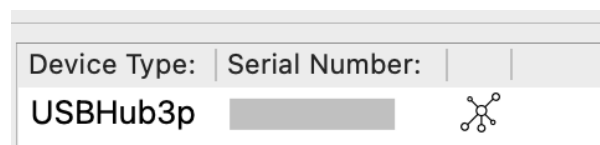
- Connect the device via USB
- Launch HubTool
- Click the device serial number to connect
- Enable aEther and deselect “Local only” in Options > aEther Config



On the local host:

- Launch HubTool
- Enable aEther and deselect “Local only” in Options > aEther Config

The serial number of the remote device and the aEther icon should appear in the local host’s device list. Click to view the remote device’s interface.



aEther-connected remote device

2.1.8 Updating device firmware

Discover					
Model String	Serial Number	Current	Target	Update	Update Status
USBCSwitch		2.10.1	▼ 2.10.2	Update	Update to: 2.10.2
USBHub3p		2.10.2	▼ 2.10.2	Update	Up to Date (Updating again will check for purchased Software Features.)
USBHub3c		2.10.2	<div> <div>✓ 2.10.2</div> <div>✗ 2.10.1</div> <div>✗ 2.10.0</div> <div>✗ 2.9.29</div> </div>	Update	Up to Date (Updating again will check for purchased Software Features.)

Firmware update panel

To update the firmware of a connected Acroname device, in the lower-right panel, click “Firmware Management” to bring up the firmware update panel. A list of connected devices with serial number, currently installed firmware version, target version, and an update button should appear. Select target firmware version, and click *update* to update the firmware. If the device isn’t visible, check that it is connected and click *discover*.

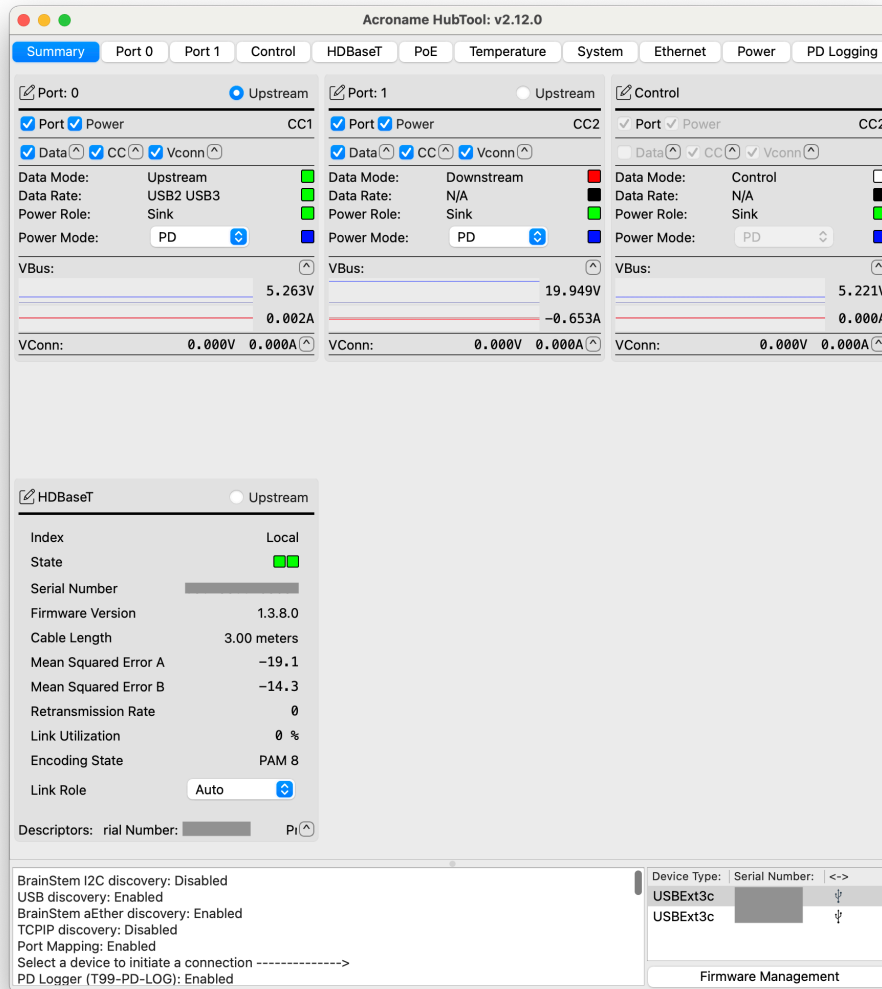
Firmware update is required to enable newly purchased software add-on features, even if the installed version is current.

See the [Firmware Management](#) documentation for more information on updating device firmware.

2.1.9 Device-specific interfaces

Each Acroname product has a distinct interface in HubTool. Select your product below:

USBExt3c



⁴⁹ HubTool interface

for USBExt3c

USBExt3c⁵⁰ is a two-port, software-managed USB Hub and extender that supports:

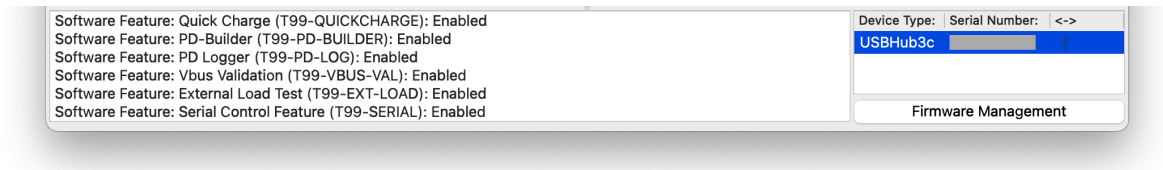
- USB 3.2 Gen 1 (5 Gbps) and USB 2.0 extension up to 100 m over Cat 6A cable (HDBaseT-USB3)
- Local USB links up to 10 Gbps (USB 3.2 Gen 2)
- Bidirectional PoE++ power extension
- Automatic host switching from either end of the extension

HubTool presents a unified dashboard to control and view the state of USBExt3c.

⁴⁹ <https://acroname.com/store/s152>

⁵⁰ <https://acroname.com/store/s152>

Add-on software features



Add-on features listed in HubTool console after selecting the USBExt3c on the right

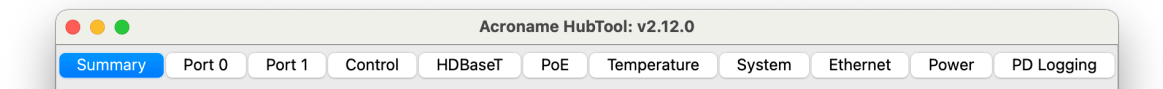
USBExt3c supports add-on software features, which can be purchased to enable new capabilities. When a USBExt3c is selected in HubTool, the console will list the available add-on software features and show their enabled / disabled status on the device.

Table 2: Add-on licensed software features

Feature	Capability
PD builder ⁵¹	Edit Power Data Objects (PDOs) on each port to emulate any USB-PD configuration
PD log-ger ⁵²	Log USB-PD communications on all ports
VBus vali-dation ⁵³	Override normal V_{BUS} voltage set points and current limits. Used in testing a device's response to incorrect V_{BUS} voltages after USB-PD negotiation.
Serial con-trol ⁵⁴	Enable RS-232 serial control of the USBExt3c

Licenses for additional features are available for purchase⁵⁵. After purchase, update the hub firmware to enable the new features.

Dashboard tabs



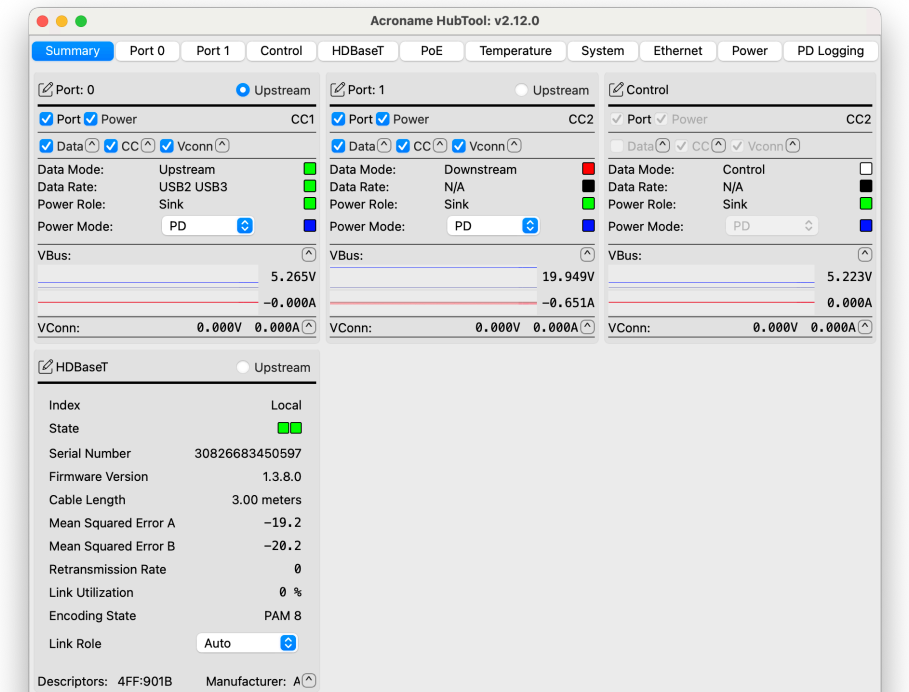
HubTool USBExt3c dashboard tab view

After selecting USBExt3c, HubTool will launch the device dashboard in summary view. Buttons along the top of the window represent tabs for:

⁵¹ <https://acroname.com/store/t99-pd-builder>
⁵² <https://acroname.com/store/t99-pd-log>
⁵³ <https://acroname.com/store/t99-vbus-val>
⁵⁴ <https://acroname.com/store/t99-ext-load>
⁵⁵ https://acroname.com/store-grid/field_category/Software-Licenses

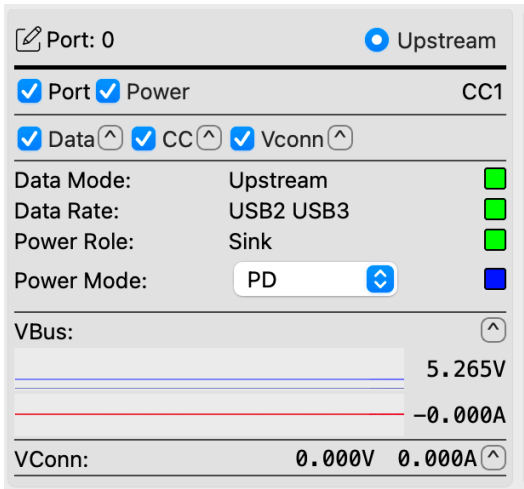
Summary Tab

The summary tab displays simplified interface panels for ports 0, 1, Control, and the HDBaseT-USB3 interface.



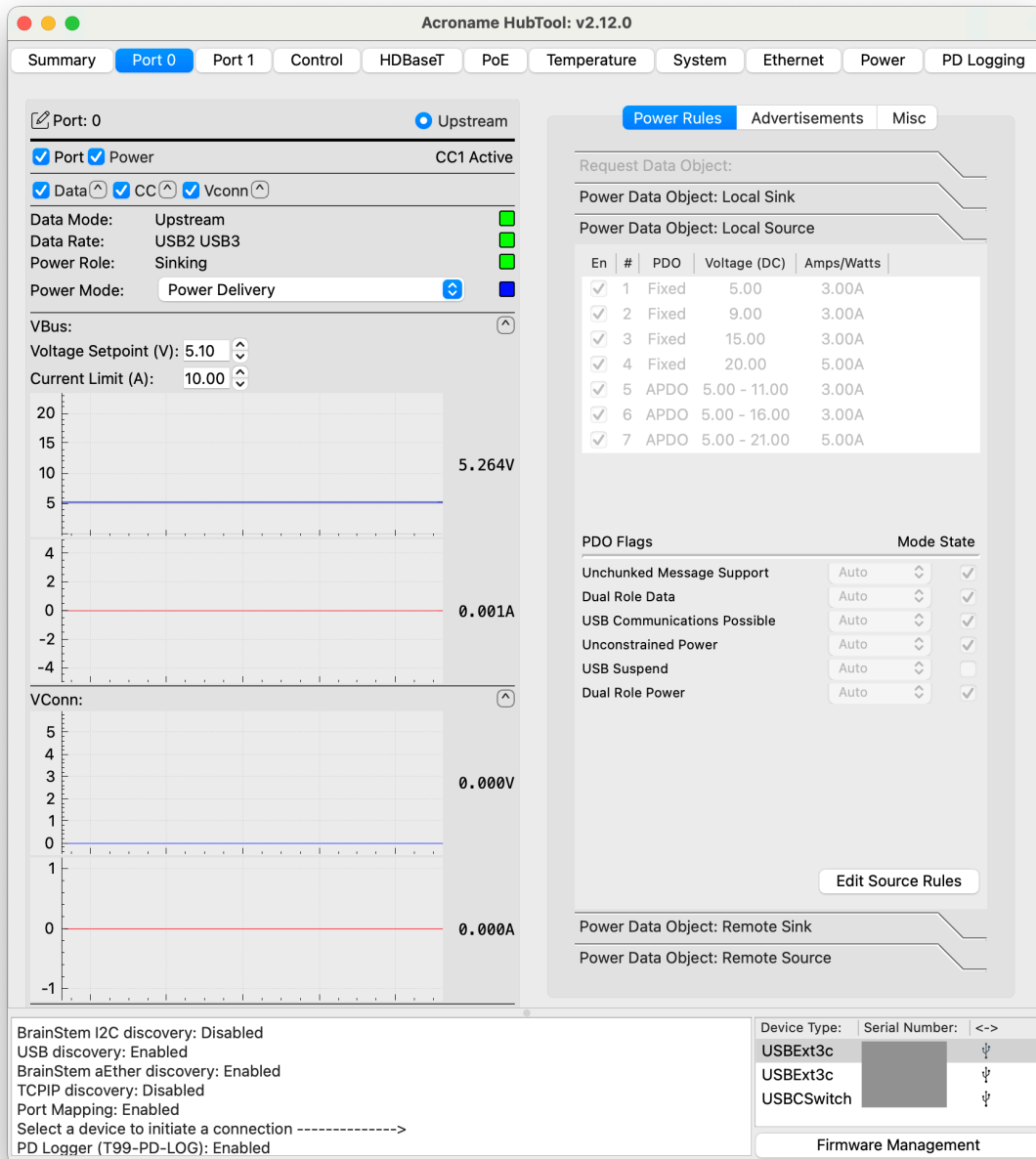
Summary tab view

Each port interface panel shows a consolidated view of the corresponding *port* and *HDBaseT* tabs.



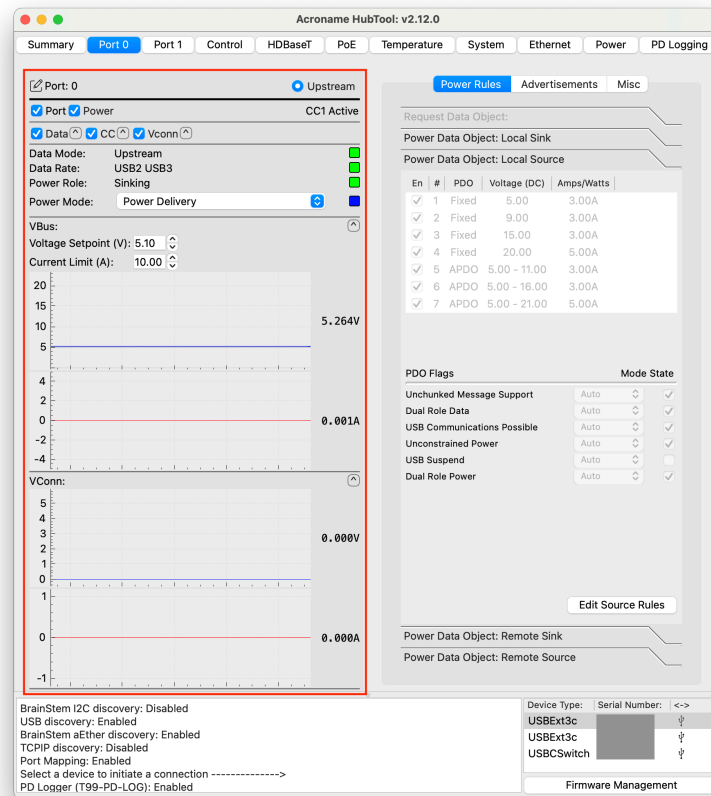
Port panel in summary view

Port Tabs




The *port tabs* (0, 1, and Control) show a more detailed view of each port. The left side of the port tab contains *Power and data settings*. The right side of the port tab views contains sub tabs for USB PD *Power rules*, *Advertisements*, and *Miscellaneous messages and settings*.

Port Tab: Power and Data Settings



Port tab view - power and data settings

The left side of each port tab view contains power and data settings for the port. At the top are the *port name* and *upstream toggle*.

Port name (editable ) - friendly port name used by HubTool and ControlRoom)

Upstream toggle - indicates and selects which port is in upstream mode and able to connect to a host. Note that the Control port can not be set to upstream.

Power and data toggles


Below the port name are the power and data toggles. Click the carats () to expand the view. Clicking the Data carat again cycles through views.



Table 3: Power and data toggles

Toggle name		Enables and disables
Port		Port data and power
Power		V_{BUS}
Data		All data
1x	HS	USB 2 (High Speed) pins.
	SS	USB 3 (SuperSpeed) pins
2x	HSA, HSB	HS pins side A and B
	SSA, SSB	SS pins side A and B
3x	HS Routing:	
	Follow CC	Normal operation
	Side 1	Force Side A
	Side 2	Force Side B
	Shorted	HS data on both sides
	SS Routing:	
	Follow CC	Normal operation
	Side 1	Force Side A
	Side 2	Force Side B
CC		Both CC pins
	CC1, CC2	CC pins individually
Vconn		V_{CONN} pins
	Vconn1, Vconn2	V_{CONN} pins individually

Data and power status, power mode

Below the toggles are *data mode*, *data rate*, and *power role* status indicators, followed by the *power mode* menu. The virtual LED color corresponds to the color of the real LED indicators on the USBExt3c front panel.














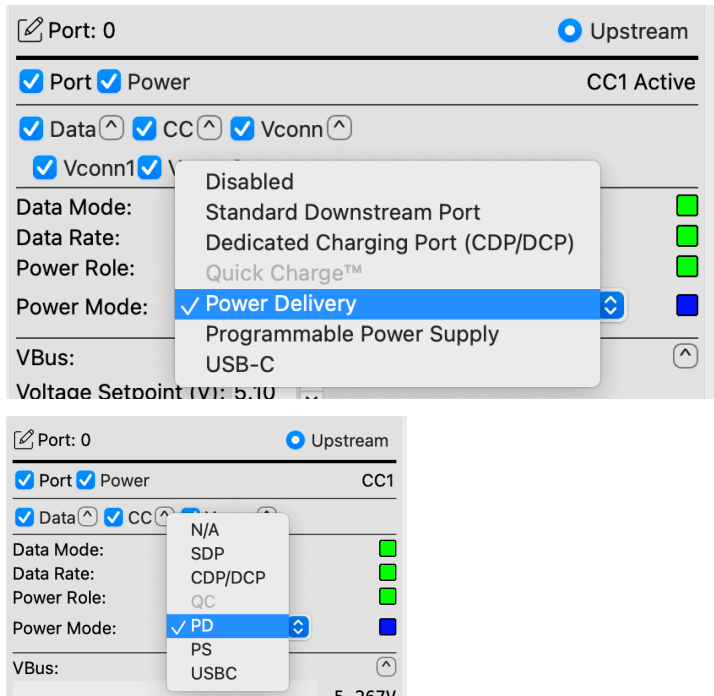
Data Mode:	Upstream	
Data Rate:	USB2 USB3 SuperSpeed+	
Power Role:	Sourcing	
Power Mode:	Power Delivery 	

Table 4: Data and power status








Name	Options	Virtual LED color
Data mode	Upstream (to host)	 Green
	Downstream	 Red
	Control port	 White
Data rate (highest speed)	USB 2	 Yellow
	USB 3 (5 Gb/s)	 Green
	USB 3 SuperSpeed+ (10 Gb/s)	 Blue
Power role	Sinking (receives power)	 Green
	Sourcing (provides power)	 Red

Port power mode is selected from a dropdown menu. In the summary view, the modes are abbreviated.



Power mode selection in port view (left) and summary view (right)

Table 5: Power modes

Summary abbreviation	Power mode	Definition	Virtual LED
N/A	Disabled	Power off	 Black
SDP	Standard Downstream Port	5 V, 900 mA	 Red
CDP/DCP	Dedicated Charging Port (CDP/DCP)	5 V, 3 A	 Red
QC	Quick Charge™	Qualcomm® Quick Charge™ 2.0 and 3.0 fast charging, requires Quick Charge feature license 3 A	 Green
PD	Power Delivery	USB-PD revision 2.0 and 3.2 compliant PD fixed voltage 5 - 20 V up to 5 A, USB PD-PPS and QC 4+ variable voltage up to 5 - 21 V, 5 A	 Blue
PS	Programmable Power Supply	Manually set 2.8 V – 21 V and 0 A – 5 A, requires VBus validation feature license	 White
USBC	USB-C	5 V, 3 A	 Magenta

In USB-C and PD modes, V_{BUS} is only enabled after strapping resistors are detected on the CC lines, while SDP, CDP/DCP, PS, and QC modes provide V_{BUS} continuously. In CDP and SDP modes, the port acts as a source if downward facing, and a sink if upward facing.

V_{BUS} voltage and current control

(Requires VBus validation add-on⁵⁶)

Power Mode:

Programmable Power Supply

VBus:

Voltage Setpoint (V):
11.90

Current Limit (A):
1.25

Manually set VBus voltage and current

When **Programmable Power Supply (PS)** mode is enabled, V_{BUS} voltage setpoint and current limit can be set directly, transforming the port into a programmable power supply capable of supplying up to 100 Watts. These are saveable settings (System tab / Save) that persist through reset.

⁵⁶ https://acroname.com/reference/devices/usbhub3c/software_features/vbus_validation.html

In **Power Delivery (PD)** mode, the user-set values override the negotiated values. However, PD negotiations are still active and PD events or errors can trigger re-negotiation, which will replace the user-set values. These values should only be changed when the power role is set to *sourcing*.

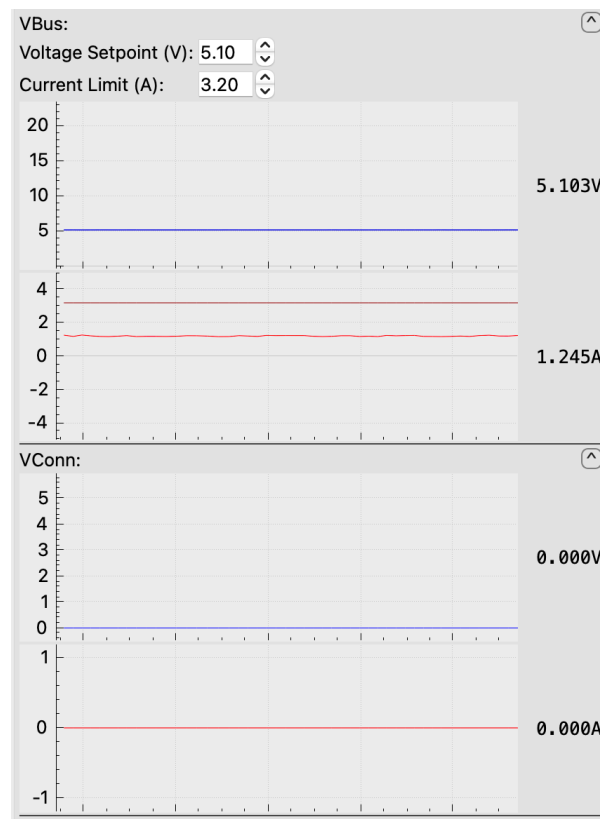
In **USB-C, SDP, CDP/DCP** modes, VBus settings revert to 5 V and default current on port connect or disconnect.

Setting	Value
Voltage setpoint	2.8 V – 21 V
Current limit	0 A – 5 A

Warning: Can damage attached equipment!

Voltage and current display

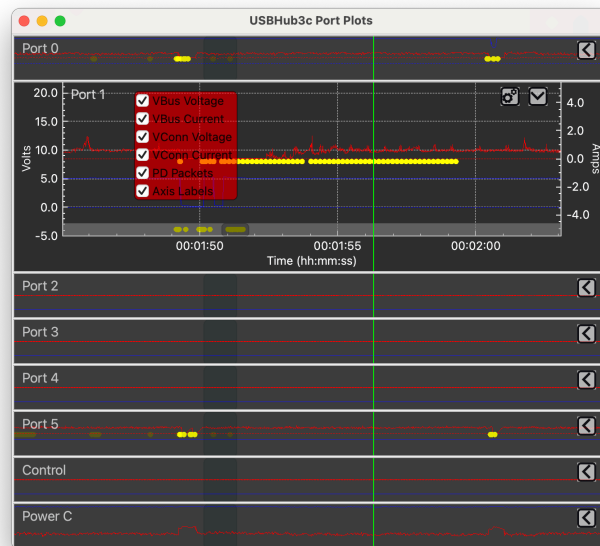
Shows a graph of the VBus and VConn voltage and current of the port.



VBus and VConn panel view

Clicking on the port graph pops up the Port Plots window with a larger rolling strip chart showing all ports. Click the carat (⌵) to expand each port chart. If *PD logging* is enabled, PD messages will be marked in yellow.

Port Plots window



Left vertical axis – voltage (V)

Right vertical axis – current (A)

Horizontal axis – time (s)

By default, the plot will be scrolling. To stop scrolling, drag the dark gray scroll bar to the left. To enable scroll-to-zoom, click the desired axis.

Device descriptors

If Options > Port Mapping is selected, when a device is attached to a downstream port, its descriptors will scroll at the bottom of the port panel. Click the carat (⤴) to expand:

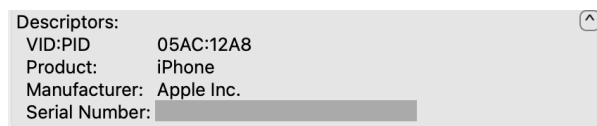
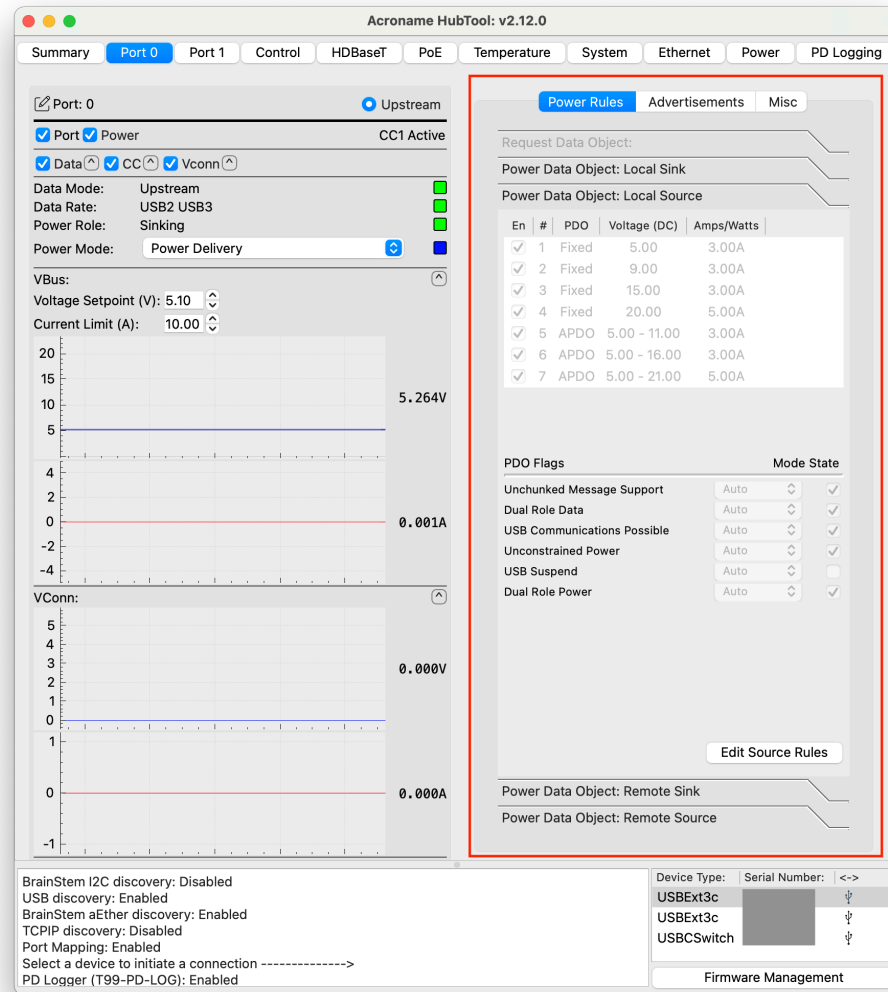


Table 6: *Descriptor table*

Descriptor	Content
VID:PID	16-bit vendor ID and 16-bit product ID
Product	Product name string
Manufacturer	Manufacturer name string
Product serial number	Product serial number
ATT	Indicates device is attached

Port Tab: Power rules

The right side of the port tab views contains sub-tabs for USB-PD power rules, advertisements, and miscellaneous messages and settings. The power rules section presents a set of vertical tabs on the right side of the port view:



Power rules vertical tabs

The vertical tabs are:

- *Request Data Object*
- *Power Data Object: Local Sink*
- *Power Data Object: Local Source*
- *Power Data Object: Remote Sink*
- *Power Data Object: Remote Source*

Request Data Object: Local [Source or Sink]

A Request Data Object (RDO) tells the host about the sink's capabilities and selects one of the Power Data Objects (PDOs) offered by the source by specifying the PDO's index (1-7).

When the port is configured as a sink:

- Tab label shows "Local Sink"
- Panel displays the RDO sent from the hub (sink) to the source device

When the port is configured as a source:

- Tab label shows "Local Source"
- Panel displays the RDO received by the hub (source) from the sinking device (informational only)

The screenshot shows a software window with three tabs: "Power Rules" (selected), "Advertisements", and "Misc". The main panel is titled "Request Data Object: Local Source". It contains a table with two columns: "RDO Information" and "State".

RDO Information	State
Unchunked Message Support	<input type="checkbox"/>
USB Suspend	<input type="checkbox"/>
No USB Communications Possible	<input checked="" type="checkbox"/>
Capability Mismatch	<input type="checkbox"/>
Giveback	<input type="checkbox"/>
Maximum Operate Current	3000mA
Operating Current	3000mA
Object Position (index)	1
Raw	0x1204B12C

Below the table, there is a section titled "Power Data Object: Local Sink". At the bottom of the window, there are fields for "Device Type:", "Serial Number:", and a "<->" button.

Power Rules Advertisements Misc

Request Data Object: Local Sink

RDO Information	State
Unchunked Message Support	<input checked="" type="checkbox"/>
USB Suspend	<input checked="" type="checkbox"/>
No USB Communications Possible	<input checked="" type="checkbox"/>
Capability Mismatch	<input type="checkbox"/>
Giveback	<input type="checkbox"/>
Maximum Operate Current	1500mA <input type="button" value="up"/> <input type="button" value="down"/>
Operating Current	1500mA <input type="button" value="up"/> <input type="button" value="down"/>
Object Position (index)	1 <input type="button" value="up"/> <input type="button" value="down"/>
Raw	0x13825896

Power Data Object: Local Sink

Power Data Object: Local Source

Power Data Object: Remote Sink

Power Data Object: Remote Source -Active- Rule: 1

RDO local source (left) and local sink (right) views

Table 7: Request Data Object tab options

RDO Information	Meaning
Unchunked message support	Checked = unchunked mode; supports larger data payloads Unchecked = legacy “chunked” mode; leave unchecked for maximum compatibility
USB suspend	Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
No USB communications possible	Device can not communicate over USB
Capability mismatch	Source PDO modes are not sufficient for device to operate at full capability
Giveback	Device can go to a lower power state if needed
Maximum operate current	Maximum current the sinking device is capable of drawing
Operating current	Nominal operating current of the sinking device
Object position (index)	Index of PDO requested, range 1-7
Raw	The RDO message expressed as 32-bit hexadecimal value

Power Data Object: Local Sink

This tab shows the settings for the port's local sink Power Data Object (PDO). This is how the port tells a remote source about its capabilities and how much power it can accept. If the port is not operating as a sink, these settings have no effect. At the top of the panel is a table of PDO modes that the port can accept as a sink.

Table 8: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Power Rules

Advertisements

Misc

Request Data Object: Local Sink

Power Data Object: Local Sink

En	#	PDO	Voltage (DC)	Amps/Watts
<input checked="" type="checkbox"/>	1	Fixed	5.00	3.00A
<input checked="" type="checkbox"/>	2	Battery	3.30 - 21.00	100.00W
<input checked="" type="checkbox"/>	3	Variable	3.30 - 21.00	5.00A
<input checked="" type="checkbox"/>	4	APDO	3.30 - 21.00	5.00A
<input type="checkbox"/>	5	Fixed	0.00	0.00A
<input type="checkbox"/>	6	Fixed	0.00	0.00A
<input type="checkbox"/>	7	Fixed	0.00	0.00A

PDO Flags

Mode State

Dual Role Data	Auto	<input checked="" type="checkbox"/>
USB Communications Possible	Auto	<input checked="" type="checkbox"/>
Unconstrained Power	Auto	<input type="checkbox"/>
Higher Capability	Auto	<input checked="" type="checkbox"/>
Dual Role Power	Auto	<input checked="" type="checkbox"/>

Edit Sink Rules

Power Data Object: Local Source

Power Data Object: Remote Sink

Power Data Object: Remote Source -Active- Rule: 1

Local Sink PDO list and flags

Table 9: *PDO flags*

PDO flag	Mode	State	Meaning
Dual Role Data USB Communications Possible	En- able, Dis- able, Auto	Checker if flag is en- abled	Can be host or peripheral Supports USB data
Unconstrained Power Higher Capability Dual Role Power			AC powered or large battery Sink needs more than 5 V for full functionality Can source or sink power

Edit sink rules button (requires PD Builder license)

Pops up [Power Rule Editor](#).

Power Data Object: Local Source

This tab shows the settings for the port's local source Power Data Object (PDO). If the port is not operating as a source, these settings have no effect. At the top is a table of PDO modes that the port can provide as a source. If active, the PDO rule index is listed in the tab title.

Table 10: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Power Rules Advertisements Misc

Request Data Object: Local Source

Power Data Object: Local Sink

Power Data Object: Local Source -Active- Rule: 4

En	#	PDO	Voltage (DC)	Amps/Watts
<input checked="" type="checkbox"/>	1	Fixed	5.00	3.00A
<input checked="" type="checkbox"/>	2	Fixed	9.00	3.00A
<input checked="" type="checkbox"/>	3	Fixed	15.00	3.00A
<input checked="" type="checkbox"/>	4	Fixed	20.00	5.00A
<input checked="" type="checkbox"/>	5	APDO	5.00 - 11.00	3.00A
<input checked="" type="checkbox"/>	6	APDO	5.00 - 16.00	3.00A
<input checked="" type="checkbox"/>	7	APDO	5.00 - 21.00	5.00A

PDO Flags

	Mode	State
Unchunked Message Support	Auto	<input checked="" type="checkbox"/>
Dual Role Data	Auto	<input checked="" type="checkbox"/>
USB Communications Possible	Auto	<input checked="" type="checkbox"/>
Unconstrained Power	Auto	<input checked="" type="checkbox"/>
USB Suspend	Auto	<input type="checkbox"/>
Dual Role Power	Auto	<input checked="" type="checkbox"/>

RDO

	State
Maximum Operate Current	4700mA
Operating Current	3750mA
Object Position (index)	4
Raw	0x4785DDD6

Edit Source Rules

Power Data Object: Remote Sink

Power Data Object: Remote Source

Local source PDO list and flags

Table 11: PDO flags

PDO flag	Mode	State	Meaning
Unchunked Message Support	En-able, Dis-able, Auto	Checked if flag is en-abled	Checked = unchunked mode; supports larger data payloads Unchecked = legacy “chunked” mode; leave unchecked for maximum compatibility
Dual Role Data			Can be host or peripheral
USB Communications Possible			Supports USB data
Unconstrained Power			AC powered or large battery
USB Suspend			Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
Dual Role Power			Can source or sink power

Edit source rule button (requires PD Builder license)

Pops up *Power Rule Editor*.

Power Data Object: Remote Sink

This tab shows the power sink settings for the remote device's Power Data Object (PDO). Since these settings are controlled by the remote device, this tab is informational only. At the top is a list of PDO modes that the remote device can accept as a sink.

Table 12: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Power Rules
Advertisements
Misc

Request Data Object: Local Source
Power Data Object: Local Sink
Power Data Object: Local Source -Active- Rule: 4
Power Data Object: Remote Sink

#	PDO	Voltage (DC)	Amps/Watts
1	Fixed	5.00	3.00A
2	Variable	4.75 - 21.00	4.70A

PDO Flags

Dual Role Data
USB Communications Possible
Unconstrained Power
Higher Capability
Dual Role Power

State
☒
☒
☐
☐
☒

Request Minimum
Request Update

Power Data Object: Remote Source

Remote sink PDO list and flags

Table 13: *PDO flags (informational only)*

PDO flag	State	Meaning
Dual Role Data	Checked if flag is enabled	Can be host or peripheral
USB Communications Possible		Supports USB data
Unconstrained Power		AC powered or large battery
Higher Capability		Sink needs more than 5 V for full functionality
Dual Role Power		Can source or sink power

Request minimum, Request update buttons

[Not implemented]

Power Data Object: Remote Source

This tab shows settings for the remote device's Power Data Object (PDO) and its Request Data Object if active. Since the PDO settings are controlled by the remote device, the PDO mode list and flags are informational only. At the top is a list of PDO modes that the remote device can provide as a source. If the remote device is acting as a source, the active PDO rule index is listed in the tab title.

Power RulesAdvertisementsMisc

Request Data Object: Local Source

Power Data Object: Local Sink

Power Data Object: Local Source -Active- Rule: 1

Power Data Object: Remote Sink

Power Data Object: Remote Source

#	PDO	Voltage (DC)	Amps/Watts
1	Fixed	5.00	1.50A

PDO FlagsState

Unchunked Message Support

Dual Role Data

USB Communications Possible

Unconstrained Power

USB Suspend

Dual Role Power

☒

☒

☒

☐

☐

☒

Request Update

Power Rules
Advertisements
Misc

Request Data Object: Local Sink
Power Data Object: Local Sink
Power Data Object: Local Source
Power Data Object: Remote Sink
Power Data Object: Remote Source -Active- Rule: 1

#	PDO	Voltage (DC)	Amps/Watts
1	Fixed	5.00	1.50A

PDO Flags

Unchunked Message Support

Dual Role Data

USB Communications Possible

Unconstrained Power

USB Suspend

Dual Role Power

State

☒
☒
☒
☐
☐
☒

RDO Control

Maximum Operate Current

Operating Current

Object Position (index)

Raw

State

1500mA

1500mA

1

0x13825896

Request Update

Remote source PDO list, flags and RDO control

Table 14: PDO mode list column headings

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Table 15: Remote source PDO flags (informational only)

PDO flag	State	Meaning
Unchunked Message Support	Checked if flag is enabled	Checked = unchunked mode; supports larger data payloads Unchecked = legacy “chunked” mode; leave unchecked for maximum compatibility
Dual Role Data		Can be host or peripheral
USB Communications Possible		Supports USB data
Unconstrained Power		AC powered or large battery
USB Suspend		Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
Dual Role Power		Can source or sink power

RDO control

When the hub port is acting as a sink, some sink RDO settings can be controlled directly from the bottom of the Remote Source tab. These settings mirror the fields in the *Request Data Object: Local Sink* tab.

Table 16: Remote source RDO control settings

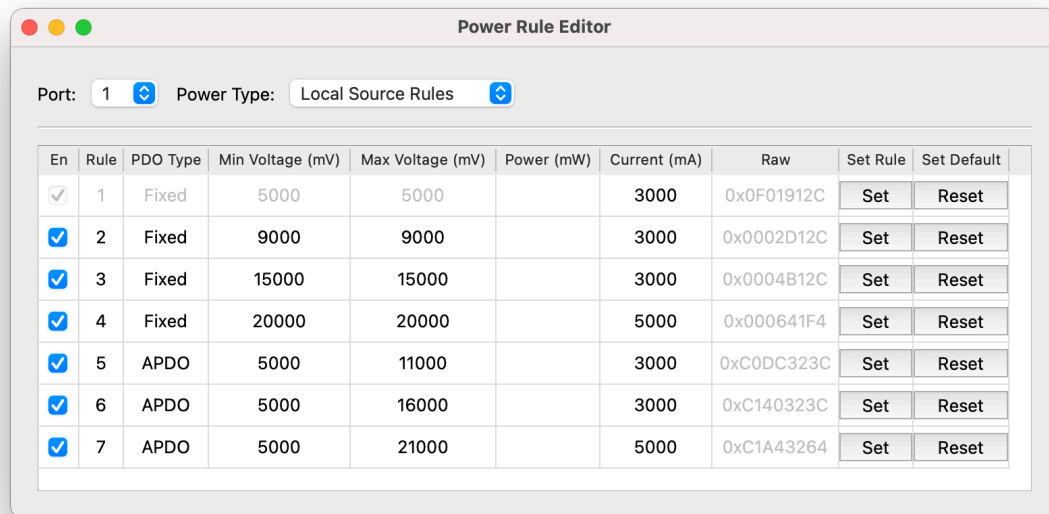
RDO Information	Meaning
Maximum operate current	Maximum current the sinking device is capable of drawing
Operating current	Nominal operating current of the sinking device
Object position (index)	Index of PDO requested, range 1-7
Raw	The RDO message expressed as 32-bit hexadecimal value

Request update button

[Not implemented]

Power rule editor (requires PD builder add-on feature⁵⁷)

⁵⁷ <https://acroname.com/store/t99-pd-builder>



The power rule editor sets sink and source power rules for each port.

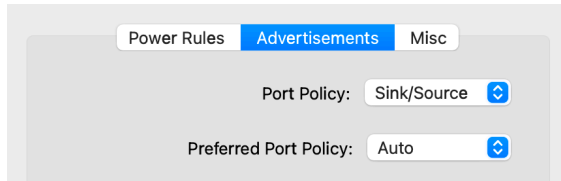
Table 17: *Power rule editor headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
Rule	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Min Voltage (mV)	Min PDO voltage
Max Voltage (mV)	Max PDO voltage
Power (mW)	Max PDO power
Current (mA)	Max PDO current
Raw	Raw bytes of PDO message
Set rule	Set rule after changes
Set default	Return rule to default

Table 18: *PDO Types*

PDO type	Definition
Fixed	Fixed supply voltage of 5 V, 9 V, 15 V, 20 V, current limit 0 A – 5 A. Source must have at least one fixed PDO at 5 V
Variable	Represents a poorly regulated supply. The voltage must stay between the minimum and maximum voltage; minimum voltage must be $\geq 80\%$ of maximum voltage
Battery	Batteries can be connected to VBus directly as a source. Specifies maximum and minimum voltage (in 50 mV steps) and maximum power (in 250 mW steps)
APDO	Programmatically controllable source. Voltage range set in 20 mV steps, up to 3.3V – 21V. Current limit 0 A – 5 A

Port Tab: Advertisements



Power role advertisements subtab

USB-PD advertisements tell connected devices the power policy of the port and its preferred policy.

Port policy

- **None** - no power
- **Source** - provides source
- **Sink** - receives power
- **Sink / Source** - can provide or receive power according to the *Preferred port policy*

Preferred port policy (Only has an effect if port policy is set to *Sink / Source*)

- **None** - no power
- **Source** - provides power
- **Sink** - receives power
- **Follow data** - provides power if downstream-facing, receives power if upstream-facing
- **Auto** - negotiate with the connected device

Port Tab: Misc

The screenshot shows the 'Misc' subtab of the Port configuration window. It contains three main sections:

- Cable Information:** A table showing various cable properties.

Voltage Maximum:	N/A
Current Maximum:	N/A
Speed Maximum:	N/A
Type:	N/A
Orientation:	CC1
- Send Request:** A dropdown menu set to 'Hard Reset' and a 'Send' button.
- Overrides:** Three unchecked checkboxes: 'Cable Current', 'Port Power Limit', and 'Auto Discovery'.

Misc subtab

The Misc subtab contains *Cable information*, *Send request*, and *Overrides*

Cable information

This screenshot shows the 'Cable Information' panel with populated e-mark data:

Voltage Maximum:	50 VDC
Current Maximum:	5 A
Speed Maximum:	USB 3.2 / USB 4 Gen 2
Type:	Passive (E-marked)
Orientation:	CC1

Below this, the 'Send Request' dropdown is set to 'Manufacturer Info Soppp'.

Cable information panel with e-mark data

Displays power and data capacity for e-marked cables connected to the hub and a device.

- Hub port must be providing V_{CONN} (V_{CONN} strip chart showing 5 V)

- If V_{CONN} strip chart shows 0 V, send “VConn Swap” message to swap whether the hub or device is providing V_{CONN}

To read e-mark without a device, connect both ends of the cable to ports of USBExt3c, with one port set as *sink* or *auto* and the other set to *source* or *auto*. View the cable info panel on the source port.

Send Request

This control sends USB PD requests to the connected device to initiate a connection sequence. There is no guarantee that the request will succeed, and many of these request messages will be unsupported by the connected device.

Send request	Definition
Hard Reset	Reinitialize PD communications and cycle V_{BUS}
Soft Reset	Reinitialize PD communications, but maintains power connection
Data Reset	Cycle USB data connection [unimplemented]
Power Role Swap	Exchange source and sink power roles
Power Fast Role Swap	Fast exchange source and sink power roles
Data Role Swap	Swap upstream and downstream ask roles
VCONN Swap	Swap whether the hub or device is supplying V_{CONN}
Sink Go To Min	Sink goes to minimum power draw
Request Remote Source PDOs	Get PDO from remote source
Request Remote Sink PDOs	Get PDO from remote sink
Request Remote Extended Source Caps	Get extended source capabilities
Request Remote Extended Sink Caps	Get extended sink capabilities
Status	Get status
PPS Status	Get PPS status
Battery Capabilities	Get battery capabilities (Design cap, last full cap)
Battery Status	Get Battery status (state of charge, charging status)
Manufacturer Info Sop	Get device manufacture info
Manufacturer Info Sopp	Get cable manufacturer info (nearest V_{CONN} sourcing port)
Manufacturer Info Soppp	Get cable manufacturer info (opposite from V_{CONN} sourcing port)
Discover Identity Sop	Device type and capabilities
Discover Identity Sopp	Cable type and capabilities (V_{CONN} end)
Discover Identity Soppp	Cable type and capabilities (not V_{CONN} end)
Revision	USB PD revision and version numbers for connected device
Source Info	Source port power capability
Country Codes	Country of origin
Country Info	Country of origin

Overrides

Cable current – overrides the 3 A current limit for cables that don't specify 5 A via e-mark

Port power limit – overrides port power budgeting and allows full power output up to 105 W per port

Warning: Can exceed power supply current limit and brown out the hub, triggering a reset

Auto discovery – overrides the auto discovery feature. When enabled, the hub will only establish a basic power connection and not request vendor information. Used for legacy compatibility and debugging.

HDBaseT Tab

The HDBaseT tab shows the status of the HDBaseT-USB3 connection. The top panel shows data from the local extender on the left and the remote extender on the right.

- Index (local or remote)
- State (Device present, Link role and status)
- Serial Number (of the VS6320)
- VS6320 Firmware Version
- Cable Length (m)
- Mean Squared Error A/B (μ dB)
- Retransmission rate (# messages between retransmission, 0 = no errors)
- Link Utilization (μ %)
- Encoding state (e.g. PAM 8)
- Link role (Auto, Leader, Follower)

Below that is a strip chart that can plot:

- Mean square error (A or B, local, or remote)
- Link Utilization % (local or remote)
- Retransmission rate (local or remote)

Autoscroll - toggles scrolling and the time value (sec) sets the visible buffer size.

Logging - enables storing data beyond the autoscroll duration.

Update - sets the plot update time in mS. Low values make the plot scroll more smoothly.

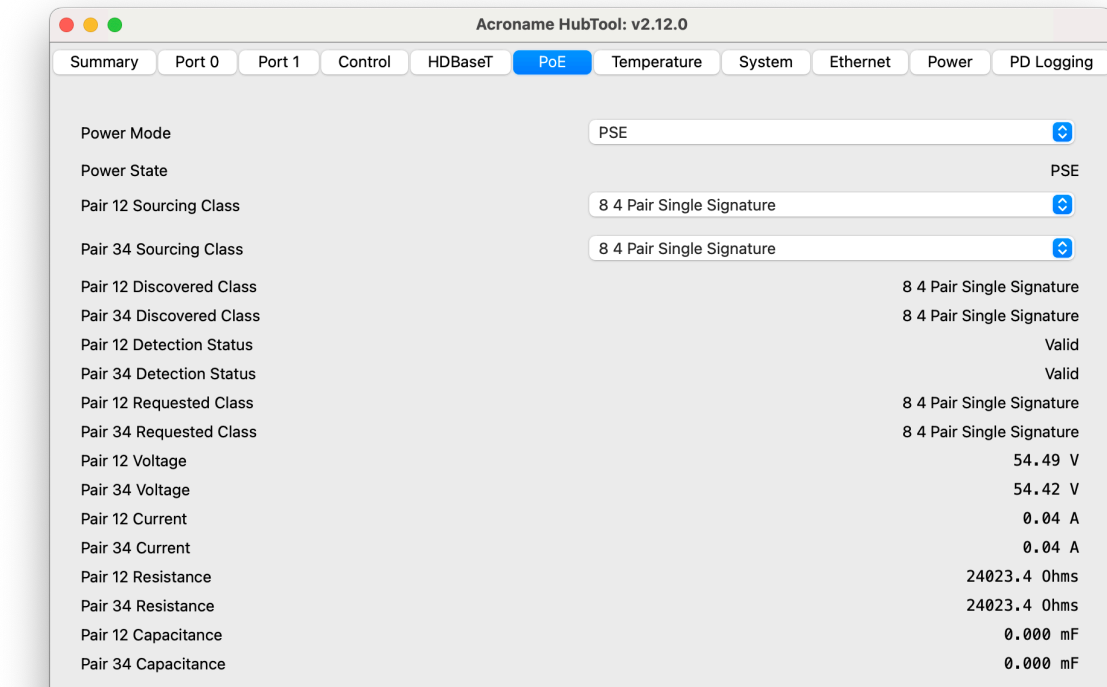
Axis Control - selects the axis to zoom when scrolling with the mouse wheel (Time, Mean squared error, Link utilization, or Transmission rate)

Generate Plot - opens a new plot window

Save CSV -saves the current buffer to one CSV per channel selected



PoE Tab



The **PoE** tab shows the status of the **Power over Ethernet** system on the extension link. The PoE controller automatically manages power distribution between endpoints based on available sources and connected loads. For testing, diagnostics, and advanced applications, **manual control and monitoring** are available.

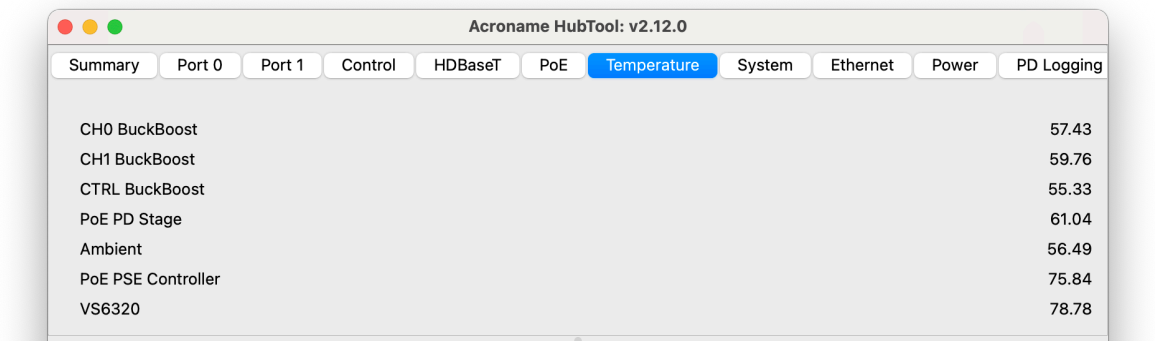
Power Mode / State

- **Power Mode** — sets the device role: **PSE (source)**, **PD (sink)**, **Auto (default)**, or **Off**
- **Power State** — reports the current operating mode (**PSE**, **PD**, or **Off**)

Per-Pair Values (Pairs 1–2 and 3–4)

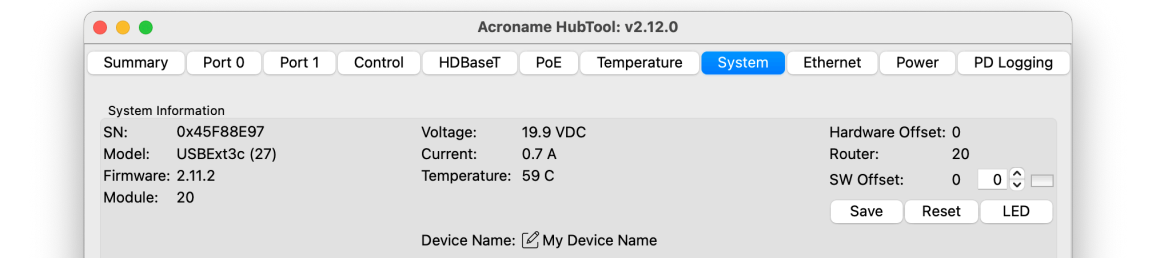
- **Sourcing Class** — sets the PoE source class signature
- **Discovered Class** — shows the negotiated class after detection
- **Detection Status** — indicates connection condition (**valid**, **short**, **open**, **Hi-Z**, **Lo-Z**, **switch failure**, or **unknown**)
- **Requested Class** — class requested from the **PD** to the **PSE**
- **Voltage (V)**
- **Current (A)**
- **Resistance (mΩ)**
- **Capacitance (μF)**

Temperature Tab



The Temperature tab shows temperature readings in Celsius for internal parts of the USBExt3c.

System Tab



HubTool USBExt3c system tab

The system tab contains information and settings for the USBExt3c.

System information

Heading	Meaning
SN	USBExt3c serial number
Model	Model name and number, e.g. USBExt3c (27)
Firmware	Currently installed firmware version
Module	Address of the module on the Brainstem network
Voltage	Voltage at the input power source port
Current	Input current at the input power source port
Temperature	System temperature
Hardware offset	Increments Brainstem module address by a hardware setting - fixed to 0 in USBExt3c
Router	Address of the routing Brainstem module
Software offset	Increment the module address. Click the button to send to the device, Save and Reset to apply
Save	Store select changed settings
Reset	Soft reset of the Extender
LED	Toggles blue User LED (person icon next to the control port), used for debugging and to identify the extender

Ethernet Tab

The screenshot shows the 'Ethernet' tab in the Acroname HubTool v2.12.0 application. The interface includes a top navigation bar with tabs for Summary, Port 0, Port 1, Control, HDBaseT, PoE, Temperature, System, Ethernet (selected), Power, and PD Logging. The main content area is titled 'Network Mode:' and features a 'Static' button with a dropdown arrow. Below this, there are input fields for IPv4 Address (192.168.44.42), IPv4 Netmask (255.255.255.0), IPv4 Gateway (192.168.44.43), and IPv4 DNS (208.67.222.222). A list of DNS servers is shown with a table containing one entry: '1' and '208.67.222.222'. At the bottom, there are fields for Mac Address (04:E6:62:00:00:9A), Hostname (usbext3c-45f88e9), and REST Server Port (0). Buttons for '-', '+', and 'Set' are located near the DNS list.

The Ethernet tab shows network settings for the Ethernet connection. For most setups, we recommend a direct Ethernet link between the host test machine and the USBExt3c.

When using a pair of USBExt3c connected by HDBaseT-USB3, only one Ethernet connection is required; the second device is available through Brainstem networking over the extension link.

By default, the USBExt3c acts as a DHCP client and will receive an IP address from a DHCP server. If no server is detected, the USBExt3c falls back to a static IP address of 192.168.44.42. In static mode, the host computer interface IP must be set to an address in the 192.168.44.x range. The DHCP client is limited to hosts on the local link and does not operate across network bridges or gateways.

Host firewall rules must allow:

- Outgoing UDP multicast on port 9888
- Incoming UDP responses on port 9889
- Outgoing TCP connections to port 8000
- Incoming / Outgoing TCP connections on ports 9005 and 9006

Network Mode - How the IP address is set:

- None - no network
- DHCP (default) - receive IP from DHCP server
- Static, defaults to 192.168.44.42

IPv4 Address - e.g. 192.168.44.42

IPv4 Netmask - 255.255.255.0

IPv4 Gateway - host computer IP

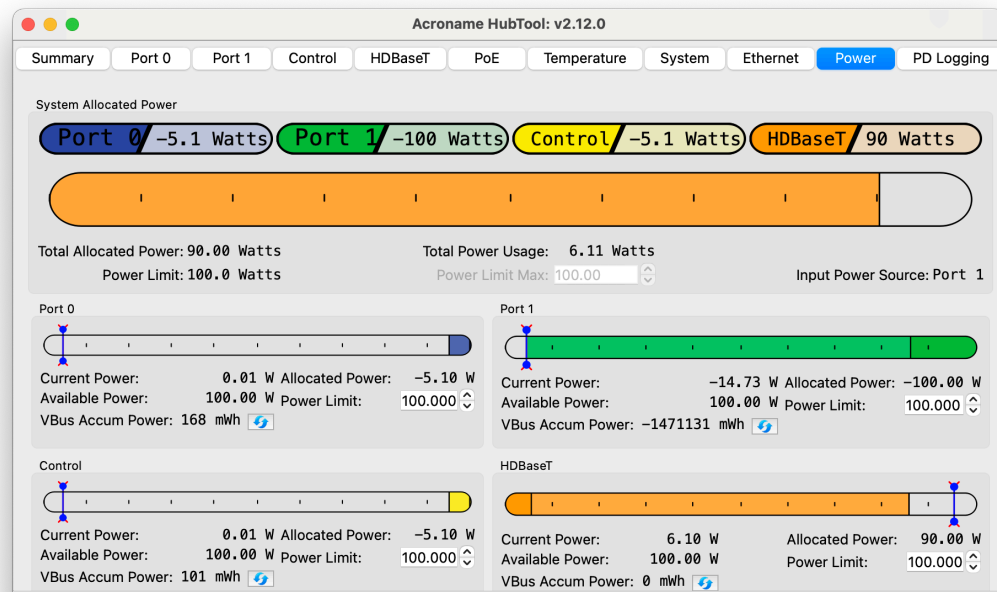
IPv4 DNS - defaults to 208.67.222.222

Mac Address - unique to the specific USBExt3c

Hostname - defaults to "usbext3c-[first 7 digits of hub serial]"

REST Server port - REST port on host

Power Tab

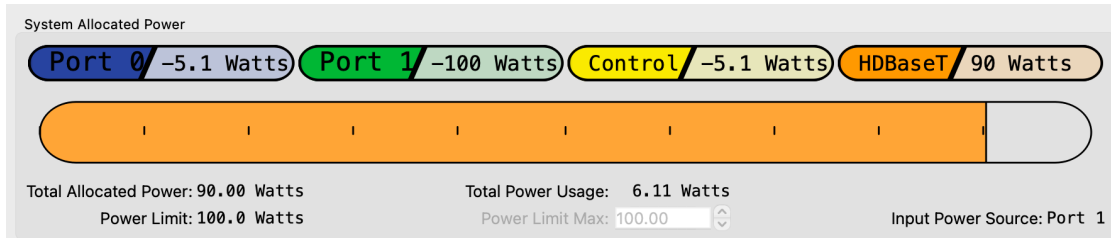


HubTool USBHub3c Power tab

The power tab shows the allocation and actual flow of power through the hub ports and PoE link.

System allocated power panel

Across the top of this panel are the allocated currents for each port. Positive currents represent that the port is a power source, negative currents are for sinks.



System allocated power panel

Below the row of ports is a visualization of the stackup of allocated currents.

Total allocated power - sum of the allocated source power

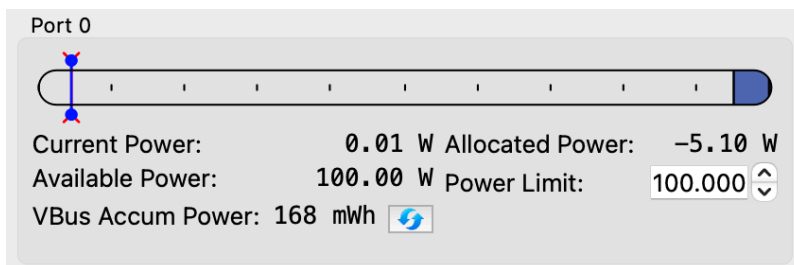
Power limit - maximum power that can be allocated across all ports. Determined by input power minus nominal losses, e.g. 90 W for 100 W supply

Total power usage - sum of actual power being sourced

Power limit max - set power limit for DC input when unregulated power is provided to the DC port

Input power source - input port with the highest power capability

Port power panel



Port power panel

Current power - actual port power output (positive) or input (negative)

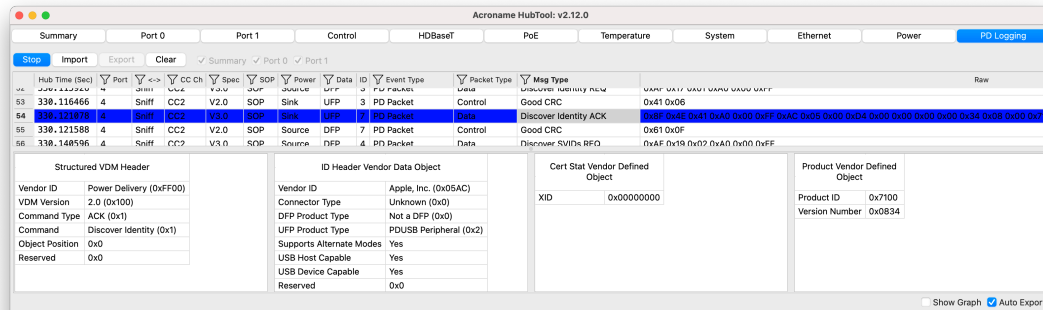
Available power - maximum power that could be allocated to the port, not to exceed *Power limit*

VBus accum power - total energy sunk or sourced on V_{BUS} since reset

Allocated power - power allocated to the port, not to exceed *Power limit*

Power limit - power limit for the port, max 105 W

PD Logging Tab (Add-on feature)



PD logging tab view

PD logging⁵⁸ is a software add-on feature enabling capture, logging, and decoding of USB PD communications on all USB-C ports of the hub, including power negotiations and timing. At the top of the panel are logging controls and port selection toggles.

Stop - stops logging

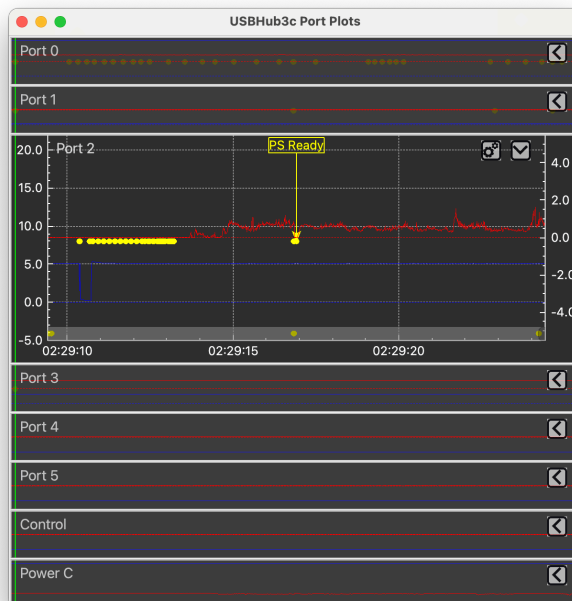
Import - imports a CSV log file

Export - exports a CSV log file (stop logging before export)

Clear - clears the PD log


Port toggles - select which ports to monitor (can only change toggles when logging is stopped)

Show graph - when checked, clicking on an event pops up the *Port Plot* graph window and highlights the corresponding PD packet. Clicking on a yellow PD message in the port plot highlights the corresponding message in the log



⁵⁸ <https://acroname.com/store/t99-pd-log>

Graph view with highlighted PD packet

Below these controls is the PD packet log. The left axis is the row number, which resets when the log is cleared. Columns can be filtered by clicking the filter icon () on the column headings:

Hub Time (Sec)	Port	<->	Spec	SOP	Power	Data	ID	Event Type	Packet Type	Msg Type	Raw
----------------	------	-----	------	-----	-------	------	----	------------	-------------	----------	-----

PD packet log headings

Time - clicking on this heading cycles among time references:

- Hub time (s) - time since the hub powered on
- App time (hh:mm:ss) - time since the HubTool App was launched
- System time (yyyy.MM.dd hh:mm:ss:zzzz) - date and time

Port - ports 0,1, Control (2)

<-> - message direction - *RX*, *TX*, or '-' (none)

Spec - USB PD version

SOP* - "Start Of Packet"

- SOP - for messages between source and sink
- SOP' - for messages to the cable connector closest to the downstream-facing port (DFP)
- SOP" - for messages to the cable connector closest to the upstream-facing port (UFP)

Power - *sink*, *source*, *none*

Data - *UFP*, *DFP*, '-'

ID - message ID, (0 – 7), increments with each new message. Acknowledgements should match message ID

Event type - description of the event

Packet type

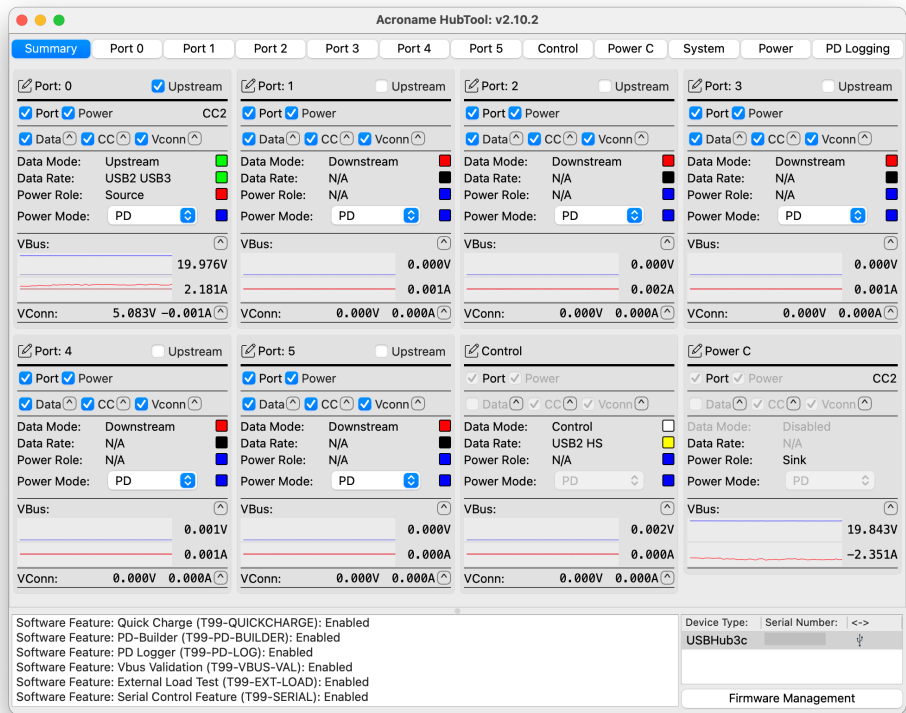
- Control - short messages that typically require no data exchange
- Data - messages contain data objects that are transmitted between devices
- Extended - data messages with larger data payloads

Message type - description of message

Raw - raw PD message (hexadecimal)

Each tab except *Summary* can be dragged out of the main hubtool window into its own window to allow simultaneous viewing of multiple tabs.

USBHub3c



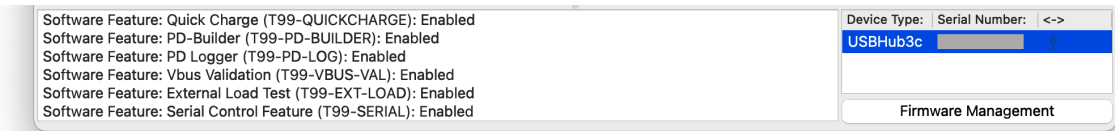
59 HubTool interface

for USBHub3c

USBHub3c⁶⁰ is an industrial USB-C hub featuring 6 full-featured 10 Gbps USB 3.2 data ports, one USB 2.0 control port, and a dedicated PD power input port. Each data port can provide up to 100 W of power and can act as a downstream-facing or upstream-facing port.

HubTool presents a unified dashboard to control and view the state of USBHub3c.

Add-on software features



Add-on features listed in HubTool console after selecting the USBHub3c on the right

USBHub3c supports add-on software features, which can be purchased to enable new capabilities. When a USBHub3c is selected in HubTool, the console will list the available add-on software features and show their enabled / disabled status on the device.

⁵⁹ <https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro>

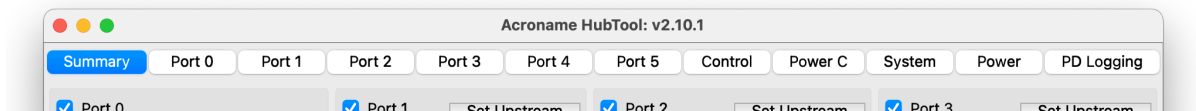
⁶⁰ <https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro>

Table 19: Add-on licensed software features

Feature	Capability
Quick charge ⁶¹	Support Qualcomm Quick Charge® 2 and 3 on ports 0 - 5 (Pro model only)
PD builder ⁶²	Edit Power Data Objects (PDOs) on each port to emulate any USB-PD configuration
PD logger ⁶³	Log USB-PD communications on all ports
VBus validation ⁶⁴	Override normal VBus voltage set points and current limits. Used in testing a device's response to incorrect VBus voltages after USB-PD negotiation.
External load test ⁶⁵	Test VBus sinking with external programmable or resistive electronic loads. Use with External Load Expansion Accessory ⁶⁶
Serial control ⁶⁷	Enable RS-232 serial control of the hub via the serial expansion accessory ⁶⁸

Licenses for additional features are [available for purchase](#)⁶⁹. After purchase, [update the hub firmware](#) to enable the new features.

Dashboard tabs



HubTool USBHub3c dashboard tab view

After selecting USBHub3c, HubTool will launch the device dashboard in summary view. Buttons along the top of the window represent tabs for:

Summary Tab

The summary tab displays simplified interface panels for Ports 0-5, Control port, and Power C port (power input on rear panel).

Summary tab view

Each port interface panel shows a consolidated view of the corresponding [port tab](#).

Port panel in summary view

⁶¹ <https://acroname.com/store/t99-quickcharge>

⁶² <https://acroname.com/store/t99-pd-builder>

⁶³ <https://acroname.com/store/t99-pd-log>

⁶⁴ <https://acroname.com/store/t99-vbus-val>

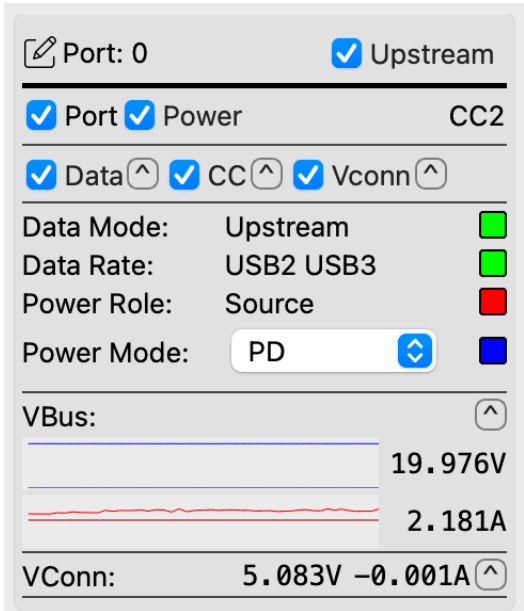
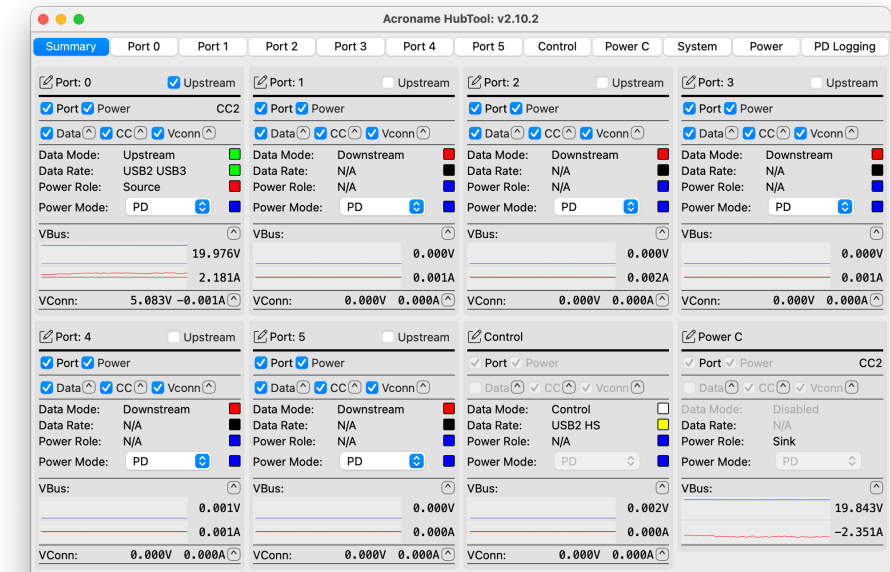
⁶⁵ <https://acroname.com/store/t99-ext-load>

⁶⁶ <https://acroname.com/store/s101-load-exp>

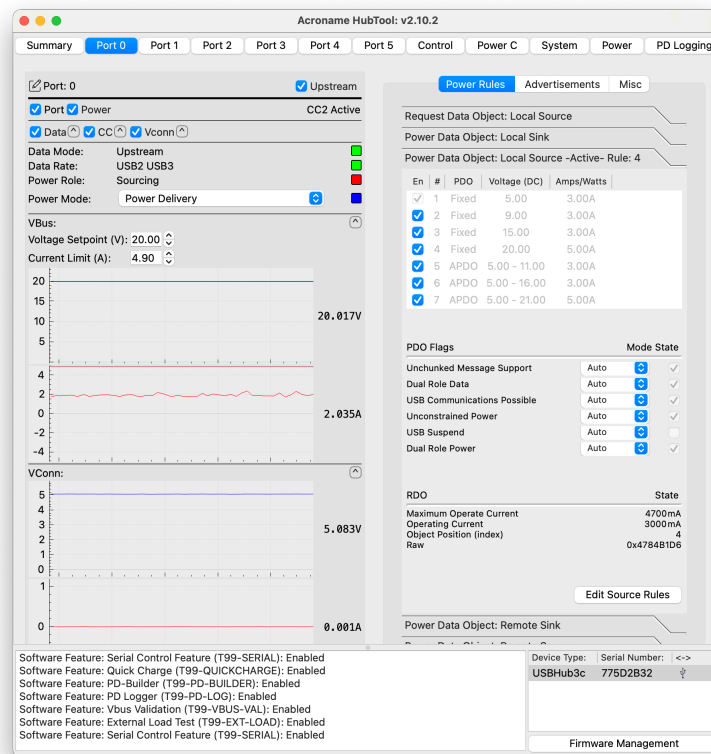
⁶⁷ <https://acroname.com/store/t99-ext-load>

⁶⁸ <https://acroname.com/store/s103-serial-exp>

⁶⁹ https://acroname.com/store-grid/field_category/Software-Licenses



Port Tabs




The port tabs show a more detailed view of each port. Control and Power C ports have specialized functions and some options are unavailable. The left side of the port tab contains *Power and data settings*. The right side of the port tab views contains sub tabs for USB-PD *Power rules*, *Advertisements*, and *Miscellaneous messages and settings*.

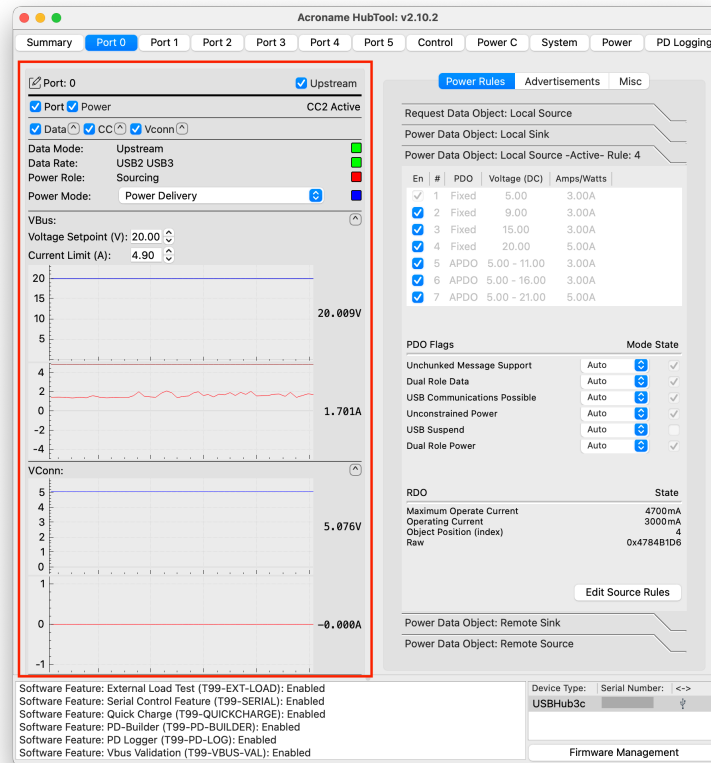
Port Tab: Power and Data Settings

Port tab view - power and data settings

The left side of each port tab view contains power and data settings for the port. At the top are the *port name* and *upstream toggle*.

Port name (editable  - friendly port name used by HubTool and ControlRoom)

Upstream toggle - indicates and selects which port is in upstream mode and able to connect to a host



Power and data toggles


Below the port name are the power and data toggles. Click the carats () to expand the view. Clicking the Data carat again cycles through views.



Table 20: Power and data toggles

Toggle name		Enables and disables
Port		Port data and power
Power		VBus
Data		All data
1x	HS	USB 2 (High Speed) pins
	SS	USB 3 (SuperSpeed) pins
2x	HSA, HSB	HS pins side A and B
	SSA, SSB	SS pins side A and B
3x	HS Routing:	
	Follow CC	Normal operation
	Side 1	Force Side A
	Side 2	Force Side B
	Shorted	HS data on both sides
	SS Routing:	
	Follow CC	Normal operation
	Side 1	Force Side A
	Side 2	Force Side B
CC		Both CC pins
	CC1, CC2	CC pins individually
Vconn		Vconn pins
	Vconn1, Vconn2	Vconn pins individually

Data and power status, power mode

Below the toggles are *data mode*, *data rate*, and *power role* status indicators, followed by the *power mode* menu. The virtual LED color corresponds to the color of the real LED indicators on the USBHub3c front panel.














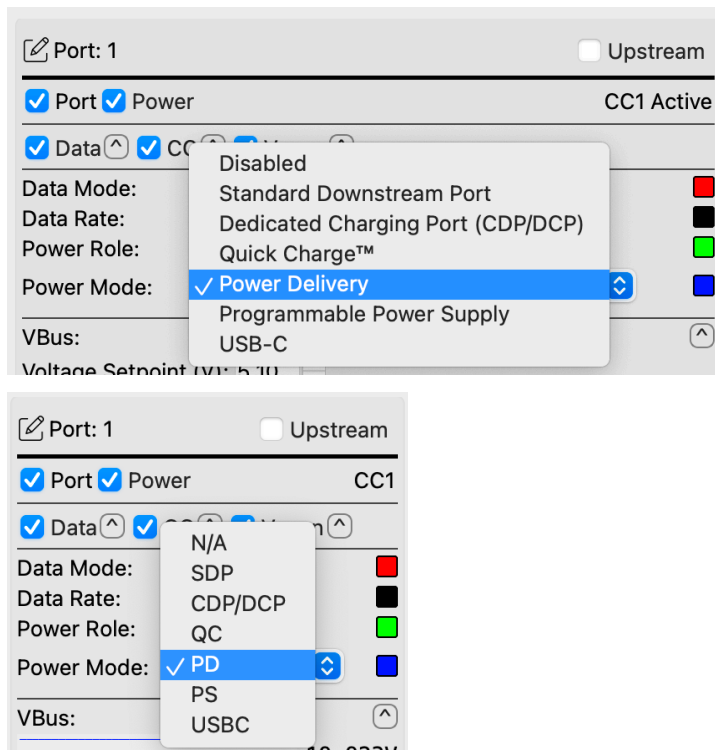
Data Mode:	Upstream	
Data Rate:	USB2 USB3 SuperSpeed+	
Power Role:	Sourcing	
Power Mode:	Power Delivery 	

Table 21: Data and power status








Name	Options	Virtual LED color
Data mode	Upstream (to host)	 Green
	Downstream	 Red
	Control port	 White
Data rate (highest speed)	USB 2	 Yellow
	USB 3 (5 Gb/s)	 Green
	USB 3 SuperSpeed+ (10 Gb/s)	 Blue
Power role	Sinking (receives power)	 Green
	Sourcing (provides power)	 Red

Port power mode is selected from a dropdown menu. In the summary view, the modes are abbreviated.



Power mode selection in port view (left) and summary view (right)

Table 22: Power modes

Summary abbreviation	Power mode	Definition	Virtual LED
N/A	Disabled	Power off	 Black
SDP	Standard Downstream Port	5 V, 900 mA	 Red
CDP/DCP	Dedicated Charging Port (CDP/DCP)	5 V, 3 A	 Red
QC	Quick Charge™	Qualcomm® Quick Charge™ 2.0 and 3.0 fast charging, requires Quick Charge feature license 3 A	 Green
PD	Power Delivery	USB-PD revision 2.0 and 3.2 compliant PD fixed voltage 5 - 20 V up to 5 A, USB PD-PPS and QC 4+ variable voltage up to 5 - 21 V, 5 A	 Blue
PS	Programmable Power Supply	Manually set 2.8 V – 21 V and 0 A – 5 A, requires VBus validation feature license	 White
USBC	USB-C	5 V, 3 A	 Purple

In USB-C and PD modes, the VBus is only enabled after strapping resistors are detected on the CC lines, while SDP, CDP/DCP, PS, and QC modes provide VBus continuously. In CDP and SDP modes, the port acts as a source if downward facing, and a sink if upward facing.

VBus voltage and current control

(Requires [VBus validation add-on](https://acroname.com/reference/devices/usbhub3c/software_features/vbus_validation.html)⁷⁰)

Power Mode:

Programmable Power Supply

VBus:

Voltage Setpoint (V): 11.90

Current Limit (A): 1.25

Manually set VBus voltage and current

When **Programmable Power Supply (PS)** mode is enabled, VBus voltage setpoint and current limit can be set directly, transforming the port into a programmable power supply capable of supplying up to 100 Watts. These are saveable settings (System tab / Save) that persist through reset.

⁷⁰ https://acroname.com/reference/devices/usbhub3c/software_features/vbus_validation.html

In **Power Delivery (PD)** mode, the user-set values override the negotiated values. However, PD negotiations are still active and PD events or errors can trigger re-negotiation, which will replace the user-set values. These values should only be changed when the power role is set to *sourcing*.

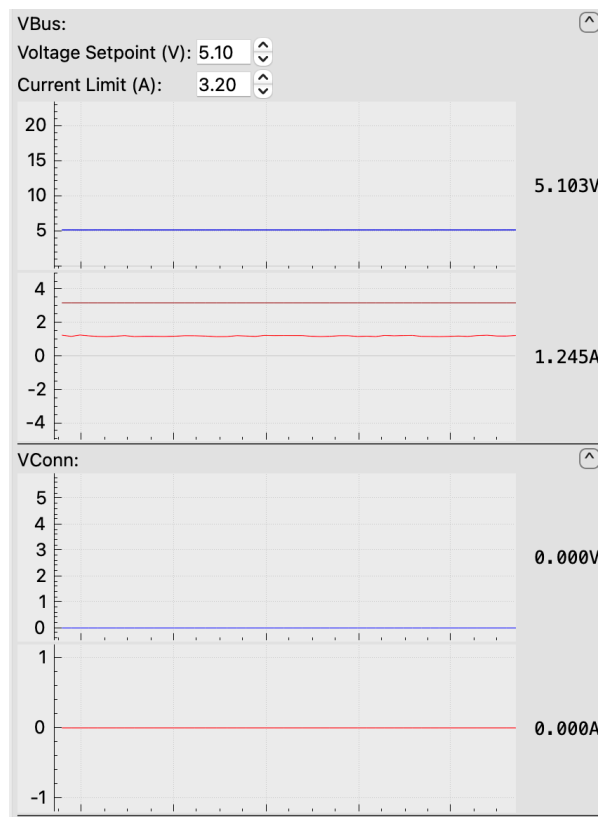
In **USB-C, SDP, CDP/DCP** modes, VBus settings revert to 5 V and default current on port connect or disconnect.

Setting	Value
Voltage setpoint	2.8 V – 21 V
Current limit	0 A – 5 A

Warning: Can damage attached equipment!

Voltage and current display

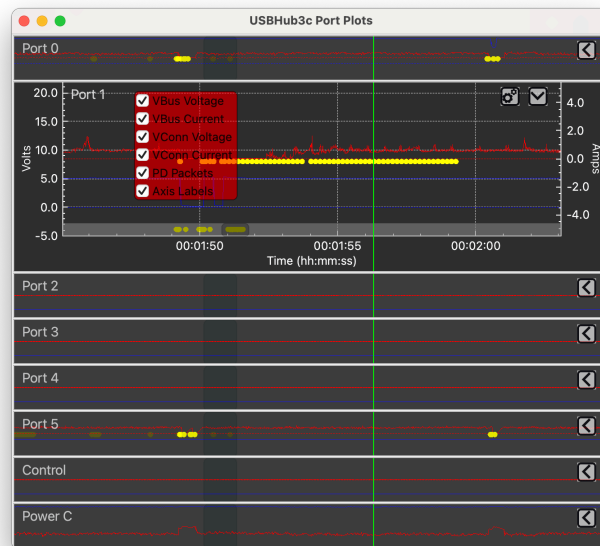
Shows a graph of the VBus and VConn voltage and current of the port.



VBus and VConn panel view

Clicking on the port graph pops up the Port Plots window with a larger rolling strip chart showing all ports. Click the carat (⏏) to expand each port chart. If *PD logging* is enabled, PD messages will be marked in yellow.

Port Plots window



Left vertical axis – voltage (V)

Right vertical axis – current (A)

Horizontal axis – time (s)

By default, the plot will be scrolling. To stop scrolling, drag the dark gray scroll bar to the left. To enable scroll-to-zoom, click the desired axis.

Device descriptors

If Options > Port Mapping is selected, when a device is attached to a downstream port, its descriptors will scroll at the bottom of the port panel. Click the carat (⤴) to expand:

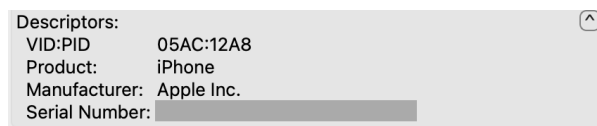
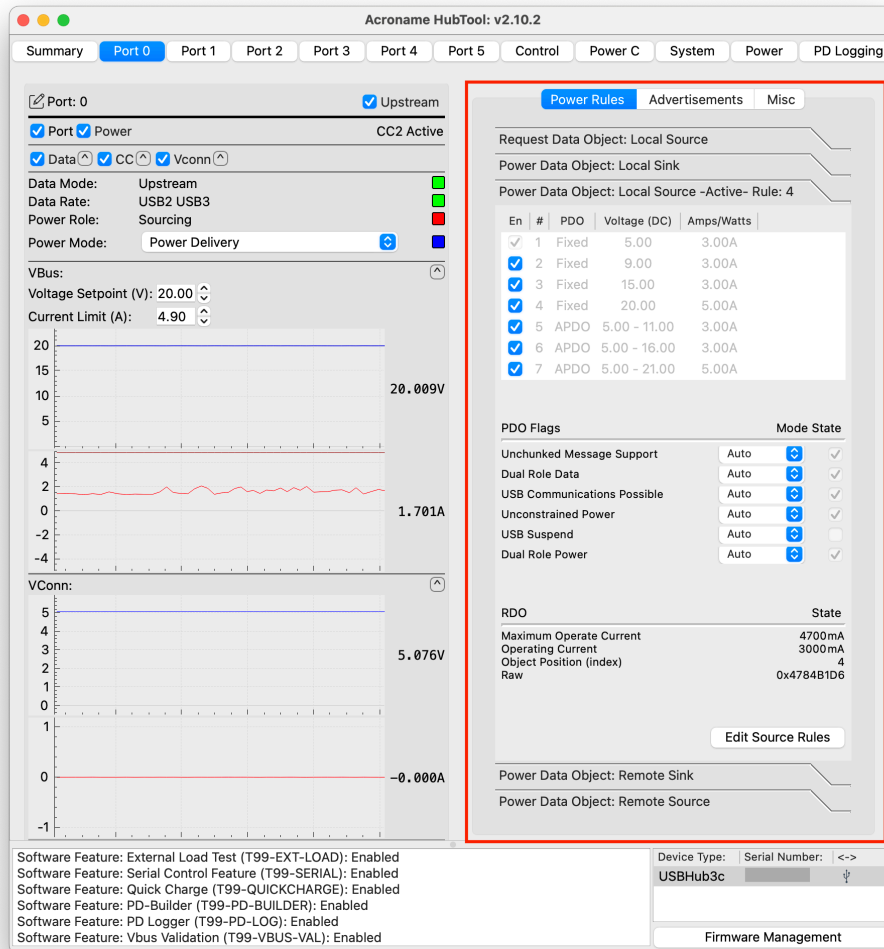


Table 23: *Descriptor table*

Descriptor	Content
VID:PID	16-bit vendor ID and 16-bit product ID
Product	Product name string
Manufacturer	Manufacturer name string
Product serial number	Product serial number
ATT	Indicates device is attached

Port Tab: Power Rules

The right side of the port tab views contains sub-tabs for USB-PD power rules, advertisements, and miscellaneous messages and settings. The power rules section presents a set of vertical tabs on the right side of the port view:



Power rules vertical tabs

The vertical tabs are:

- [Request Data Object](#)
- [Power Data Object: Local Sink](#)
- [Power Data Object: Local Source](#)
- [Power Data Object: Remote Sink](#)
- [Power Data Object: Remote Source](#)

Request Data Object: Local [Source or Sink]

A Request Data Object (RDO) tells the host about the sink's capabilities and selects one of the Power Data Objects (PDOs) offered by the source by specifying the PDO's index (1-7).

When the port is configured as a sink:

- Tab label shows "Local Sink"
- Panel displays the RDO sent from the hub (sink) to the source device

When the port is configured as a source:

- Tab label shows "Local Source"
- Panel displays the RDO received by the hub (source) from the sinking device (informational only)

The screenshot shows a software window with three tabs: "Power Rules" (selected), "Advertisements", and "Misc". The main panel is titled "Request Data Object: Local Source". It contains a table with two columns: "RDO Information" and "State".

RDO Information	State
Unchunked Message Support	<input type="checkbox"/>
USB Suspend	<input type="checkbox"/>
No USB Communications Possible	<input checked="" type="checkbox"/>
Capability Mismatch	<input type="checkbox"/>
Giveback	<input type="checkbox"/>
Maximum Operate Current	3000mA
Operating Current	3000mA
Object Position (index)	1
Raw	0x1204B12C

Below the table, there is a section titled "Power Data Object: Local Sink". At the bottom of the window, there are fields for "Device Type:", "Serial Number:", and a "<->" button.

Power Rules Advertisements Misc

Request Data Object: Local Sink

RDO Information	State
Unchunked Message Support	<input checked="" type="checkbox"/>
USB Suspend	<input checked="" type="checkbox"/>
No USB Communications Possible	<input checked="" type="checkbox"/>
Capability Mismatch	<input type="checkbox"/>
Giveback	<input type="checkbox"/>
Maximum Operate Current	1500mA <input type="button" value="up"/> <input type="button" value="down"/>
Operating Current	1500mA <input type="button" value="up"/> <input type="button" value="down"/>
Object Position (index)	1 <input type="button" value="up"/> <input type="button" value="down"/>
Raw	0x13825896

Power Data Object: Local Sink

Power Data Object: Local Source

Power Data Object: Remote Sink

Power Data Object: Remote Source -Active- Rule: 1

RDO local source (left) and local sink (right) views

Table 24: *Request Data Object tab options*

RDO Information	Meaning
Unchunked message support	Checked = unchunked mode; supports larger data payloads Unchecked = legacy “chunked” mode; leave unchecked for maximum compatibility
USB suspend	Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
No USB communications possible	Device can not communicate over USB
Capability mismatch	Source PDO modes are not sufficient for device to operate at full capability
Giveback	Device can go to a lower power state if needed
Maximum operate current	Maximum current the sinking device is capable of drawing
Operating current	Nominal operating current of the sinking device
Object position (index)	Index of PDO requested, range 1-7
Raw	The RDO message expressed as 32-bit hexadecimal value

Power Data Object: Local Sink

This tab shows the settings for the port's local sink Power Data Object (PDO). This is how the port tells a remote source about its capabilities and how much power it can accept. If the port is not operating as a sink, these settings have no effect. At the top of the panel is a table of PDO modes that the port can accept as a sink.

Table 25: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Power Rules

Advertisements

Misc

Request Data Object: Local Sink

Power Data Object: Local Sink

En	#	PDO	Voltage (DC)	Amps/Watts
<input checked="" type="checkbox"/>	1	Fixed	5.00	3.00A
<input checked="" type="checkbox"/>	2	Battery	3.30 - 21.00	100.00W
<input checked="" type="checkbox"/>	3	Variable	3.30 - 21.00	5.00A
<input checked="" type="checkbox"/>	4	APDO	3.30 - 21.00	5.00A
<input type="checkbox"/>	5	Fixed	0.00	0.00A
<input type="checkbox"/>	6	Fixed	0.00	0.00A
<input type="checkbox"/>	7	Fixed	0.00	0.00A

PDO Flags

Mode State

Dual Role Data	Auto	<input checked="" type="checkbox"/>
USB Communications Possible	Auto	<input checked="" type="checkbox"/>
Unconstrained Power	Auto	<input type="checkbox"/>
Higher Capability	Auto	<input checked="" type="checkbox"/>
Dual Role Power	Auto	<input checked="" type="checkbox"/>

Edit Sink Rules

Power Data Object: Local Source

Power Data Object: Remote Sink

Power Data Object: Remote Source -Active- Rule: 1

Local Sink PDO list and flags

Table 26: *PDO flags*

PDO flag	Mode	State	Meaning
Dual Role Data USB Communications Possible	En- able, Dis- able, Auto	Checker if flag is en- abled	Can be host or peripheral Supports USB data
Unconstrained Power Higher Capability Dual Role Power			AC powered or large battery Sink needs more than 5 V for full functionality Can source or sink power

Edit sink rules button (requires PD Builder license)

Pops up [Power Rule Editor](#).

Power Data Object: Local Source

This tab shows the settings for the port's local source Power Data Object (PDO). If the port is not operating as a source, these settings have no effect. At the top is a table of PDO modes that the port can provide as a source. If active, the PDO rule index is listed in the tab title.

Table 27: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Power Rules

Advertisements

Misc

Request Data Object: Local Source

Power Data Object: Local Sink

Power Data Object: Local Source -Active- Rule: 4

En	#	PDO	Voltage (DC)	Amps/Watts
<input checked="" type="checkbox"/>	1	Fixed	5.00	3.00A
<input checked="" type="checkbox"/>	2	Fixed	9.00	3.00A
<input checked="" type="checkbox"/>	3	Fixed	15.00	3.00A
<input checked="" type="checkbox"/>	4	Fixed	20.00	5.00A
<input checked="" type="checkbox"/>	5	APDO	5.00 - 11.00	3.00A
<input checked="" type="checkbox"/>	6	APDO	5.00 - 16.00	3.00A
<input checked="" type="checkbox"/>	7	APDO	5.00 - 21.00	5.00A

PDO Flags

Mode State

Unchunked Message Support

Auto

☒

Dual Role Data

Auto

☒

USB Communications Possible

Auto

☒

Unconstrained Power

Auto

☒

USB Suspend

Auto

☐

Dual Role Power

Auto

☒

RDO

State

Maximum Operate Current

4700mA

Operating Current

3750mA

Object Position (index)

4

Raw

0x4785DDD6

Edit Source Rules

Power Data Object: Remote Sink

Power Data Object: Remote Source

Local source PDO list and flags

Table 28: PDO flags

PDO flag		Mode	State	Meaning
Unchunked Support	Message	En-able, Dis-able, Auto	Checked if flag is en-abled	Checked = unchunked mode; supports larger data payloads Unchecked = legacy “chunked” mode; leave unchecked for maximum compatibility
Dual Role Data				Can be host or peripheral
USB Communications Possible				Supports USB data
Unconstrained Power				AC powered or large battery
USB Suspend				Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
Dual Role Power				Can source or sink power

Edit source rule button (requires PD Builder license)

Pops up *Power Rule Editor*.

Power Data Object: Remote Sink

This tab shows the power sink settings for the remote device's Power Data Object (PDO). Since these settings are controlled by the remote device, this tab is informational only. At the top is a list of PDO modes that the remote device can accept as a sink.

Table 29: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Power Rules
Advertisements
Misc

Request Data Object: Local Source
Power Data Object: Local Sink
Power Data Object: Local Source -Active- Rule: 4
Power Data Object: Remote Sink

#	PDO	Voltage (DC)	Amps/Watts
1	Fixed	5.00	3.00A
2	Variable	4.75 - 21.00	4.70A

PDO Flags

Dual Role Data
USB Communications Possible
Unconstrained Power
Higher Capability
Dual Role Power

State
☒
☒
☐
☐
☒

Request Minimum
Request Update

Power Data Object: Remote Source

Remote sink PDO list and flags

Table 30: *PDO flags (informational only)*

PDO flag	State	Meaning
Dual Role Data	Checked if flag is enabled	Can be host or peripheral
USB Communications Possible		Supports USB data
Unconstrained Power		AC powered or large battery
Higher Capability		Sink needs more than 5 V for full functionality
Dual Role Power		Can source or sink power

Request minimum, Request update buttons

[Not implemented]

Power Data Object: Remote Source

This tab shows settings for the remote device's Power Data Object (PDO) and its Request Data Object if active. Since the PDO settings are controlled by the remote device, the PDO mode list and flags are informational only. At the top is a list of PDO modes that the remote device can provide as a source. If the remote device is acting as a source, the active PDO rule index is listed in the tab title.

Power RulesAdvertisementsMisc

Request Data Object: Local Source

Power Data Object: Local Sink

Power Data Object: Local Source -Active- Rule: 1

Power Data Object: Remote Sink

Power Data Object: Remote Source

#	PDO	Voltage (DC)	Amps/Watts
1	Fixed	5.00	1.50A

PDO Flags

State

Unchunked Message Support

Dual Role Data

USB Communications Possible

Unconstrained Power

USB Suspend

Dual Role Power

☒

☒

☒

☐

☐

☒

Request Update

Power Rules
Advertisements
Misc

Request Data Object: Local Sink
Power Data Object: Local Sink
Power Data Object: Local Source
Power Data Object: Remote Sink
Power Data Object: Remote Source -Active- Rule: 1

#	PDO	Voltage (DC)	Amps/Watts
1	Fixed	5.00	1.50A

PDO Flags

Unchunked Message Support

Dual Role Data

USB Communications Possible

Unconstrained Power

USB Suspend

Dual Role Power

State

☒
☒
☒
☐
☐
☒

RDO Control

Maximum Operate Current

Operating Current

Object Position (index)

Raw

State

1500mA

1500mA

1

0x13825896

Request Update

Remote source PDO list, flags and RDO control

Table 31: PDO mode list column headings

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Table 32: Remote source PDO flags (informational only)

PDO flag	State	Meaning
Unchunked Message Support	Checked if flag is enabled	Checked = unchunked mode; supports larger data payloads Unchecked = legacy “chunked” mode; leave unchecked for maximum compatibility
Dual Role Data		Can be host or peripheral
USB Communications Possible		Supports USB data
Unconstrained Power		AC powered or large battery
USB Suspend		Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
Dual Role Power		Can source or sink power

RDO control

When the hub port is acting as a sink, some sink RDO settings can be controlled directly from the bottom of the Remote Source tab. These settings mirror the fields in the *Request Data Object: Local Sink* tab.

Table 33: Remote source RDO control settings

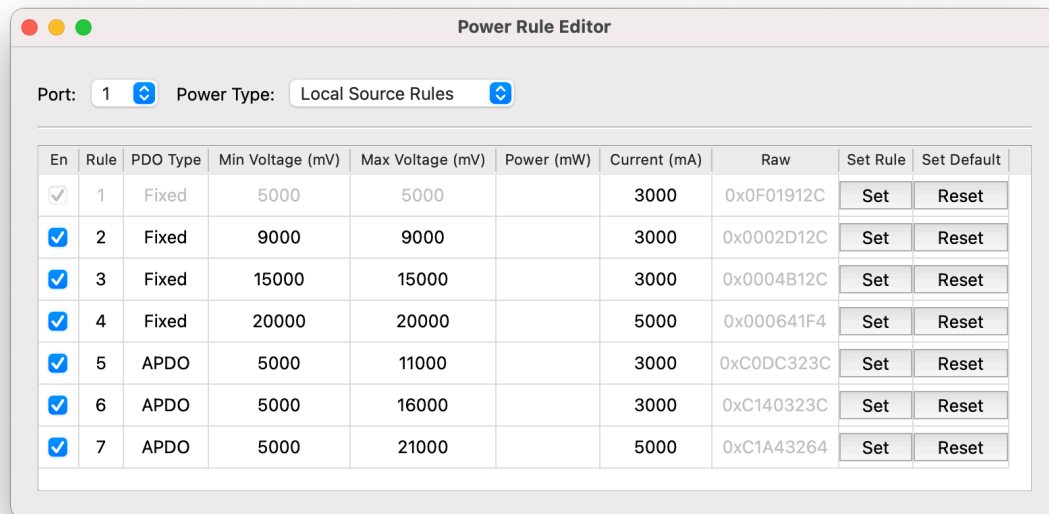
RDO Information	Meaning
Maximum operate current	Maximum current the sinking device is capable of drawing
Operating current	Nominal operating current of the sinking device
Object position (index)	Index of PDO requested, range 1-7
Raw	The RDO message expressed as 32-bit hexadecimal value

Request update button

[Not implemented]

Power rule editor (requires PD builder add-on feature^{Page 387, 71})

⁷¹ <https://acroname.com/store/t99-pd-builder>



The power rule editor sets sink and source power rules for each port.

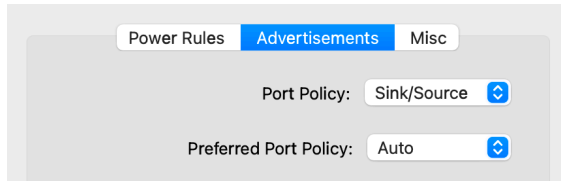
Table 34: *Power rule editor headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
Rule	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Min Voltage (mV)	Min PDO voltage
Max Voltage (mV)	Max PDO voltage
Power (mW)	Max PDO power
Current (mA)	Max PDO current
Raw	Raw bytes of PDO message
Set rule	Set rule after changes
Set default	Return rule to default

Table 35: *PDO Types*

PDO type	Definition
Fixed	Fixed supply voltage of 5 V, 9 V, 15 V, 20 V, current limit 0 A – 5 A. Source must have at least one fixed PDO at 5 V
Variable	Represents a poorly regulated supply. The voltage must stay between the minimum and maximum voltage; minimum voltage must be $\geq 80\%$ of maximum voltage
Battery	Batteries can be connected to VBus directly as a source. Specifies maximum and minimum voltage (in 50 mV steps) and maximum power (in 250 mW steps)
APDO	Programmatically controllable source. Voltage range set in 20 mV steps, up to 3.3V – 21V. Current limit 0 A – 5 A

Port Tab: Advertisements



Power role advertisements subtab

USB-PD advertisements tell connected devices the power policy of the port and its preferred policy.

Port policy

- **None** - no power
- **Source** - provides source
- **Sink** - receives power
- **Sink / Source** - can provide or receive power according to the *Preferred port policy*

Preferred port policy (Only has an effect if port policy is set to *Sink / Source*)

- **None** - no power
- **Source** - provides power
- **Sink** - receives power
- **Follow data** - provides power if downstream-facing, receives power if upstream-facing
- **Auto** - negotiate with the connected device

Port Tab: Misc

Power Rules Advertisements **Misc**

Cable Information:

Voltage Maximum:	N/A
Current Maximum:	N/A
Speed Maximum:	N/A
Type:	N/A
Orientation:	CC1

Send Request:

Hard Reset

Send

Overrides:

- ☐ Cable Current
- ☐ Port Power Limit
- ☐ Auto Discovery

Misc subtab

The Misc subtab contains *Cable information*, *Send request*, and *Overrides*

Cable information

Power Rules Advertisements **Misc**

Cable Information:

Voltage Maximum:	50 VDC
Current Maximum:	5 A
Speed Maximum:	USB 3.2 / USB 4 Gen 2
Type:	Passive (E-marked)
Orientation:	CC1

Send Request:

Manufacturer Info Soppp

Send

Overrides:

- ☐ Cable Current
- ☐ Port Power Limit
- ☐ Auto Discovery

Cable information panel with e-mark data

Displays power and data capacity for e-marked cables connected to the hub and a device.

- Hub port must be providing VConn (VConn strip chart showing 5 V)

- If VConn strip chart shows 0 V, send “VConn Swap” message to swap whether the hub or device is providing VConn

To read e-mark without a device, connect both ends of the cable to ports of USBHub3c, with one port set as *sink* or *auto* and the other set to *source* or *auto*. View the cable info panel on the source port.

Send Request

This control sends USB PD requests to the connected device to initiate a connection sequence. There is no guarantee that the request will succeed, and many of these request messages will be unsupported by the connected device.

Send request	Definition
Hard Reset	Reinitialize PD communications and cycle VBus
Soft Reset	Reinitialize PD communications, but maintains power connection
Data Reset	Cycle USB data connection [unimplemented]
Power Role Swap	Exchange source and sink power roles
Power Fast Role Swap	Fast exchange source and sink power roles
Data Role Swap	Swap upstream and downstream ask roles
VCONN Swap	Swap whether the hub or device is supplying VConn
Sink Go To Min	Sink goes to minimum power draw
Request Remote Source PDOs	Get PDO from remote source
Request Remote Sink PDOs	Get PDO from remote sink
Request Remote Extended Source Caps	Get extended source capabilities
Request Remote Extended Sink Caps	Get extended sink capabilities
Status	Get status
PPS Status	Get PPS status
Battery Capabilities	Get battery capabilities (Design cap, last full cap)
Battery Status	Get Battery status (state of charge, charging status)
Manufacturer Info Sop	Get device manufacture info
Manufacturer Info Sopp	Get cable manufacturer info (nearest VConn sourcing port)
Manufacturer Info Soppp	Get cable manufacturer info (opposite from VConn sourcing port)
Discover Identity Sop	Device type and capabilities
Discover Identity Sopp	Cable type and capabilities (Vconn end)
Discover Identity Soppp	Cable type and capabilities (not VConn end)
Revision	USB PD revision and version numbers for connected device
Source Info	Source port power capability
Country Codes	Country of origin
Country Info	Country of origin

Overrides

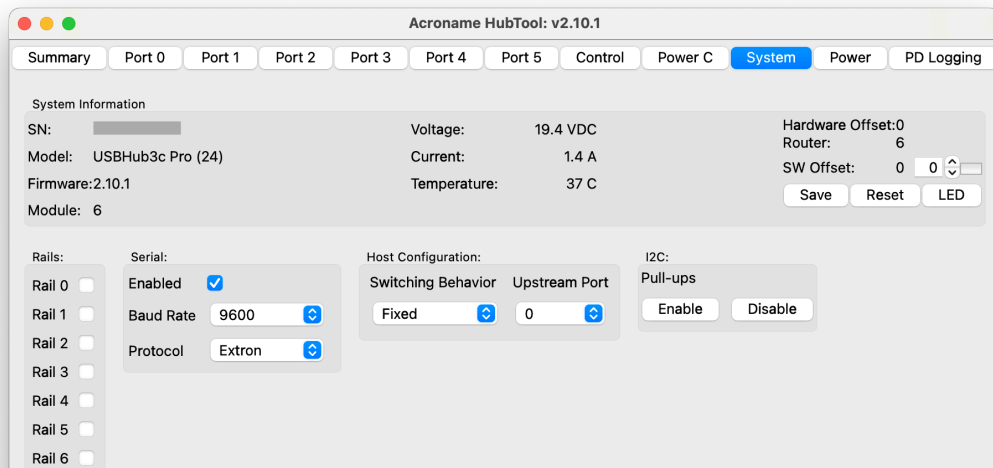
Cable current – overrides the 3 A current limit for cables that don't specify 5 A via e-mark

Port power limit – overrides port power budgeting and allows full power output up to 105 W per port

Warning: Can exceed power supply current limit and brown out the hub, triggering a reset

Auto discovery – overrides the auto discovery feature. When enabled, the hub will only establish a basic power connection and not request vendor information. Used for legacy compatibility and debugging.

System Tab



HubTool USBHub3c system tab

The system tab contains information and settings for the hub.

System information

Heading	Meaning
SN	Hub serial number
Model	Hub model name and number, e.g. USBHub3c Pro (24)
Firmware	Currently installed firmware version
Module	Address of the module on the Brainstem network
Voltage	Voltage at the input power source port
Current	Input current at the input power source port
Temperature	System temperature
Hardware off-set	Increments Brainstem module address by a hardware setting - fixed to 0 in USBHub3c
Router	Address of the routing Brainstem module
Software offset	Increment the module address. Requires Save and Reset
Save	Store select changed settings
Reset	Soft reset of the hub
LED	Toggles blue user LED on front panel, used debugging and to identify the hub

Rails

Requires [External Load⁷²](#) add-on feature. Array of toggles to enable VBus rails 0 – 6 on the 20-pin expansion connector. Rails 0 - 5 represent ports 0 – 5. Rail 6 is an additional 5 V rail for triggering or powering external devices. Default = off

Serial

Requires RS232 [Serial Control Feature⁷³](#) feature. Use [Serial RS232 Expansion Connector Accessory⁷⁴](#) to break out the serial pins.

- Enabled (toggle)
- Baud rate (1200 – 115200 bps)
- Protocol (*Undefined, Extron, BrainStem*)

⁷² <https://acroname.com/store/t99-ext-load>

⁷³ <https://acroname.com/store/t99-serial>

⁷⁴ <https://acroname.com/store/s103-serial-exp>

Host configuration

Switching behavior

- Fixed (default)
- Port priority - sets lowest-numbered port that is connected to a host-capable device to upstream mode

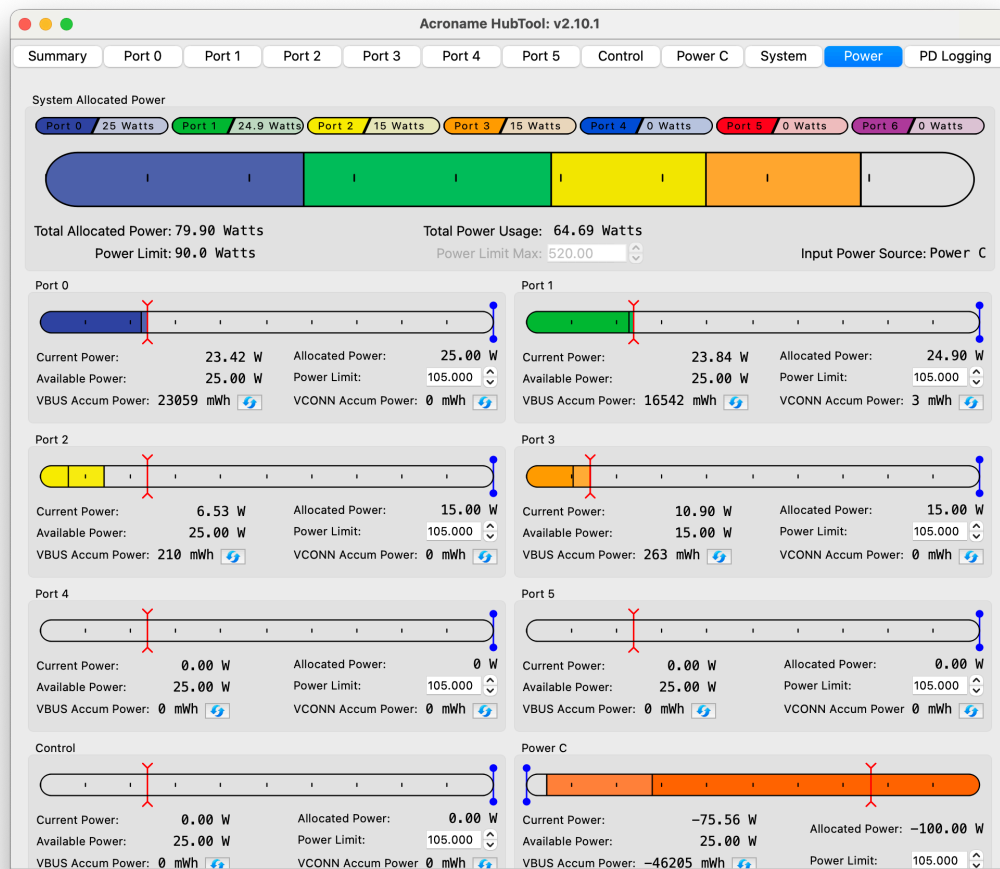
Upstream port

Shows current upstream port. Select Port 0 – 5 to connect to host - default = 0

I2C

Enables and disables pullups on the I2C bus - IO3 (SDA) and IO4 (SCL) on the expansion connector

Power Tab

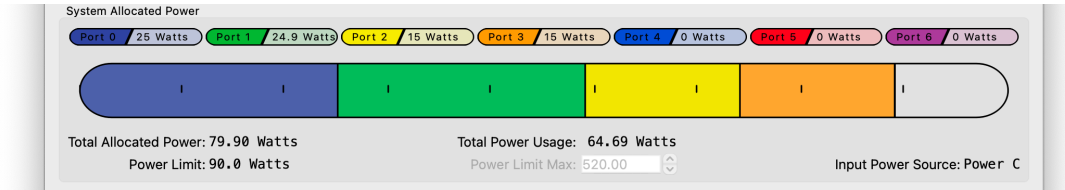


HubTool USBHub3c Power tab

The power tab shows the allocation and actual flow of power through the hub.

System allocated power panel

Across the top of this panel are the allocated currents of each port. Positive currents represent that the port is a power source, negative currents are for sinks.



System allocated power panel

Below the row of ports is a visualization of the stackup of allocated currents.

Total allocated power - sum of the allocated source power

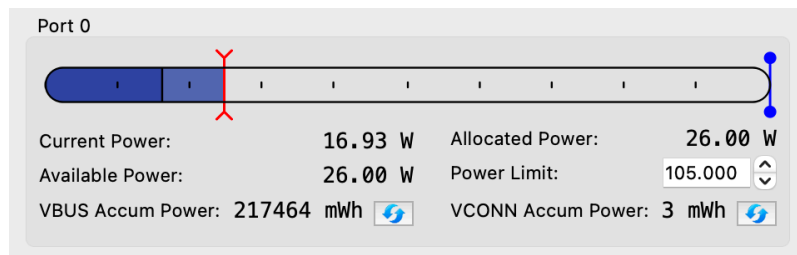
Power limit - maximum power that can be allocated across all ports. Determined by input power minus nominal losses, e.g. 90 W for 100 W supply

Total power usage - sum of actual power being sourced

Power limit max - set power limit for DC input when unregulated power is provided to the DC port

Input power source - input port with the highest power capability

Port power panel



Port power panel

Current power - actual port power output (positive) or input (negative)

Available power - maximum power that could be allocated to the port, not to exceed *Power limit*

VBUS accum power - total energy sunk or sourced on VBus since reset

Allocated power - power allocated to the port, not to exceed *Power limit*

Power limit - power limit for the port, max 105 W

VConn accum power - total energy sunk or sourced on VConn since reset

PD Logging Tab (Add-on feature)

	Hub Time (Sec)	Port	TX/RX	Spec	SOP	Power	Data	ID	Event Type	Packet Type	Msg Type	Value
47	11145.482557	2	-	-	-	-	-	-	Rp 3A	-	-	
48	11145.528672	2	-	-	-	-	-	-	Rp 1.5A	-	-	
49	11145.549837	2	TX	V3.0	SOP	Source	DFP	7	PD Packet	Control	Get Source Info	0xB7 0x0F
50	11145.588353	2	-	-	-	-	-	-	Rp 3A	-	-	
51	11145.631715	2	-	-	-	-	-	-	Rp 1.5A	-	-	
52	11145.653669	2	TX	V3.0	SOP	Source	DFP	0	PD Packet	Control	Get Status	0xB2 0x01
53	11145.661734	2	RX	V3.0	SOP	Sink	UFP	3	PD Packet	Extended	Status	0x82 0x86 0x00
54	11145.663672	2	-	-	-	-	-	-	Rp 3A	-	-	
55	11145.766971	2	-	-	-	-	-	-	Rp 1.5A	-	-	
56	11145.790333	2	TX	V3.0	SOP	Source	DFP	1	PD Packet	Extended	Get Manufacturer Info	0xA6 0x83 0x00
57	11145.824871	2	-	-	-	-	-	-	Rp 3A	-	-	
58	11145.829320	2	-	-	-	-	-	-	Rp 1.5A	-	-	
59	11145.851615	2	TX	V3.0	SOP	Source	DFP	2	PD Packet	Data	Vendor Defined	0xAF 0x15 0x01
60	11145.858941	2	RX	V3.0	SOP	Sink	UFP	4	PD Packet	Data	Vendor Defined	0x8F 0x48 0x40
61	11145.861406	2	-	-	-	-	-	-	Rp 3A	-	-	
62	11145.865149	2	-	-	-	-	-	-	Rp 3A	-	-	
63	11145.870225	2	-	-	-	-	-	-	Rp 1.5A	-	-	
64	11145.889852	2	TX	V3.0	SOP	Source	DFP	3	PD Packet	Data	Vendor Defined	0xAF 0x17 0x02
65	11145.895155	2	RX	V3.0	SOP	Sink	UFP	5	PD Packet	Data	Vendor Defined	0x8F 0x2A 0x40
66	11145.895800	2	-	-	-	-	-	-	Rp 3A	-	-	
67	11145.899409	2	-	-	-	-	-	-	Rp 3A	-	-	
68	11145.904641	2	-	-	-	-	-	-	Rp 1.5A	-	-	
69	11145.923927	2	TX	V3.0	SOP	Source	DFP	4	PD Packet	Data	Vendor Defined	0xAF 0x19 0x03
70	11145.928094	2	-	-	-	-	-	-	Rp 3A	-	-	
71	11145.929104	2	RX	V3.0	SOP	Sink	UFP	6	PD Packet	Data	Vendor Defined	0x8F 0x2C 0x40
72	11145.932033	2	-	-	-	-	-	-	Rp 3A	-	-	
73	11149.544223	2	RX	V3.0	SOP	Sink	UFP	7	PD Packet	Data	Request	0x82 0x1E 0x2C
74	11149.547846	2	TX	V3.0	SOP	Source	DFP	5	PD Packet	Control	Accept	0xA3 0x0B
75	11149.627622	2	TX	V3.0	SOP	Source	DFP	6	PD Packet	Control	PS Ready	0xA6 0x0D

PD logging tab view

PD logging⁷⁵ is a software add-on feature enabling capture, logging, and decoding of USB PD communications on all USB-C ports of the hub, including power negotiations and timing. At the top of the panel are logging controls and port selection toggles.

Stop - stops logging

Import - imports a CSV log file

Export - exports a CSV log file (stop logging before export)

Clear - clears the PD log

Port toggles - select which ports to monitor (can only change toggles when logging is stopped)

Show graph - when checked, clicking on an event pops up the *Port Plot* graph window and highlights the corresponding PD packet. Clicking on a yellow PD message in the port plot highlights the corresponding message in the log

⁷⁵ <https://acroname.com/store/t99-pd-log>



Graph view with highlighted PD packet

Below these controls is the PD packet log. The left axis is the row number, which resets when the log is cleared. Columns can be filtered by clicking the filter icon (🔍) on the column headings:

Hub Time (Sec)	Port	<->	Spec	SOP	Power	Data	ID	Event Type	Packet Type	Msg Type	Raw
----------------	------	-----	------	-----	-------	------	----	------------	-------------	----------	-----

PD packet log headings

Time - clicking on this heading cycles among time references:

- Hub time (s) - time since the hub powered on
- App time (hh:mm:ss) - time since the HubTool App was launched
- System time (yyyy.MM.dd hh:mm:ss:zzzz) - date and time

Port - ports 0-5, Control, or Power C

<-> - message direction - *RX*, *TX*, or '-' (none)

Spec - USB PD version

SOP* - "Start Of Packet"

- SOP - for messages between source and sink
- SOP' - for messages to the cable connector closest to the downstream-facing port (DFP)
- SOP" - for messages to the cable connector closest to the upstream-facing port (UFP)

Power - *sink*, *source*, *none*

Data - *UFP*, *DFP*, '-'

ID - message ID, (0 – 7), increments with each new message. Acknowledgements should match message ID

Event type - description of the event

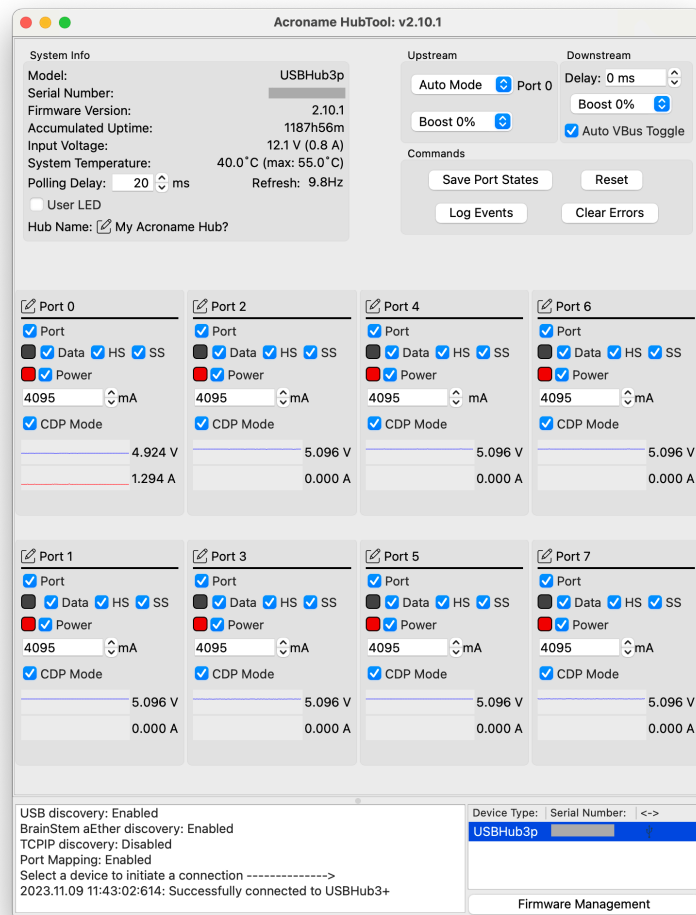
Packet type

- Control - short messages that typically require no data exchange
- Data - messages contain data objects that are transmitted between devices
- Extended - data messages with larger data payloads

Message type - description of message

Raw - raw PD message (hexadecimal)

USBHub3p

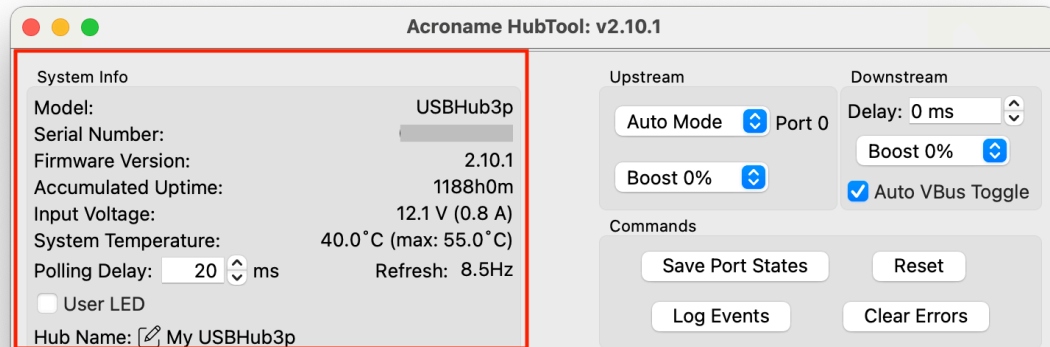


The [USBHub3p](https://acroname.com/store/programmable-industrial-hub-s79-usbhub-3p)⁷⁶ is a programmable USB hub with 8 full-featured 5 Gbps USB-A ports and 1 downlink port for daisy chaining. Two upstream ports allow automatic or manual host switching.

HubTool presents a unified dashboard to control and view state of USBHub3p.

⁷⁶ <https://acroname.com/store/programmable-industrial-hub-s79-usbhub-3p>

General system information



USBHub3p general system information

The upper-left panel shows **general system information** for the hub:

Serial number

Firmware version

Accumulated uptime

The total time the system has been powered on since leaving the factory

Input voltage (V)

System temperature (°C)

Polling delay (ms)

Sets how long to wait after receiving all information from the hub to poll again. A setting of 0 will start a new set of requests immediately. Polling takes 50-150 ms.

Refresh (Hz)

The measured refresh rate, which is the inverse of the polling delay plus the time to receive the data

User LED toggle

Toggles a blue LED located on the USBHub3p back panel - used for debugging and to identify the hub

Hub name

Editable friendly name for the hub - used by HubTool and ControlRoom

Port settings and commands

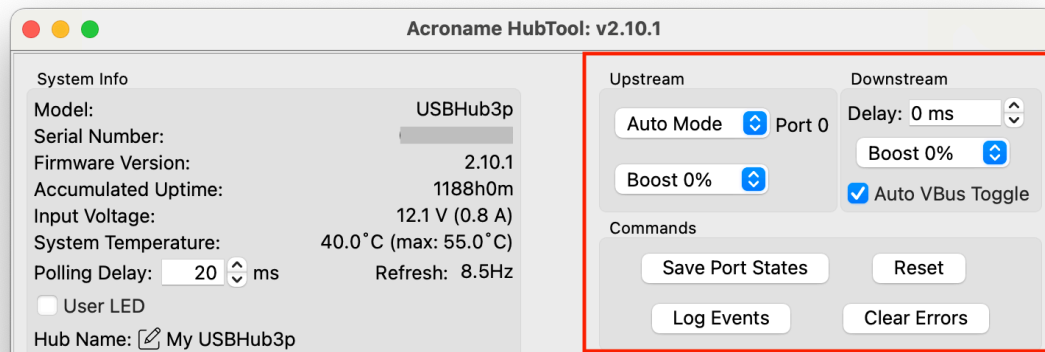
USBHub3p port settings and commands

The upper-right panel shows **port settings** and **commands**:

Upstream (host) port settings:

Host select:

- Auto mode (Port 0 if both ports connected)



- Port 0 only
- Port 1 only
- None (do not connect to either host port)

Note that the Control port (mini USB, located on USBHub3p back panel) is always accessible

Boost

Amplifies the upstream USB signal up to 12 percent to improve marginal connections - default = 0

Downstream port settings:

Delay (ms)

Delays the enumeration of all downstream ports when power is enabled. Useful if devices are slow to power on and don't respond to enumeration in time - default = 0

Boost

Amplifies the downstream USB signals up to 12 percent to improve marginal connections - default = 0

Auto VBus toggle

Toggles downstream port VBus when the upstream connection changes - default = on

Commands

Save port states

Saves the settings of all ports to the hub's internal memory so that they will persist through power cycling and reset. Saved states: **Host select, Boost, Delay, Power and data toggles, Current limit, CDP mode toggle**

Reset

Resets the hub - VBus toggles and hub returns to previously saved state

Log Events

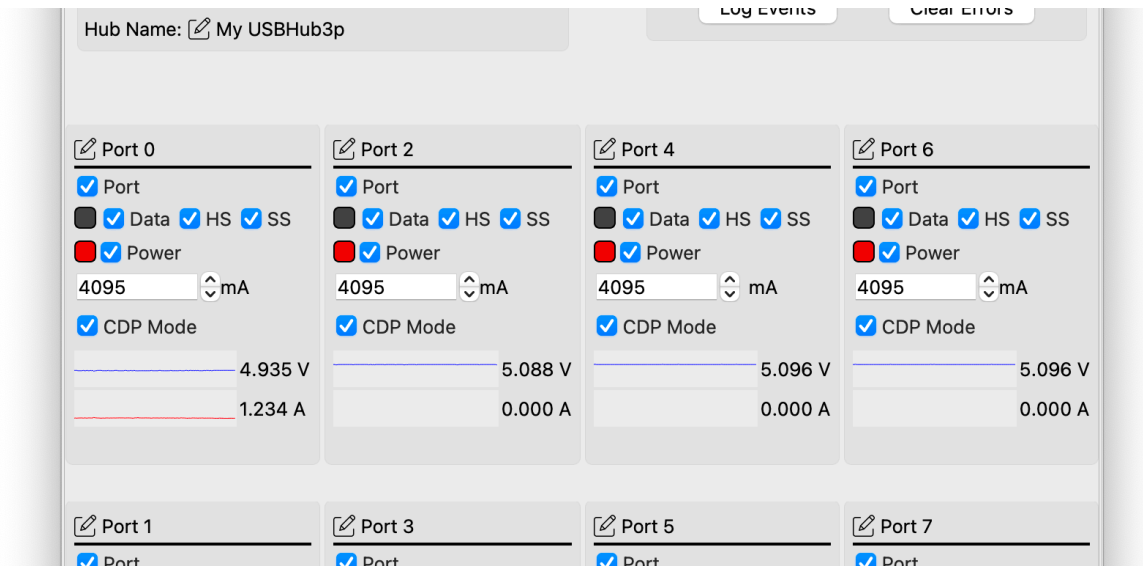
Displays the last 150 log events in the console

Clear errors

Clears internal error bits

Port view

Each of 8 full-featured downstream ports on the front panel of USBHub3p has its own interface panel in HubTool.



Downstream port view

Port name

Editable “friendly” name for the port that is displayed in HubTool and ControlRoom interfaces

Power and data toggles

Toggle name	Enables and disables
Port	Entire port
Data	All Data
HS	USB 2 (High Speed) pins
SS	USB 3 (Super Speed) pins
Power	VBus

Virtual LEDs

These indicate the data and power status of the port and match the real LEDs on the front panel of the hub.

Attribute	Status	LED color
Data rate	USB 2	Yellow
	USB 3	Green
	USB 2, USB 3	Green
VBus	Powered	Red
	Off	Black

Current limit

Maximum current that the hub will supply - default = 4095 mA

CDP mode toggle

Enables Charging Downstream Port modes (default = on). The port will signal the attached device to draw up to a maximum current based on this table:

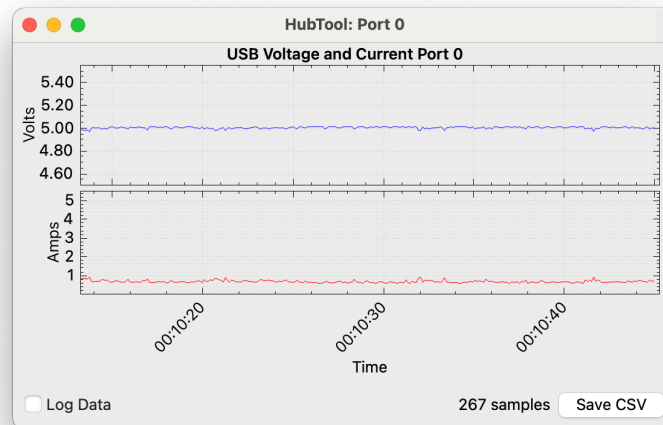
Table 36: *Port power modes*

CDP mode toggle	Condition	Port mode	Maximum current (device limited)*
On	Host present and USB 2 data lines enabled	CDP	1500 mA
	No host present and USB 2 data lines enabled	DCP (Device Charging Port) mode	5000 mA
Off	No host or no USB 2 data lines connected	SDP (Standard Downstream Port)	100 mA
	Host present and USB 2 data lines enabled		500 mA

* The hub limits current to **current limit**, up to a maximum of 4000 mA

Voltage and current display

Shows a graph of the port's bus voltage and current. Clicking on the port graph pops up a window with a larger rolling chart of the last 32 seconds.



Voltage and current strip chart and logging

Decreasing polling delay will increase the number of samples used in the chart. Selecting the “Log Data” toggle switches the chart from rolling to expanding mode. Clicking “Save CSV” saves the data of the graph view in a .CSV file.

Table 37: Example .CSV file output

Time (s)	Port 0 Voltage (V)	Port 0 Current (A)
6375.647	5.104	0.000
6375.838	5.104	0.000
6376.035	5.104	0.000
6376.238	5.104	0.000
6376.432	5.096	0.000
...		

Device descriptors

If Options > Port Mapping is selected, when a device is attached to a downstream port, its descriptors will scroll at the bottom of the port panel. Click the carat (^) to expand:

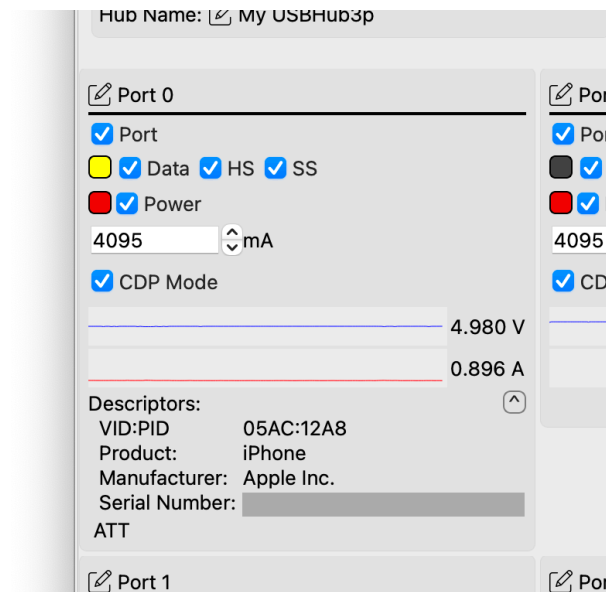


Table 38: Descriptor table

Descriptor	Content
VID:PID	16-bit vendor ID and 16-bit product ID
Product	Product name string
Manufacturer	Manufacturer name string
Product serial number	Product serial number
ATT	Indicates device is attached

USBHub2x4



The [USBHub2x4⁷⁷](https://acroname.com/store/industrial-intelligent-4-port-hub-s77-usbhub-2x4) is a 4-port software-programmable 480Mbps (USB 2.0) hub designed for demanding environments where advanced control and monitoring of USB ports is required. Two upstream ports allow automatic or manual host switching.

HubTool presents a unified dashboard to control and view state of USBHub2x4.

General system information

USBHub2x4 general system information

The upper-left panel shows **general system information** for the hub:

Serial number

Firmware version

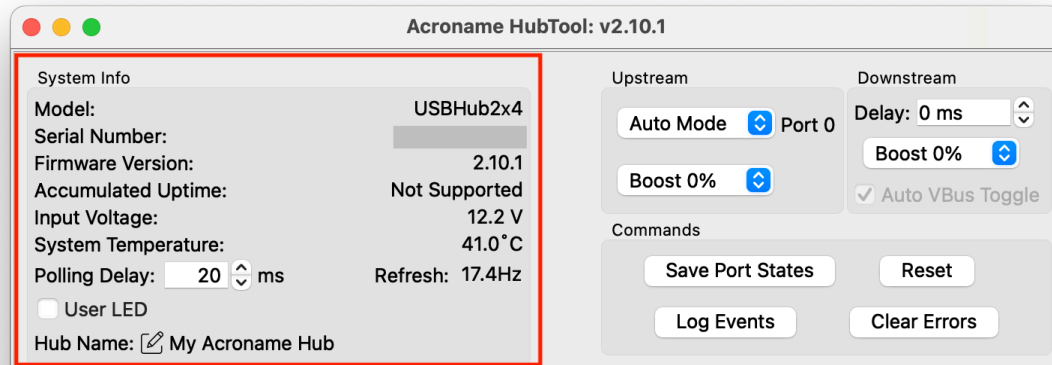
Accumulated uptime

The total time the system has been powered on since leaving the factory - *unsupported in USBHub2x4*

Input voltage (V)

System temperature (°C)

⁷⁷ <https://acroname.com/store/industrial-intelligent-4-port-hub-s77-usbhub-2x4>



Polling delay (ms)

Sets how long to wait after receiving all information from the hub to poll again. A setting of 0 will start a new set of requests immediately. Polling takes 50-150 ms.

Refresh (Hz)

The measured refresh rate, which is the inverse of the polling delay plus the time to receive the data

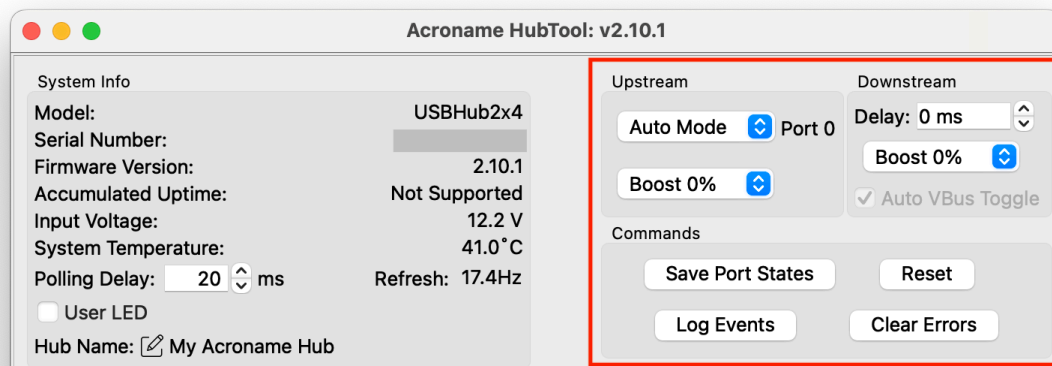
User LED toggle

Toggles a blue LED (7th from the top) visible through the slot in the cover of USBHub2x4 - used for debugging and to identify the hub.

Hub name

Editable friendly name for the hub - used by HubTool and ControlRoom

Port settings and commands



USBHub2x4 port settings and commands

The upper-right panel shows **port settings** and **commands**:

Upstream (host) port settings:

Host select:

- Auto mode (Port 0 if both ports connected, defaults to auto after power cycle)
- Port 0 only

- Port 1 only
- None (do not connect to either host port - *no effect in USBHub2x4*)

Boost

Amplifies the upstream USB signal up to 12 percent to improve marginal connections - default = 0

Downstream port settings:

Delay (ms)

Delays the enumeration of all downstream ports when power is enabled. Useful if devices are slow to power on and don't respond to enumeration in time - default = 0

Boost

Amplifies the downstream USB signals up to 12 percent to improve marginal connections - default = 0

Auto VBus toggle

Toggles downstream port VBus when the upstream connection changes - *always on for USBHub2x4*

Commands

Save port states

Saves the settings of all ports to the hub's internal memory so that they will persist through power cycling and reset. Saved states: **Boost, Delay, Power and data toggles, Current limit, CDP mode toggle**

Reset

Resets the hub - VBus toggles and hub returns to previously saved state

Log Events

Displays the last 150 log events in the console

Clear errors

Clears internal error bits

Port view

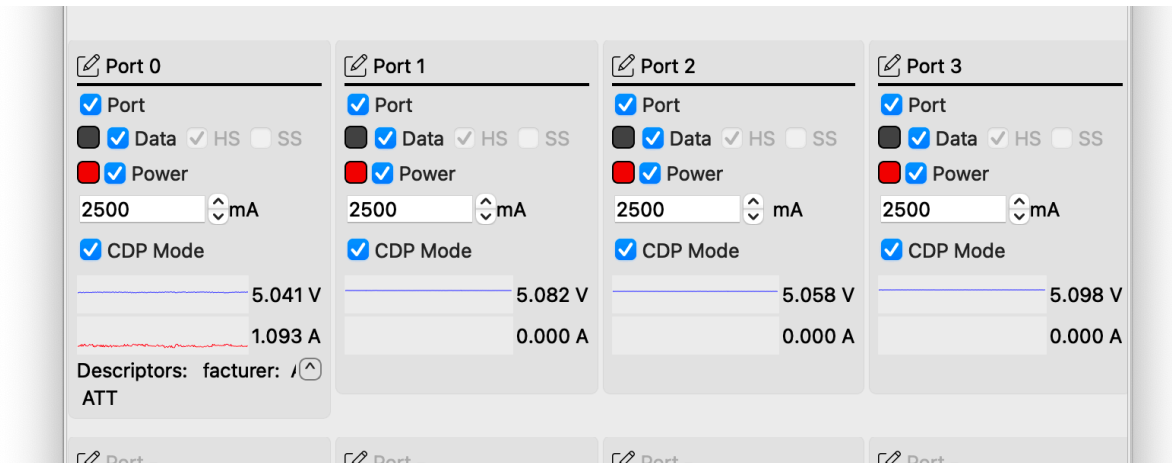
Each of 4 USB 2.0 downstream ports on the front panel of USBHub2x4 has its own interface panel in HubTool. A second row of 4 more ports are grayed out and unavailable with USBHub2x4.

Downstream port view

Port name

Editable "friendly" name for the port that is displayed in HubTool and ControlRoom interfaces

Power and data toggles



Toggle name	Enables and disables
Port	Entire port
Data	All Data
HS (grayed out, checked)	USB 2 (High Speed) pins - <i>always enabled</i>
SS (grayed out, unchecked)	USB 3 (Super Speed) pins - <i>not available on USBHub2x4</i>
Power	VBus

Current limit

Maximum current that the hub will supply - default = 2500 mA

CDP mode toggle

Enables Charging Downstream Port modes (default = on). The port will signal the attached device to draw up to a maximum current based on this table:

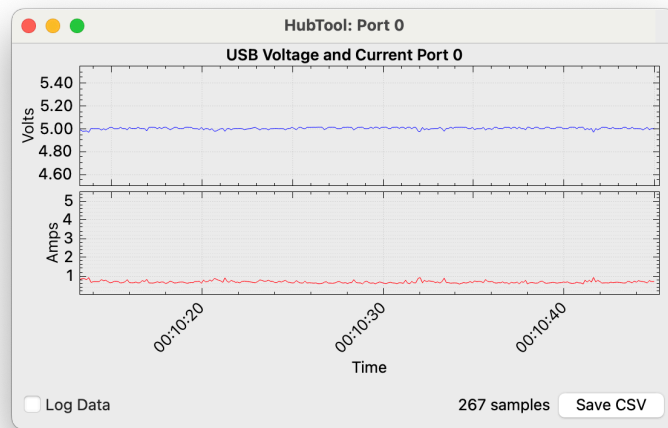
Table 39: *Port power modes*

CDP mode toggle	Condition	Port mode	Maximum current (device limited)*
On	Host present and USB 2 data lines enabled	CDP (Charging Downstream Port)	1500 mA
Off	No host or no USB 2 data lines connected	SDP (Standard Downstream Port)	100 mA
	Host present and USB 2 data lines enabled		500 mA

* The hub limits current to **current limit**, up to a maximum of 2500 mA

Voltage and current display

Shows a graph of the port's bus voltage and current. Clicking on the port graph pops up a window with a larger rolling chart of the last 32 seconds.



Voltage and current strip chart and logging

Decreasing polling delay will increase the number of samples used in the chart. Selecting the “Log Data” toggle switches the chart from rolling to expanding mode. Clicking “Save CSV” saves the data of the graph view in a .CSV file.

Table 40: *Example .CSV file output*

Time (s)	Port 0 Voltage (V)	Port 0 Current (A)
6375.647	5.104	0.000
6375.838	5.104	0.000
6376.035	5.104	0.000
6376.238	5.104	0.000
6376.432	5.096	0.000
...		

Device descriptors

If Options > Port Mapping is selected, when a device is attached to a downstream port, its descriptors will scroll at the bottom of the port panel. Click the carat (⤴) to expand:

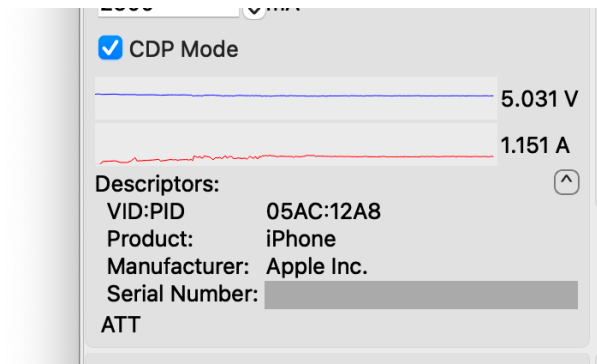
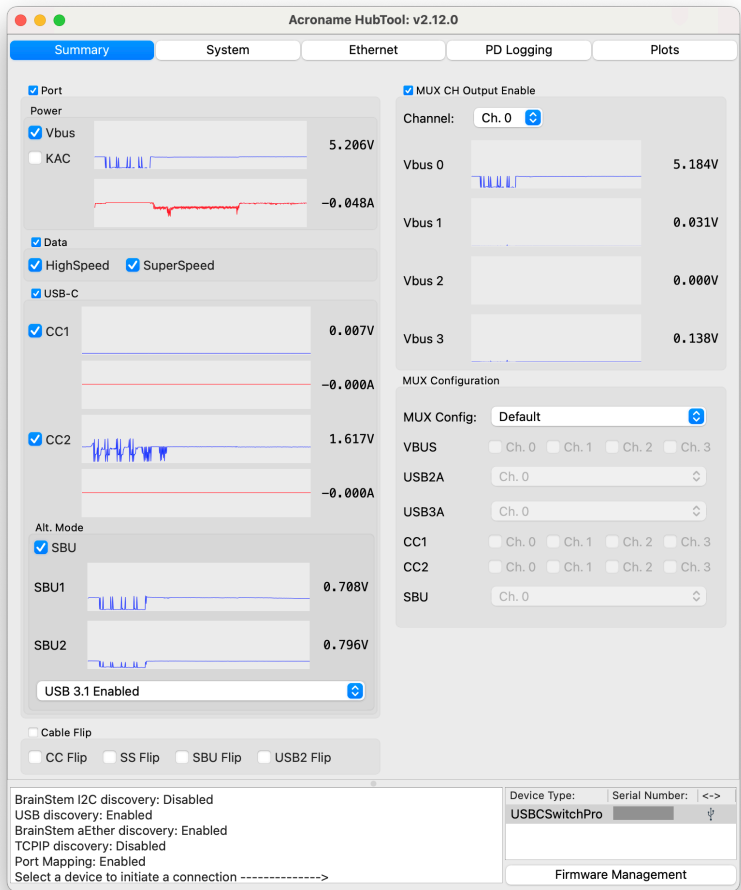


Table 41: *Descriptor table*

Descriptor	Content
VID:PID	16-bit vendor ID and 16-bit product ID
Product	Product name string
Manufacturer	Manufacturer name string
Product serial number	Product serial number
ATT	Indicates device is attached

USB-C-Switch Pro



⁷⁸ *HubTool interface for*

USB-C-Switch-Pro

Acroname's **USB-C-Switch Pro**⁷⁹ is a managed USB4 1:4 / 4:1 port selector and multiplexer switch supporting 240 W USB Power Delivery Extended Power Range (USB PD EPR) and 8K@120 Hz DisplayPort Alt mode video pass through.

The USB-C-Switch Pro is not a hub — it directly connects and redrives signals from the common port to one of four mux channels or from one mux port to the common port, appearing **like a cable** to the connected

⁷⁸ <https://acroname.com/store/s105-usbc-switch-pro>

⁷⁹ <https://acroname.com/store/s105-usbc-switch-pro>

devices. Data link, power negotiations, and power between USB devices are managed by the attached devices themselves.

When combined with an Acroname [Universal Orientation Cable⁸⁰](#), the USB-C-Switch Pro can emulate a cable flip, allowing tests of all the connections on a port without the need to manually flip the cable.

HubTool presents a unified dashboard to control and view the state of USB-C-Switch Pro.

Connections

Port	Connects to
Control, Ethernet, or RS-232	Control computer running HubTool
Common	Common host or Common device
Ports 0 – 3	Switched devices Switched hosts
DC Power	Optional DC power supply

Add-on software features

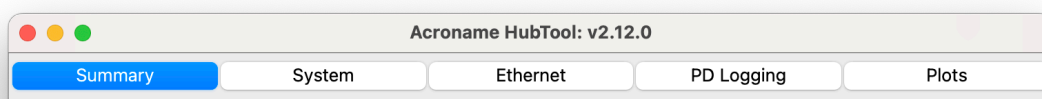
USB-C-Switch Pro supports add-on software features, which can be purchased to enable new capabilities. When a USB-C-Switch Pro is selected in HubTool, the console will list the available add-on software features and show their enabled / disabled status on the device.

Table 42: Add-on licensed software features

Feature	Capability
PD logger⁸¹	Log USB-PD communications on all ports
Serial control⁸²	Enable RS-232 serial control of the Switch

Licenses for additional features are [available for purchase⁸³](#). After purchase, [update the hub firmware](#) to enable the new features.

Dashboard tabs



HubTool USB-C-Switch Pro dashboard tab view

After selecting USB-C-Switch Pro, HubTool will launch the device dashboard in summary view. Buttons along the top of the window represent tabs for:

⁸⁰ <https://acroname.com/store/c46-usbc-uoc>

⁸¹ <https://acroname.com/store/t99-pd-log>

⁸² <https://acroname.com/store/t99-serial>

⁸³ https://acroname.com/store-grid/field_category/Software-Licenses

Summary Tab

The Summary Tab is the primary interface to the USB-C switch, with Common and Mux Port controls and status.

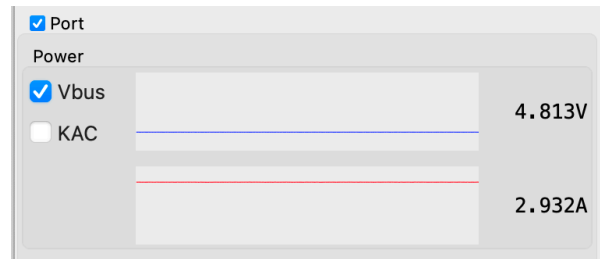
Common Port controls



The left side of the Summary tab shows controls and attributes related to the Common Port:

- Enabling and disabling individual lines
- Viewing voltage and current on V_{BUS} , CC1, CC2, SBU1, SBU2
- Keep-alive charging
- Alt mode
- Cable flip

Power



- **Port toggle** – enables and disables all lines connecting the Common port to the selected mux channel
- **Vbus toggle** – enables and disables V_{BUS} lines
- **KAC toggle** – enables Keep-alive charging (KAC)
Keep-alive charging helps keep battery-powered devices on the non-selected mux ports charged. When enabled, the KAC circuit connects power from the control port V_{BUS} to all non-selected mux channel V_{BUS} lines. See the [API reference](#) for more detail
- **VBus voltage and current plots** – shows voltage and current for V_{BUS}

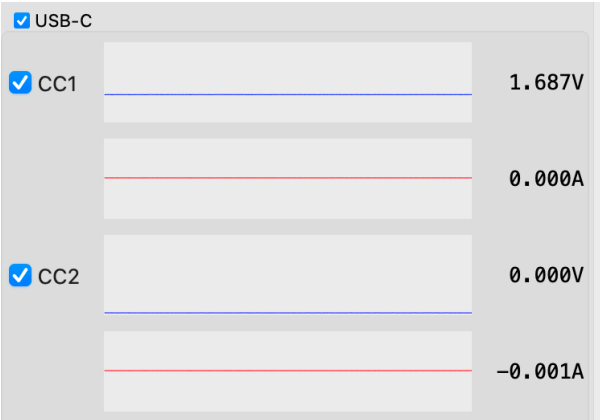
Data line toggles



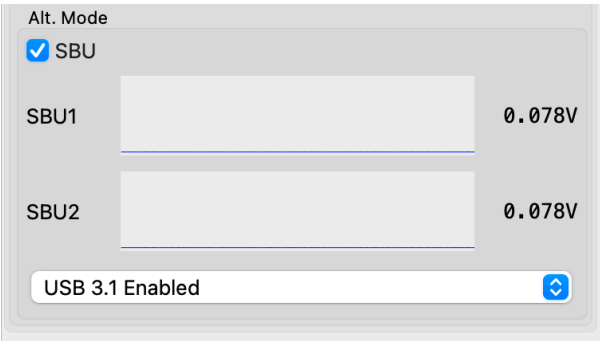
- **Data toggle** – enables and disables all USB data lines
- **HighSpeed, SuperSpeed toggles** – enable and disable USB 2 (HighSpeed) and USB 3+ (SuperSpeed) data lines

CC

- **USB-C toggle** – enables and disables CC and SBU lines
- **CC toggles** – enable and disables CC1 and CC2 lines independently
- **CC1 and CC2 voltage and current plots** – shows voltage and current for CC1 and CC2



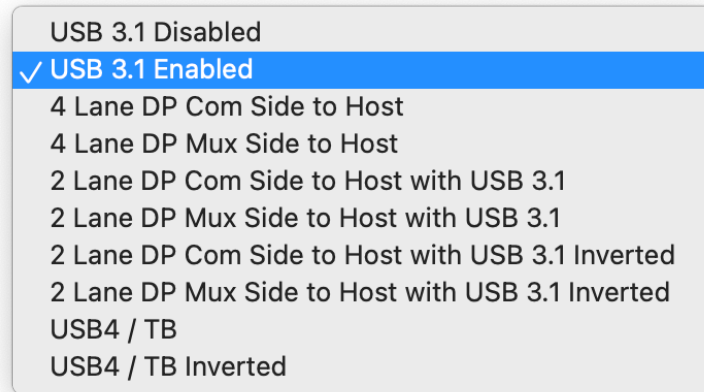
Alt mode



- **SBU toggle** – enables and disables SBU lines. Used for Alternate Mode discovery, negotiation, and configuration data exchange
- **SBU1 and SBU2 voltage and current plots** – shows voltage and current for SBU1 and SBU2
- **DisplayPort alt mode configuration menu**

Alt Mode Configuration
USB 3.1 Disabled – no SS lines connected
USB 3.1 Enabled – SS lines connected
4-Lane DisplayPort – no USB 3.1 – Host on Common Port
4-Lane DisplayPort – no USB 3.1 – Host on Mux Port
2-Lane DisplayPort + USB 3.1 – Host on Common Port
2-Lane DisplayPort + USB 3.1 – Host on Mux Port
2-Lane DisplayPort + USB 3.1 Inverted – Host on Common Port
2-Lane DisplayPort + USB 3.1 Inverted – Host on Mux Port
USB4 / Thunderbolt 3
USB4 / Thunderbolt 3 Inverted

USB4 / ThunderBolt 3 and DisplayPort Alt modes use the SS lines, but change their direction. Since each redriver line can operate in only one direction at a time, the



redriver direction needs to be set to match the mode being used by the host and device.

- If device V_{BUS} is not active, toggle cable flip or check cable orientations
- In the Alt Mode menu, select the corresponding configuration depending on whether the host is on the Common or Mux port and the protocol used. For 2-lane DP and USB4/ TB, it may be necessary to try the inverted and non-inverted option

See the :ref: *API reference <csp-alt-mode-configuration>* for more detail.

Cable flip

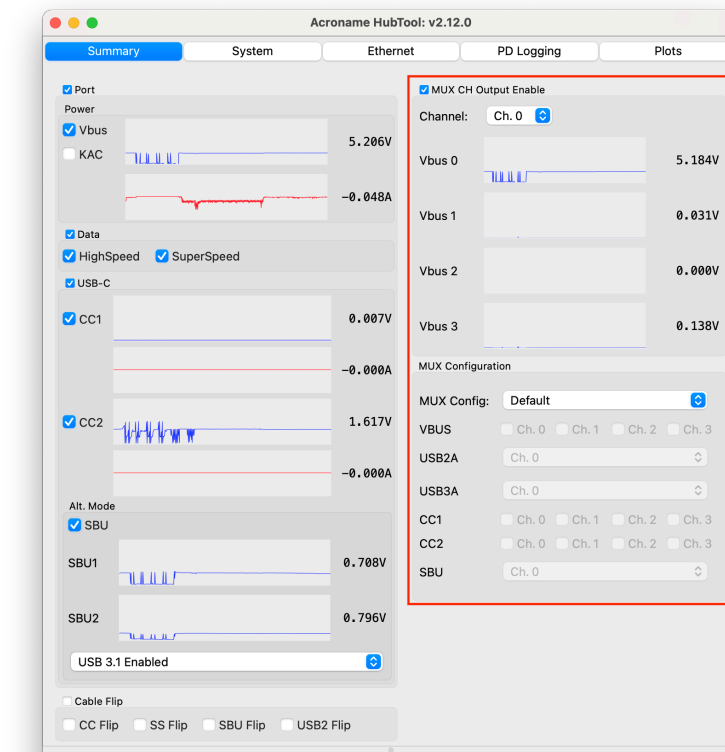


- **Cable flip toggle** – switches USB data, V_{CONN} , and SBU lines from side A to side B as if the cable had been flipped. When using standard USB-C cables with USB-C-Switch Pro, one cable orientation will work, the other will need to be flipped physically, or by using the toggle.
- **CC, SS, SBU, USB2 flip toggles** – individually flip each connection type

To enable automated cable flips for testing, use one [Universal Orientation Cable](https://acroname.com/store/c44-usb-uoc)⁸⁴ for either the host or mux connection, and one standard cable for the other connection.

⁸⁴ <https://acroname.com/store/c44-usb-uoc>

Mux Port controls



The right side of HubTool shows controls and attributes related to the mux ports:

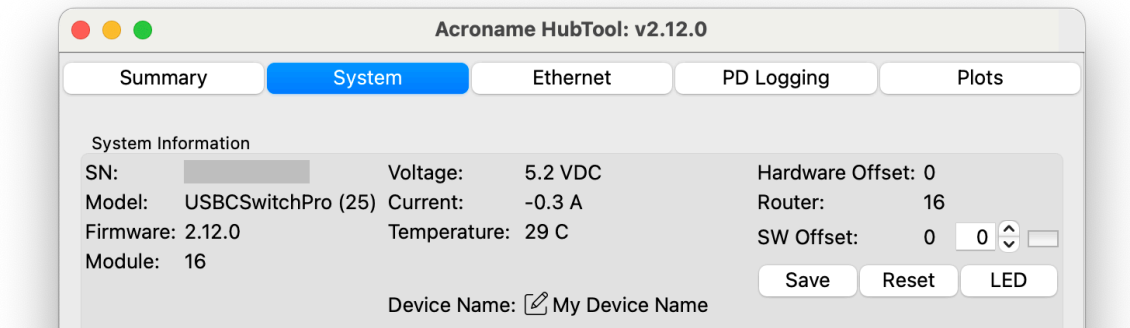
Mux channel control

- **Mux channel output enable** – toggles the output of all mux ports
- **Mux channel selector** – designates one mux port to connect to the Common port. Unavailable in Channel priority and Split configuration
- **VBus voltage plots** – shows V_{BUS} voltage for each mux port

Mux configuration

- **Default** – switches all enabled USB-C lines to the single mux port designated by the channel selector
- **Channel priority** – auto-selects the lowest-numbered mux port that has V_{BUS} present. Allows simple automatic host selection
- **Split** – allows each signal type to be independently connected to a mux port. V_{BUS} and CC lines can be assigned to any combination of ports, while USB 2, USB3, and SBU can be assigned to a single mux port. See the [Mux API reference](#) for more detail

System Tab



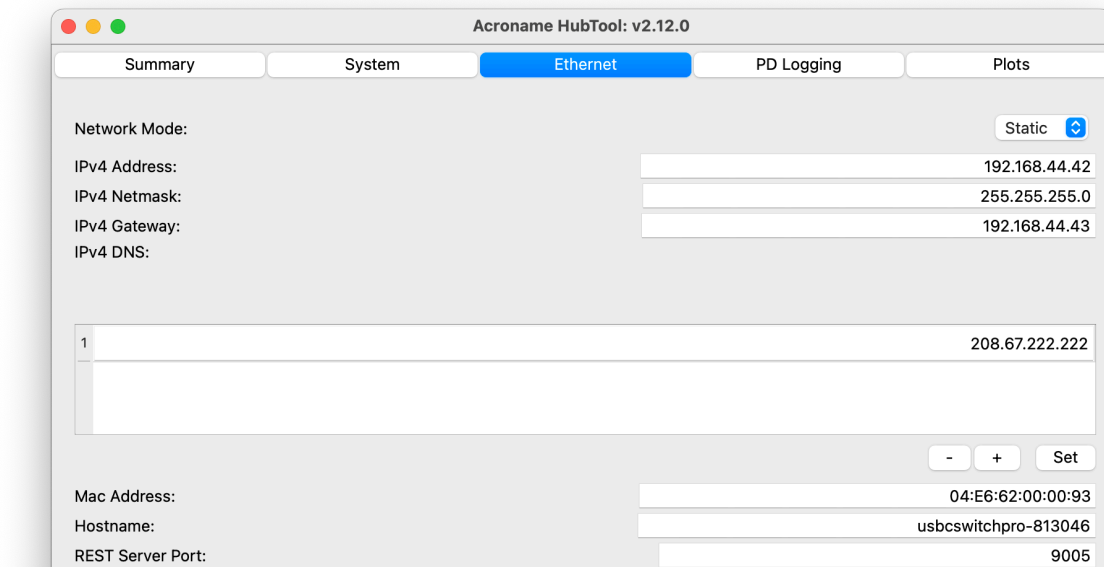
The System Information tab contains information and settings for the switch.

Heading	Meaning
SN	Switch serial number
Model	Switch model name and number, e.g. USBCSwitchPro (25)
Firmware	Currently installed firmware version
Module	Address of the module on the Brainstem network
Voltage	Voltage at the input power source port
Current	Input current at the Control port
Temperature	System temperature
Hardware offset	Increments Brainstem module address by a hardware setting - fixed to 0 in USB-C-Switch Pro
Router	Address of the routing Brainstem module
Software offset	Increment the module address by the selected amount. Click the button to send to the device, Save and Reset to apply
Save	Store select changed settings
Reset	Soft reset of the switch
LED	Toggles blue User LED (person icon next to the control port), used for debugging and to identify the switch

Ethernet Tab

The Ethernet tab shows network settings for the Ethernet connection. For most setups, we recommend a direct Ethernet link between the host test machine and the USBExt3c. By default, the USB-C-Switch Pro acts as a DHCP client and will receive an IP address from a DHCP server. If no server is detected, the USB-C-Switch Pro falls back to a static IP address of 192.168.44.42. In static mode, the host computer interface IP must be set to an address in the 192.168.44.x range. The DHCP client is limited to hosts on the local link and does not operate across network bridges or gateways.

Host firewall rules must allow:



- Outgoing UDP multicast on port 9888
- Incoming UDP responses on port 9889
- Outgoing TCP connections to port 8000
- Incoming / Outgoing TCP connections on ports 9005 and 9006

Network Mode - How the IP address is set:

- None - no network
- DHCP (default) - receive IP from DHCP server
- Static, defaults to 192.168.44.42

IPv4 Address - e.g. 192.168.44.42

IPv4 Netmask - 255.255.255.0

IPV4 Gateway - host computer IP

IPv4 DNS - defaults to 208.67.222.222

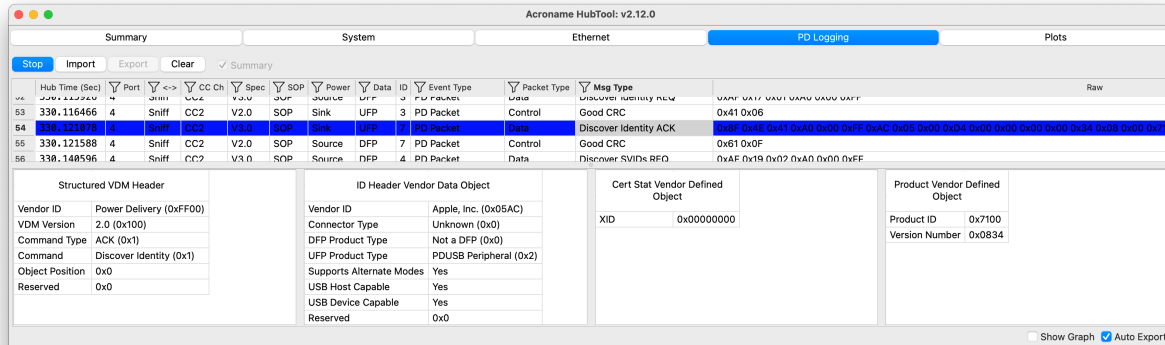
Mac Address - unique to the specific Switch

Hostname - defaults to "usbcs witchpro-[first 7 digits of hub serial]"

REST Server port - REST port on host

PD logging Tab

USB-C-Switch Pro has the ability to sniff and log USB-PD negotiations.



PD logging tab view

PD logging⁸⁵ is a software add-on feature enabling capture, logging, and decoding of USB PD communications on all USB-C ports of the hub, including power negotiations and timing. At the top of the panel are logging controls and port selection toggles.

Stop - stops logging

Import - imports a CSV log file

Export - exports a CSV log file (stop logging before export)

Clear - clears the PD log

Below these controls is the PD packet log. The left axis is the row number, which resets when the log is cleared. Columns can be filtered by clicking the filter icon (🔍) on the column headings:

Hub Time (Sec)	Port	CC Ch	Spec	SOP	Power	Data	ID	Event Type	Packet Type	Msg Type	Raw
----------------	------	-------	------	-----	-------	------	----	------------	-------------	----------	-----

PD packet log headings

Time - clicking on this heading cycles among time references:

- Hub time (s) - time since the hub powered on
- App time (hh:mm:ss) - time since the HubTool App was launched
- System time (yyyy.MM.dd hh:mm:ss:zzzz) - date and time

Port - Mux ports (0-3), Common (4)

<-> - message direction - *RX*, *TX*, or '-' (none)

Spec - USB PD version

SOP* - "Start Of Packet"

- SOP - for messages between source and sink
- SOP' - for messages to the cable connector closest to the downstream-facing port (DFP)
- SOP" - for messages to the cable connector closest to the upstream-facing port (UFP)

⁸⁵ <https://acroname.com/store/t99-pd-log>

Power - *sink, source, none*

Data - *UFP, DFP, '-'*

ID - message ID, (0 – 7), increments with each new message. Acknowledgements should match message ID

Event type - description of the event

Packet type

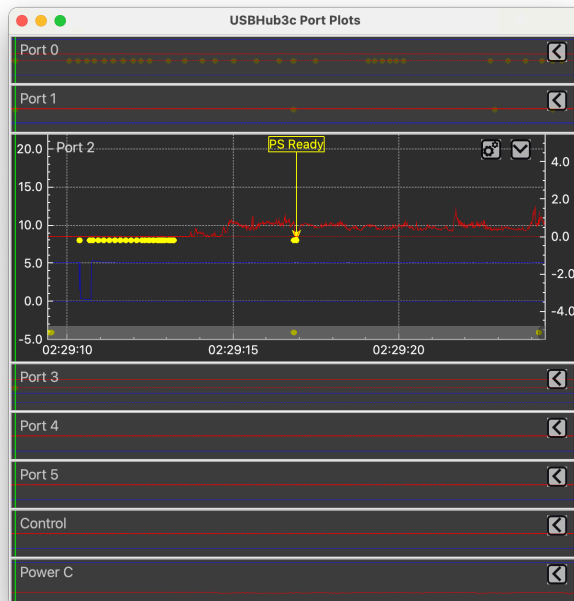
- Control - short messages that typically require no data exchange
- Data - messages contain data objects that are transmitted between devices
- Extended - data messages with larger data payloads

Message type - description of message

Raw - raw PD message (hexadecimal)

Below the packet log:

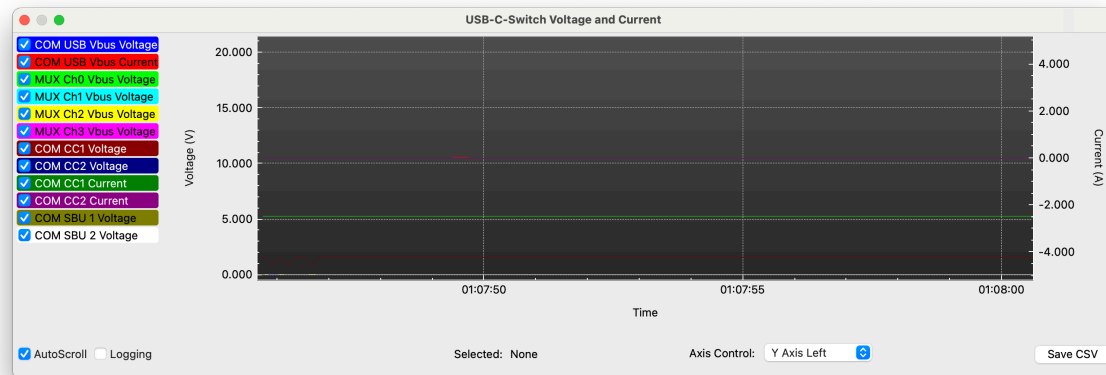
Show Graph toggle - when checked, clicking on an event highlights the corresponding PD packet. Clicking on a yellow PD message in the plot highlights the corresponding message in the log



Graph view with highlighted PD packet

Auto Export toggle - when checked, logs are stored automatically. Destination is visible on mouseover.

Plots Tab



The **Plots** tab displays real-time voltage and current measurements for the **Common** and **Mux** ports, with controls for viewing, logging, and exporting data.

Plot traces

Common Ports

- V_{BUS} — Voltage, Current
- CC1 / CC2 — Voltage, Current
- SBU1 / SBU2 — Voltage

Mux Channels (0–3)

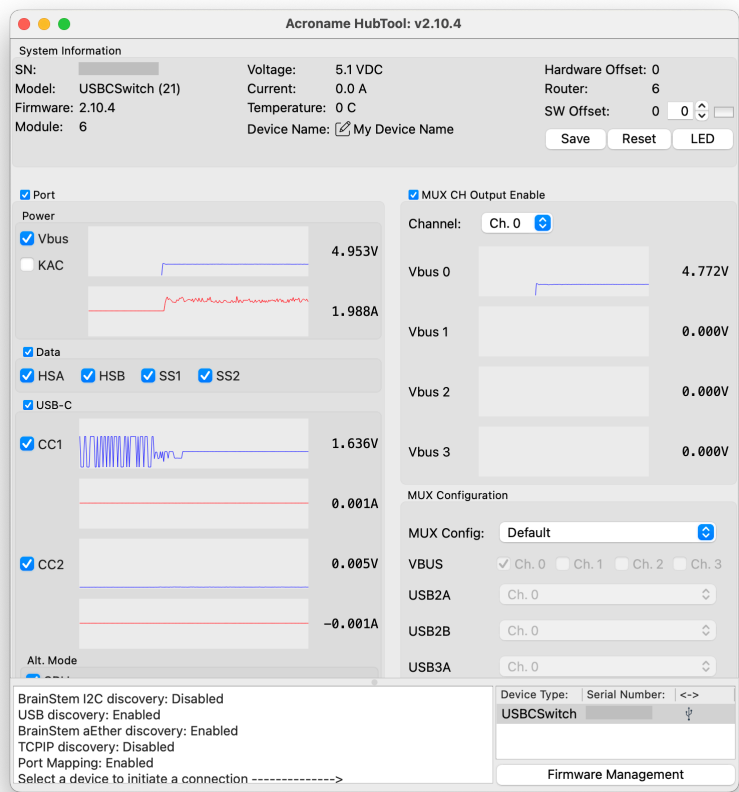
- V_{BUS} — Voltage

Controls

- **Trace Select Toggles** — show or hide individual traces
- **Autoscroll** — keeps the right edge of the plot aligned to “now”; the buffer length defines the visible time window
- **Logging** — records data beyond the visible plot window for later review or export
- **Axis Control** — selects which axis (Voltage, Current, or Time) responds to zoom or drag actions
- **Save CSV** — exports each visible trace as a separate CSV file; if logging is disabled, only current visible buffer is saved

Each tab except *Summary* can be dragged out of the main hubtool window into its own window to allow simultaneous viewing of multiple tabs.

USB-C-Switch



⁸⁶ HubTool interface for

USB-C-Switch

Acroname’s **USB-C-Switch**⁸⁷ is an industrial USB-C switch able to connect one of up to four devices to a host, or one device to one of up to four hosts. It is not a hub – the selected ports form a bidirectional connection and appear “like a cable” to connected devices, even supporting USB alt modes like DisplayPort.

When combined with an Acroname **Universal Orientation Cable**⁸⁸, the USB-C-Switch can emulate a cable flip, allowing tests of all the connections on a port without the need to manually flip the cable.

HubTool presents a unified dashboard to control and view the state of USB-C-Switch.

Connections

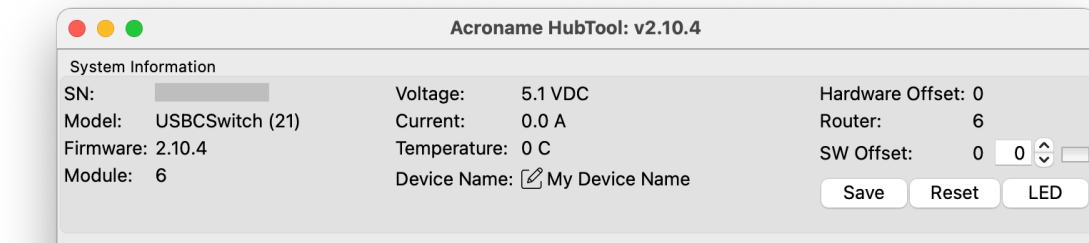
Port	Connects to		
Control	Control computer running HubTool		
Common	Common host	or	Common device
Ports 0 – 3	Switched devices		Switched hosts

⁸⁶ <https://acroname.com/store/programmable-industrial-switch-s85-rdvr-usbcsw>

⁸⁷ <https://acroname.com/store/programmable-industrial-switch-s85-rdvr-usbcsw>

⁸⁸ <https://acroname.com/store/c67-usbc-uoc>

System information



The System Information panel contains information and settings for the switch.

Heading	Meaning
SN	Switch serial number
Model	Switch model name and number, e.g. USBCSwitch (21)
Firmware	Currently installed firmware version
Module	Address of the module on the Brainstem network
Voltage	Voltage at the input power source port
Current	Input current at the Control port
Temperature	System temperature
Hardware off-set	Increments Brainstem module address by a hardware setting - fixed to 0 in USB-C-Switch
Router	Address of the routing Brainstem module
Software off-set	Increment the module address. Requires Save and Reset
Save	Store select changed settings
Reset	Soft reset of the switch
LED	Toggles blue user LED next to the control port, used for debugging and to identify the switch

Common Port controls

The left side of HubTool shows controls and attributes related to:

- Enabling and disabling individual lines
- Viewing voltage and current on VBus, CC1, CC2, SBU1, SBU2
- Keep-alive charging
- Alt mode
- Cable flip
- Equalization (redriver version only)

Acroname HubTool: v2.10.4

System Information

SN: 0x6B10667F Voltage: 5.1 VDC Hardware Offset: 0
Model: USBCSwitch (21) Current: 0.0 A Router: 6
Firmware: 2.10.4 Temperature: 0 C SW Offset: 0 0 0
Module: 6 Device Name: My Device Name Save Reset LED

☒ Port

Power

☒ Vbus 4.813V
☐ KAC 2.932A

☒ Data

☒ HSA ☒ HSB ☒ SS1 ☒ SS2

☒ USB-C

☒ CC1 1.687V
0.000A

☒ CC2 0.000V
-0.001A

Alt. Mode

☒ SBU

SBU1 0.078V
SBU2 0.078V

USB 3.1 Enabled

☐ Cable Flip

☐ CC Flip ☐ SS Flip ☐ SBU Flip ☐ USB2 Flip

Equalizer

USB 2.0

Transmitter Config Receiver Config
40mV Level 1

USB 3.0

Transmitter Config
Mux +1db, Common +0db @ 900mV

Receiver Config: Com Receiver Config: Mux
Level 1 Level 1

☒ MUX CH Output Enable

Channel: Ch. 0

Vbus 0 4.652V
Vbus 1 0.000V
Vbus 2 0.000V
Vbus 3 0.000V

MUX Configuration

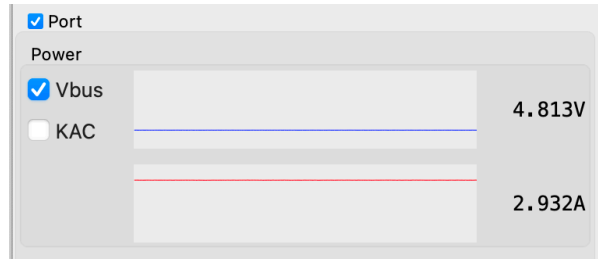
MUX Config: Default

VBUS ☒ Ch. 0 ☐ Ch. 1 ☐ Ch. 2 ☐ Ch. 3

USB2A Ch. 0
USB2B Ch. 0
USB3A Ch. 0
USB3B Ch. 0

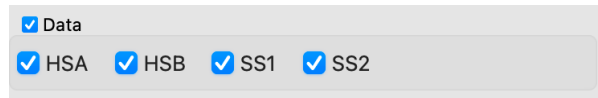
CC1 ☒ Ch. 0 ☐ Ch. 1 ☐ Ch. 2 ☐ Ch. 3
CC2 ☒ Ch. 0 ☐ Ch. 1 ☐ Ch. 2 ☐ Ch. 3
SBU Ch. 0

Power



- **Port toggle** – enables and disables all lines connecting the Common port to the selected mux channel
- **VBus toggle** – enables and disables VBus lines
- **KAC toggle** – enables Keep-alive charging (KAC)
Keep-alive charging helps keep battery-powered devices on the non-selected mux ports charged. When enabled, the KAC circuit connects power from the control port VBus to all non-selected mux channel VBus lines. See the *API reference <cs-kac>* for more detail
- **VBus voltage and current plots** – shows voltage and current for VBus. Click to pop up the *Voltage and Current plot window*

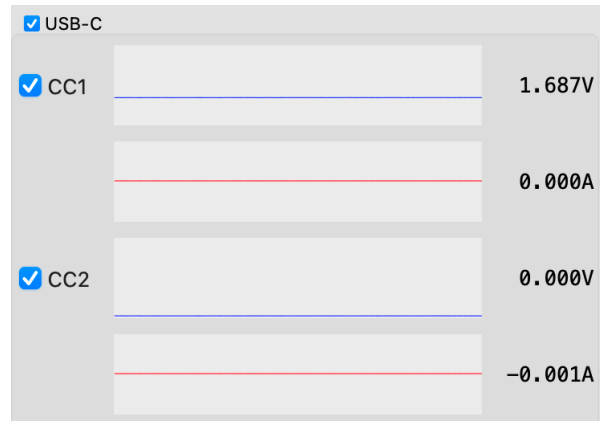
Data line toggles



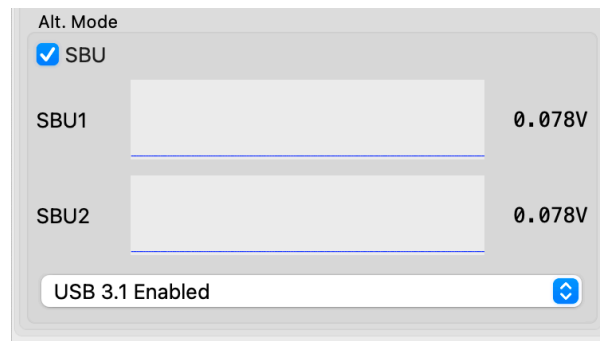
- **Data toggle** – enables and disables all USB data lines
- **HSA, HSB, SSA, SSB toggles** – enable and disable USB 2 (HS) and USB 3 (SS) data lines on side A or side B independently

CC

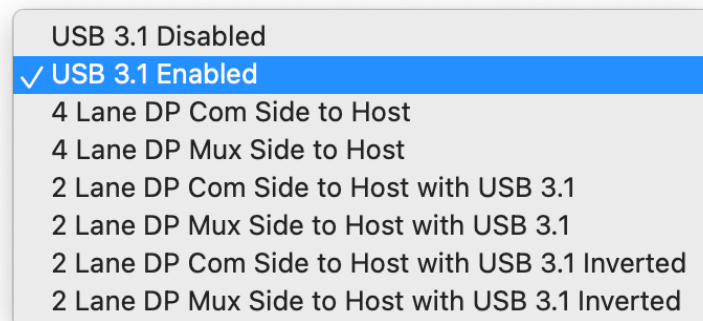
- **USB-C toggle** – enables and disables CC and SBU lines
- **CC toggles** – enable and disables CC1 and CC2 lines independently
- **CC1 and CC2 voltage and current plots** – shows voltage and current for CC1 and CC2. Click to pop up the *Voltage and Current plot window*



Alt mode



- **SBU toggle** – enables and disables SBU lines. Used for Alternate Mode discovery, negotiation, and configuration data exchange
- **SBU1 and SBU2 voltage and current plots (redriver model only)** – shows voltage and current for SBU1 and SBU2. Click to pop up the [Voltage and Current plot window](#)
- **DisplayPort alt mode configuration menu (redriver model only)**



Alt Mode Configuration
USB 3.1 Disabled – no SS lines connected
USB 3.1 Enabled – SS lines connected
4-Lane DisplayPort – no USB 3.1 – Host on Common Port
4-Lane DisplayPort – no USB 3.1 – Host on Mux Port
2-Lane DisplayPort + USB 3.1 -- Host on Common Port
2-Lane DisplayPort + USB 3.1 -- Host on Mux Port
2-Lane DisplayPort + USB 3.1 Inverted – Host on Common Port
2-Lane DisplayPort + USB 3.1 Inverted – Host on Mux Port

DisplayPort Alt modes use the SS lines for DisplayPort data, but change their direction. Since each redriver line can operate in only one direction at a time, the redriver direction needs to be set to match the mode being used by the host and display. See the :ref: *API reference <cs-alt-mode-configuration>* for more detail.

Cable flip

- **Cable flip toggle** – switches USB data, VConn, and SBU lines from side A to side B as if the cable had been flipped. When using standard USB-C cables with USB-C-Switch, one cable orientation will work, the other will need to be flipped physically, or by using the toggle.
- **CC, SS, SBU, USB2 flip toggles** – individually flip each connection type

To enable automated cable flips for testing, use one [Universal Orientation Cable](https://acroname.com/store/c44-usb-uoc)⁸⁹ for either the host or mux connection, and one standard cable for the other connection.

Equalizer (requires redriver model)

The equalizer section provides controls to set transmitter and receiver gains for USB 2 and USB 3 data lines in both directions.

USB 2.0

- **Transmitter config** – selects the amount of DC boost applied to USB 2.0 (HS) signals. USB Low Speed and Full Speed signals are unaffected. When boost is set to 0 mV, the HS redriver is disabled independent of receiver configuration

⁸⁹ <https://acroname.com/store/c44-usb-uoc>

Equalizer

USB 2.0

Transmitter Config Receiver Config

40mV Level 1

USB 3.0

Transmitter Config

Mux +1db, Common +0db @ 900mV

Receiver Config: Com Receiver Config: Mux

Level 1 Level 1

USB 2.0 transmitter DC boost

40 mV – default
 60 mV
 80 mV
 0 mV (redriver disabled)

- **Receiver config** – controls the sensitivity of the redriver to incoming HS signals by boosting higher frequencies to make sharper edges. When receiver config is set to level 0, the redriver is disabled independent of transmitter configuration

USB 2.0 Receiver Equalization

Level 1 – moderate boost – default
 Level 2 – higher boost
 Level 0 – redriver disabled

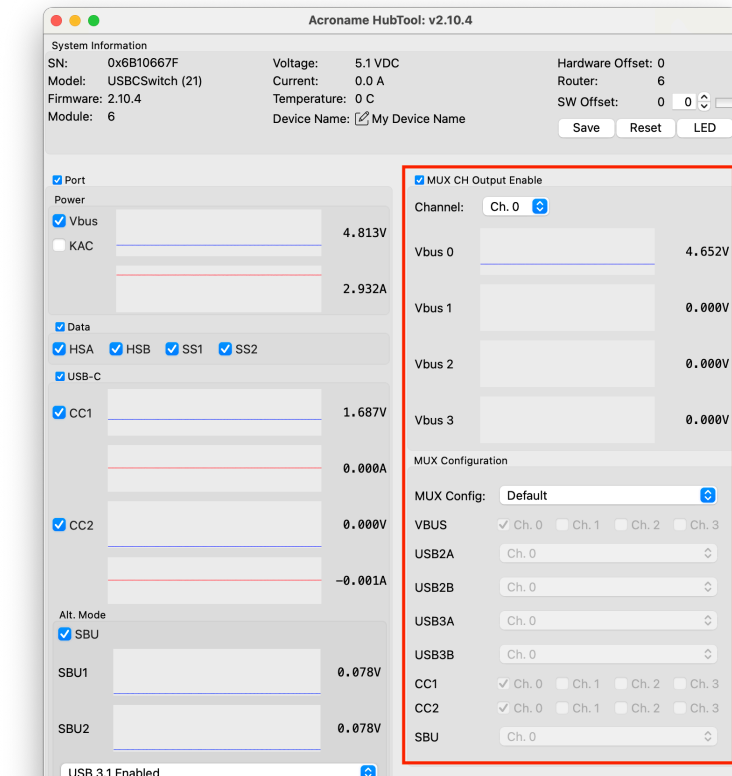
USB 3.0

- **Transmitter config** – selects preset combinations of transmitter gains for each direction of the full-duplex USB 3 (SS) data lines, and peak-to-peak voltage for both directions

Mux Side	Common Side	Range	
1 db	0 db	900 mVpp	default
0 db	1 db	900 mVpp	
1 db	1 db	900 mVpp	
0 db	0 db	900 mVpp	
0 db	0 db	1100 mVpp	
1 db	0 db	1100 mVpp	
0 db	1 db	1100 mVpp	
2 db	2 db	1100 mVpp	
0 db	0 db	1300 mVpp	

- **Receiver config: com** – sets the sensitivity of the redriver to incoming SS signals on the com side by boosting higher frequencies to make sharper edges. Level 1 (lowest boost, default) to Level 16 (highest)
- **Receiver config: mux** – sets the sensitivity of the redriver to incoming SS signals on the mux side by boosting higher frequencies to make sharper edges. Level 1 (lowest boost, default) to Level 16 (highest boost)

Mux Port controls



The right side of HubTool shows controls and attributes related to the mux ports.

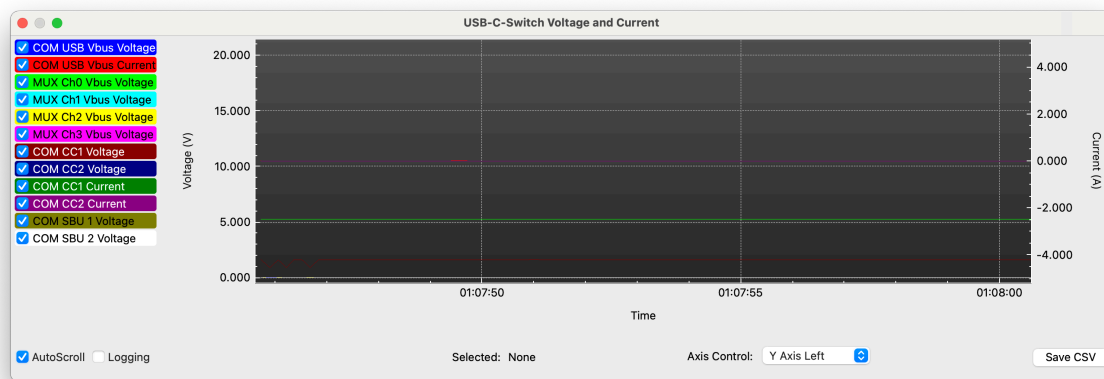
Mux Channel control

- **Mux channel output enable** – toggles the output of the selected port
- **Mux channel selector** – designates one mux port to connect to the Common port. Unavailable in Channel priority and Split configuration
- **VBus voltage plots** – shows VBus voltage for each mux port. Click the graph to pop up the [Voltage and Current plot window](#)

Mux configuration

- **Default** – switches all enabled USB-C lines to the single mux port designated by the channel selector
- **Channel priority** – auto-selects the lowest-numbered mux port that has VBus present. Allows simple automatic host selection (requires USB A-to-C cables)
- **Split** – allows each signal type to be independently connected to a mux port. VBus and CC lines can be assigned to any combination of ports, while USB2, USB3, and SBU can be assigned to a single mux port. See the [API reference](#)⁹⁰ for more detail

Voltage and current plot window

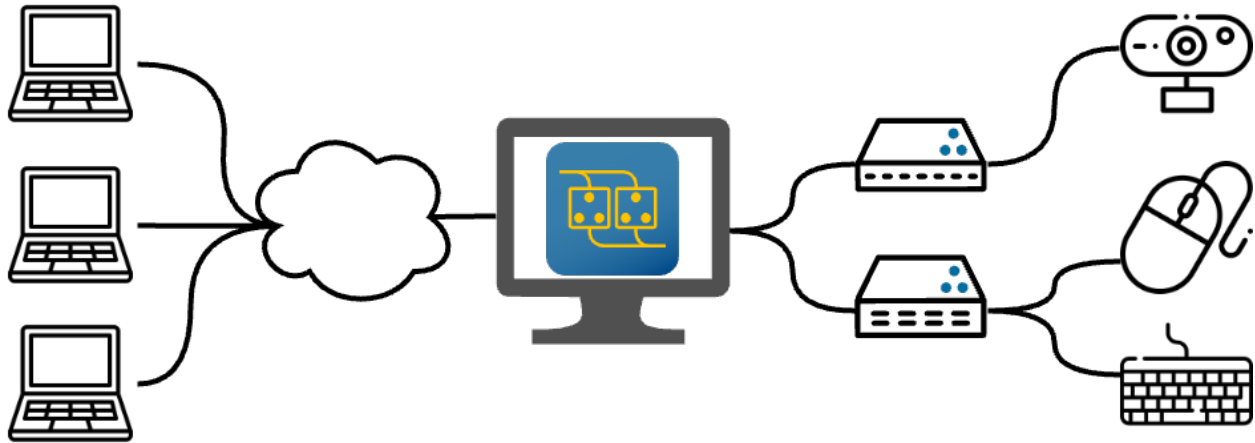


The voltage and current plot window pops up when any of the traces in the main window are clicked.

- **Trace select toggles** – show and hide each trace
- **Autoscroll** – scrolls automatically. When enabled, the right edge is “now” – default = on
- **Logging** – when enabled, data is stored beyond the current plot view
- **Axis control** – sets whether dragging and scrolling zoom affect the voltage, current, or time axis
- **Save CSV** – saves data for each non-hidden trace as an independent CSV file. If logging is not enabled, only the data visible in the graph is saved

⁹⁰ <https://acroname.com/reference/devices/usbcswitch/functionality.html#mux-split-mode>

2.2 BrainD



BrainD is a server application that provides simultaneous multi-client access and control to Acroname devices connected to a host computer. Connected devices can be accessed without drivers via any intranet or internet connection, including tunneling over VPNs. Using a standard HTTP request interface, BrainD serves a REST-Ful API that provides telemetry and status information on the entire USB tree. BrainD builds upon decades of experience from successfully deploying complex test and measurement systems, conference room solutions, and artificially intelligent robotic applications.

2.2.1 Installation

Procedure

Please ensure you have appropriate privileges on your computer and back up any critical data to prevent data loss during the upgrade process.

1. Download latest software release.
 1. Navigate to the [Acroname Download Center](https://acroname.com/software/braind)⁹¹.
 2. Choose the appropriate software for your operating system and click the link to initiate the download.
 3. If prompted to choose a download location, select a place that is easily accessible.
 4. Wait for the download to complete.
2. Installation of BrainD
 1. Windows:
 1. The downloaded file will be a setup .exe file. Open the .exe file to run the program installer.
 2. If BrainD is already installed, the installer will prompt to uninstall the existing version.
 3. Follow the installer prompts to accept the license and select the application installation directory.
 4. Open BrainD from the Start Menu or Windows Explorer.
 2. macOS

⁹¹ <https://acroname.com/software/braind>

1. The downloaded file will be a .dmg disk image. Open the .dmg file to mount the disk image.
2. Drag the BrainD application into the Applications folder inside the disk image
3. Open BrainD from the Applications folder.
3. Linux

1. The downloaded file will be a .deb installer package.
2. In a command line, run `sudo dpkg -i <image file>.deb`. This will install the BrainD application.

Note: If dependencies are missing, the above command may fail. Run `sudo apt-get -f install` to install the dependencies, then re-run the previous install command.

3. After installation, open BrainD from the applications selector or from the command line.

Note: On Linux, the `api/v1/devices` REST endpoint may not enumerate all devices on the system if the user does not have write permissions to `/dev/bus/usb`.

Software Requirements

Web Browsers

The following web browsers are supported for accessing and using the web-based application:

Table 43: Supported Web Browsers

Browser	Version	Windows	macOS	Linux	iOS	Android
Safari	Latest stable	Yes	Yes	Yes	Yes	No
Brave	Latest stable	Yes	Yes	Yes	Yes	Yes
Microsoft Edge	Latest stable	Yes	Yes	No	No	No
Google Chrome	Latest stable	Yes	Yes	Yes	Yes	Yes
Mozilla Firefox	Latest stable	Yes	Yes	Yes	Yes	Yes

Operating Systems

BrainD application is compatible with the following operating systems:

Table 44: Supported Operating Systems

Operating System	Version	BrainD	ControlRoom
Windows (x86/x86_64)	10	Yes	Yes
	11	Yes	Yes
macOS (Intel/Apple Silicon)	10.15	Yes	Yes
	11.x	Yes	Yes
	12.x	Yes	Yes
	13.x	Yes	Yes
Linux (x86_64)	Ubuntu 20.04 LTS	Yes	Yes
	Ubuntu 22.04 LTS	Yes	Yes
Linux (arm64)	Ubuntu 20.04 LTS	Yes	Yes
Linux (armhf)	Debian Bullseye	Yes	Yes

Please note that while our web-based application may function on earlier versions of the mentioned operating systems, we strongly recommend using the specified minimum versions for optimal performance, security, and compatibility.

2.2.2 Quick Start Guide

To get started with BrainD, follow these steps:

1. Install BrainD following the [Installation Guide](#).
2. Connect an Acroname product with a valid Software License installed. For this guide, a [USBHub3c](#) will be used, with serial number F7A9DFC6.
3. Start BrainD by selecting it in the Start Menu under “Acroname” (Windows) or from the Applications directory (Mac). An icon will appear in the system taskbar.
4. The BrainD service is now running. The [RESTful API](#) is available at `http://127.0.0.1:9005`. See the following sections for example usage. Additional examples are in the documentation for each RESTful endpoint.

GET Example - Get Version

The version of software that is running can be read by issuing a GET request to the `version` endpoint.

Bash

```
curl -H 'Accept: application/json' http://127.0.0.1:9005/api/v1/version
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/version')
json_data = response.json()
print(json.dumps(json_data, indent=3))
```

The resulting return of the version will return the following JSON body.

```

1 {
2   "braind": {
3     "version": {
4       "major": 1,
5       "minor": 0,
6       "patch": 0
7     },
8     "buildDate": "Thu Aug 24 13:58:24 2023",
9     "buildHash": "d38f0a1dd5f8daa2c8cd0a0170fee8210d1356b4"
10  },
11  "brainstem": {
12    "version": {
13      "major": 2,
14      "minor": 10,
15      "patch": 0
16    },
17    "buildDate": "2023-09-08T19:03:10Z",
18    "buildHash": "7af9c07e44601d6987fe3dab0ea201a13609683e"
19  }
20 }

```

PUT Example - Set BrainStem Port Off

BrainD is designed to manage and manipulate Acroname products through the *RESTful API*

Bash

```

curl -X PUT http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/1/enabled -H
  ↪ 'Content-Type: application/json' -d '{"value": 0}'

```

Python

```

import requests
import json
response = requests.put('http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/1/
  ↪enabled', json={'value': 0})
json_data = response.json()
print(json.dumps(json_data, indent=3))

```

The resulting return of the version will return the following JSON body.

```

1 {
2   "timestamp": "2023-09-18T04:43:09.813Z",
3   "request": {
4     "endpointName": "/api/v1/brainstem/F7A9DFC6/port/1/enabled",
5     "parameters": {
6       "value": 0
7     }
8   },
9   "response": {}
10 }

```

GET Example - Get BrainStem Port 2 Vbus Voltage

Bash

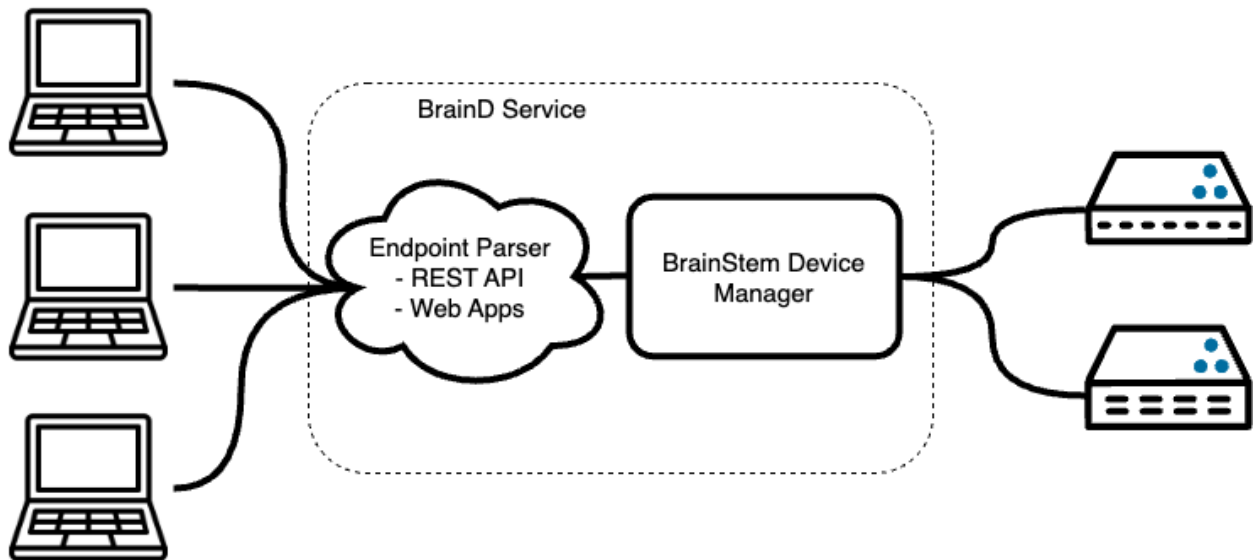
```
curl http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/2/VbusVoltage
```

Python

```
import requests
import json
response = requests.put('http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/2/
↳VbusVoltage')
json_data = response.json()
print(json.dumps(json_data, indent=3))
```

```
1 {
2   "timestamp": "2023-09-18T04:56:16.639Z",
3   "request": {
4     "endpointName": "/api/v1/brainstem/F7A9DFC6/port/2/VbusVoltage",
5     "parameters": {}
6   },
7   "response": {
8     "value": 5100097,
9     "rawValue": 5100097
10  }
11 }
```

2.2.3 Usage



BrainD has multiple methods of interface and interacting with a consumer or client. A host processor must host the BrainD service for endpoints and user applications to run. The BrainD service will inherently manage all links to BrainStem devices that are connected to the host processor with the use of *aEther*. Also, BrainD service creates specific *RESTful endpoints* for connected devices, bulk data endpoints, and web based application endpoints.

Background Service

BrainD user interface application enables simple configuration and manipulation of the application service.

Taskbar

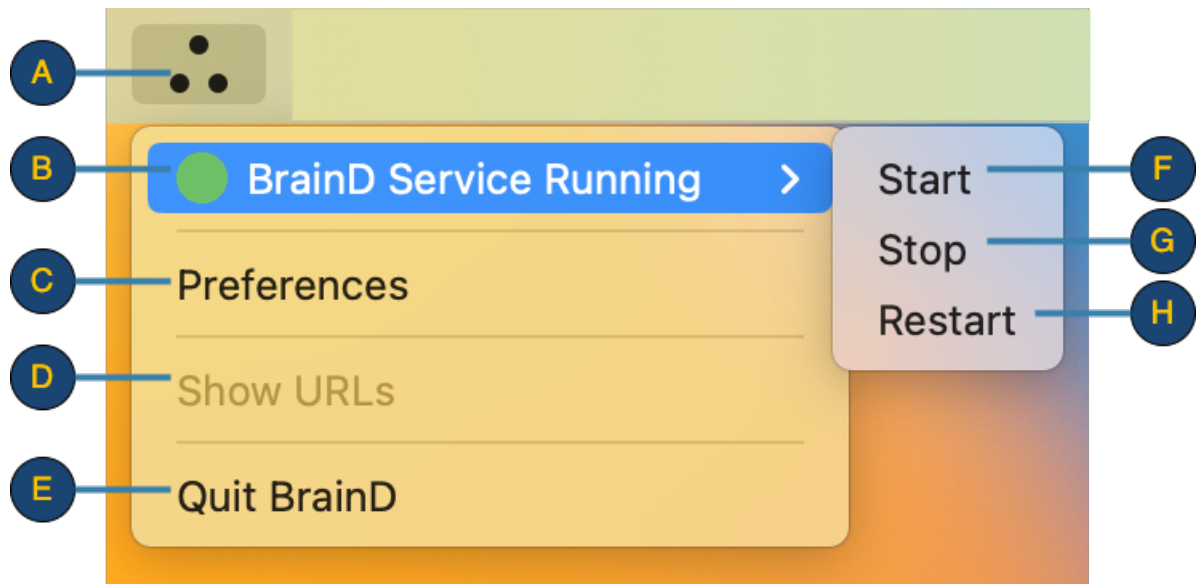


Table 45: Taskbar Details

Item	Description
A	Application icon.
B	BrainD operation status. A green indicator shows the service is running. A red indication shows the service is stopped and no application connections or updates will be available.
D	Opens the settings and configuration dialog .
E	Close and quit the BrainD service.
F	Start the BrainD service.
G	Stop the BrainD service.
H	Restart the BrainD service. All active connections will be terminated.

Command Line

For headless servers, starting as a system service, or as part of a larger solution, BrainD may be directly run from a command line, without the use of the taskbar GUI application.

The list of command line arguments may be found with the `--help` argument:

```
$ braind_service --help

BrainD Version 1.0.0
Service providing a REST interface to Acroname BrainStem devices

Application Usage:
```

(continues on next page)

(continued from previous page)

```
braind_service [-f <filename>]... [-d <json object>]
```

Parameters:

-h, --help	- Show full help message
-v, --version	- Show version information
-f, --config-file <file>	- Specifies any additional configuration files that override the default setting. May be specified multiple times for multiple files.
-d, --config-data <json>	- JSON Object containing configuration to override the default settings

The `-f` flag is used to specify an additional configuration file that will be read, and this may be specified multiple times.

The `-d` flag allows a JSON configuration object to be passed directly to the service, which will take priority over all configuration files.

The details of the configuration is documented in [Configuration](#).

Examples

For endpoint usage examples, please refer to the [RESTful API](#) section which will have additional information.

2.2.4 Security

SSL Certificates

SSL (Secure Sockets Layer) is a fundamental technology for securing web applications by encrypting data, ensuring data integrity, authenticating servers, and enhancing user trust.

SSL layers encrypt the data transmitted between a user's web browser and server. This encryption ensures that the data remains unreadable and secure even if intercepted by malicious actors. SSL includes mechanisms to ensure data integrity during transmission using cryptographic hash functions to detect unauthorized modifications or tampering. In the event of altered data during transit, the recipient can see and reject the modified data.

BrainD allows loading a user's SSL certificate through the [settings and configuration dialog](#). Many external service providers can provide an SSL certificate; one may already have an SSL certificate for BrainD installations.

Use of a self-signed certificate may cause a warning to appear when accessing the REST endpoints. This is caused by the browser failing to authenticate the received certificate against a trusted certificate authority.



This Connection Is Not Private

This website may be impersonating "127.0.0.1" to steal your personal or financial information. You should go back to the previous page.

Go Back

Safari warns you when a website has a certificate that is not valid. This may happen if the website is misconfigured or an attacker has compromised your connection.

To learn more, you can [view the certificate](#). If you understand the risks involved, you can [visit this website](#).

OpenSSL is an option that one can use to create a self-signed SSL certificate. The following command may be used to generate a self-signed certificate:

Bash

```
openssl req -new -x509 -days 365 -nodes -keyout example.key -out example.crt
```

By default, BrainD will serve an insecure web page over HTTP, without any certificates or encryption. If insecure communications over HTTP are not desirable, the [settings and configuration dialog](#) can be used to enable HTTPS/SSL communication. If a certificate is not specified, then BrainD will automatically generate a self-signed certificate.

Certificates are stored in the following directory:

1. Windows: %AppData%\Acroname\BrainD\config
2. Mac: ~/Library/Application\ Support/Acroname/BrainD/config
3. Linux: ~/.acroname/BrainD/config

2.2.5 Configuration

Preferences

To change the default settings of BrainD, the desktop application provides a simple dialog window that enables users to change the HTTP server settings and specify [SSL/TLS Certificates](#). To open the dialog, click on the taskbar menu and select "Preferences".

The following diagram lists the available configuration options in the Settings dialog:

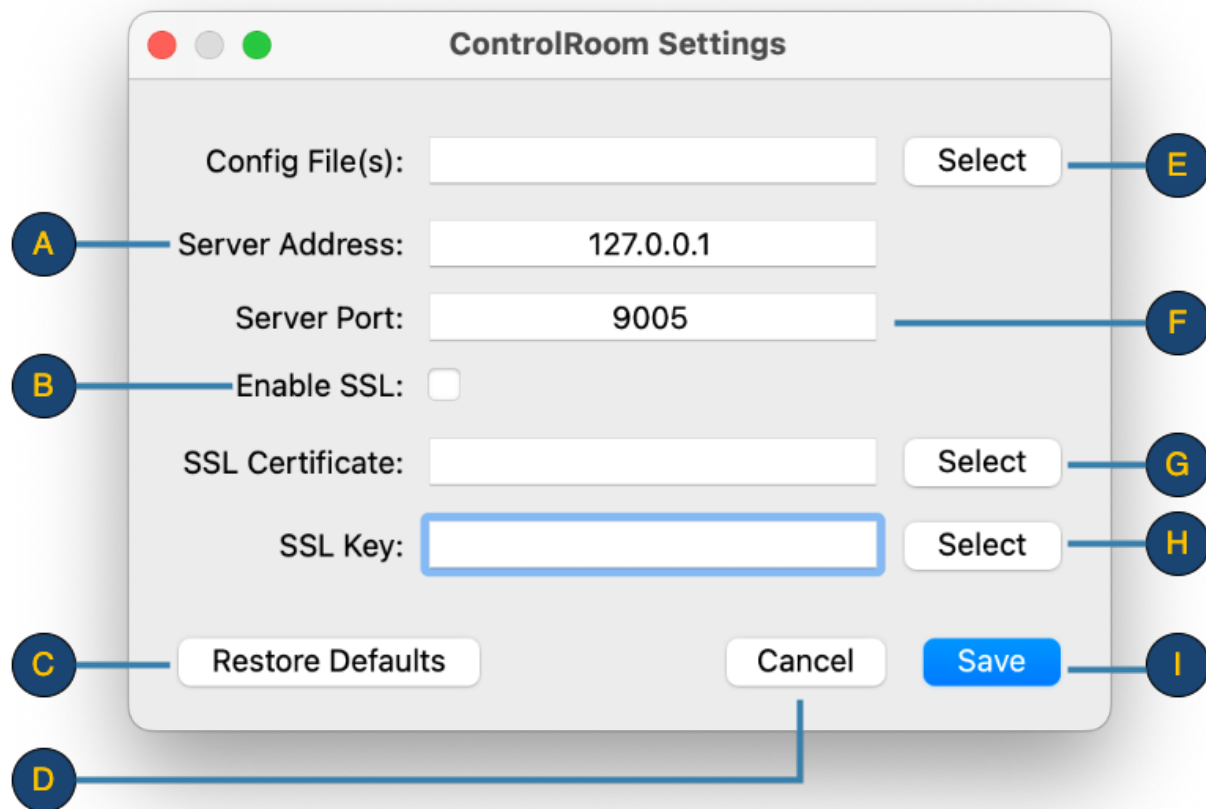


Table 46: Preferences Details

Item	Description
A	Default server address. Default is 127.0.0.1, which will allow connection from external devices.
B	Enable SSL/TLS on client connections.
C	Restore configuration settings to factory defaults.
D	Close the preferences window without saving.
E	Optional path to a JSON Configuration File.
F	Server port to bind to for clients. Default is 9005.
G	SSL/TLS Certificate (.cert) file . If not specified, a self-signed cert will be generated.
H	SSL/TLS Key (.key) file . If not specified, a self-signed cert will be generated.
I	Save application settings.

Configuration Files

BrainD allows for more fine-grained control of the backend service through the use of JSON configuration files. This configuration includes settings for *logging*, polling rates, and web server configuration.

The BrainD service will read the following configuration file for the current operating system:

1. Windows: %AppData%\Acroname\BrainD\config\config.json
2. Mac: ~/Library/Application Support/Acroname/BrainD/config/config.json
3. Linux: ~/.acroname/BrainD/config/config.json

Each entry in the `config.json` file will be merged into the default configuration object, overriding each field that is listed. In addition, if any configuration files are specified in the “Config File(s)” entry of the Settings menu, these files will be read and merged as well. The default configuration object is defined as the following:

```
{
  "logging": {
    "loggers": {
      "default": "debug"
    },
    "sinks": {
      "consoleSinks": [
        {
          "enable": true,
          "level": "debug",
          "enableColors": true,
          "colorPattern": "\u001B[0m[\u001B[32m%Y-%m-%d %H:%M:%S.%e\
\u001B[0m] [%^%l%$] [\u001B[35m%n\u001B[0m:\u001B[33m%#\u001B[0m] %v",
          "noColorPattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v"
        }
      ],
      "fileSinks": [
        {
          "enable": true,
          "level": "info",
          "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
          "path": {
            "win": "%AppData%\Acroname\BrainD\log",
            "mac": "~/Library/Logs/Acroname/BrainD",
            "lin": ~/.acroname/BrainD/log"
          },
          "baseFileName": "braind.log",
          "maxFiles": 5,
          "maxFileSize": 5242880
        }
      ]
    },
    "flushInterval": 3000
  },
  "httpServer": {
    "enable": false,
    "address": "0.0.0.0",
    "port": 9005,
    "threadCount": 1
  },
  "httpsServer": {
    "enable": true,
```

(continues on next page)

(continued from previous page)

```

        "address": "0.0.0.0",
        "port": 9006,
        "threadCount": 1,
        "sslCert": "",
        "sslKey": ""
    },
    "pollingRates": {
        "acronameDevicesState": 300,
        "devices": 300,
        "brainstem": 1000
    }
}

```

Note: The `config['httpServer']` and `config['httpsServer']` configuration objects will have no effect if the BrainD desktop application is used. The value in the Settings dialog takes priority.

Examples

Add an additional log file output:

```

{
    "logging": {
        "sinks": {
            "fileSinks": [
                {
                    "enable": true,
                    "level": "info",
                    "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
                    "path": {
                        "win": "C:\\\\Users\\username\\Desktop",
                        "mac": "~/Desktop",
                        "lin": "~/Desktop"
                    },
                    "baseFileName": "custom-log-file.log"
                }
            ]
        }
    }
}

```

Increase the log flush rate from 3000ms to 500ms, and reduce the USB tree polling rate from 300ms to 1000ms:

```

{
    "logging": {
        "flushInterval": 500
    },
    "pollingRates": {
        "devices": 1000
    }
}

```

2.2.6 Logging

BrainD generates runtime logging to aid in debugging and understanding for many events. These log files are stored by default in the following path for each operating system:

1. Windows: %AppData%\Acroname\BrainD\log
2. Mac: ~/Library/Logs/Acroname/BrainD
3. Linux: ~/.acroname/BrainD/log

By default, these log files will rotate when they reach 5MB in size, up to a maximum of five log files. After this point, old log files will be removed to make room for new ones.

For information on how to adjust the logging parameters or specify new log files, see [Configuration](#).

2.2.7 Platform Specific Considerations

Configure MacOS to Advertise Its Hostname

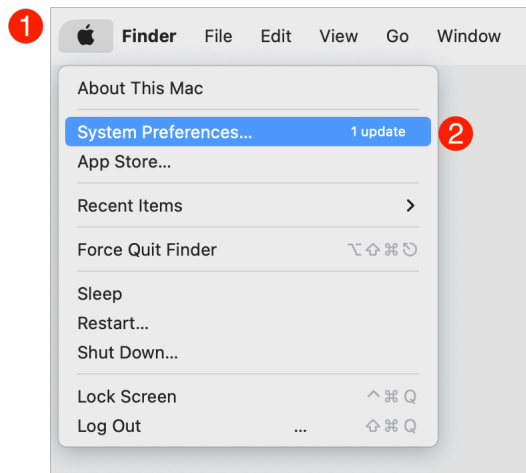
Name resolution is a process where a readable name, e.g. `ElizaPC`, can be resolved into a numerical IP address, e.g. `192.168.41.75`. Name resolution eases server connections, by allowing usage of URLs like `https://ElizaPC` instead of a hard-to-remember, and fragile, `https://192.168.41.75`.

MacOS, in default configuration, doesn't provide resolution for the host's own name.

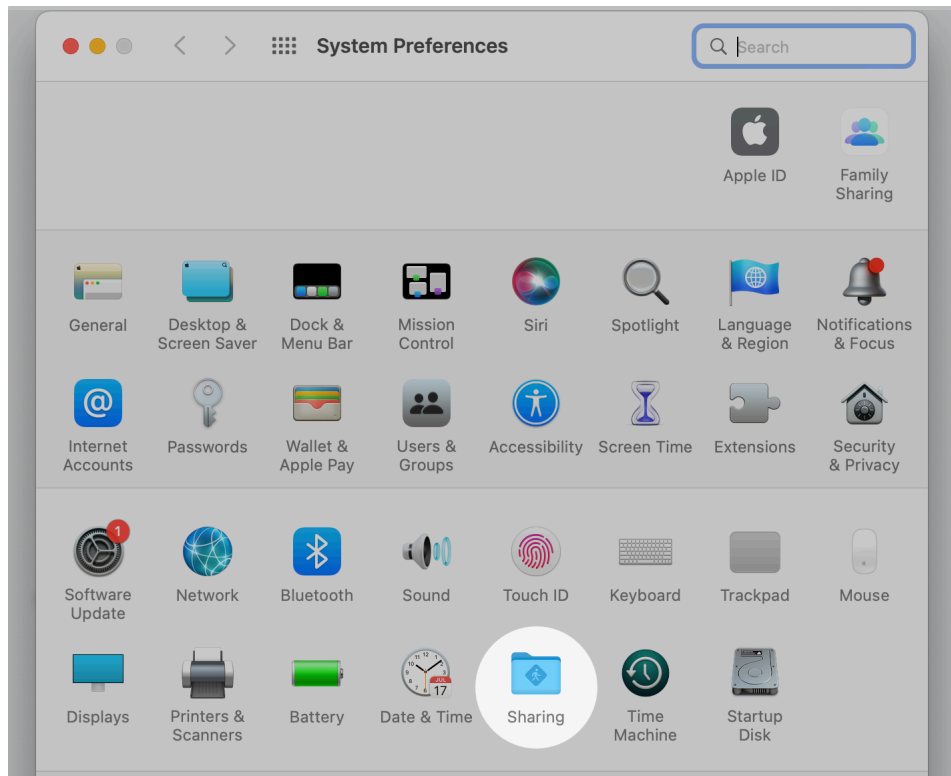
Here's how to fix that.

1. Get to System Preferences via the Apple Menu in the top left of the screen.

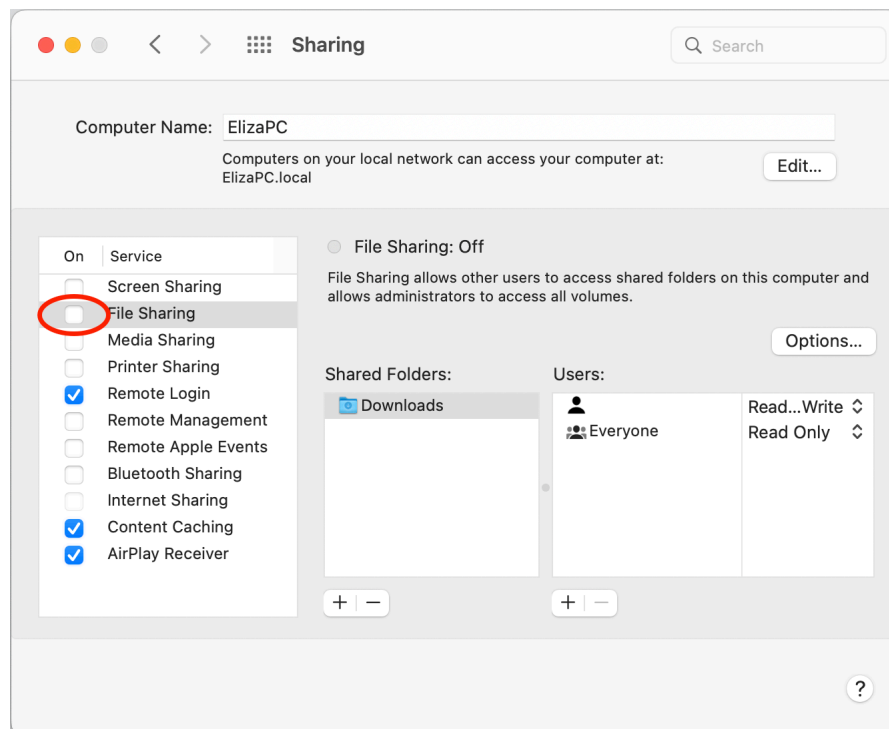
Top Left of the Screen



2. Go to the "Sharing" Panel



3. Enable File Sharing by checking this box. Other devices on the network can now resolve this machine's name. Note: File sharing does not have to remain enabled but must be enabled at least once for name resolution.



4. Configure these settings to your application's needs. This configuration is just for pictorial example. Other devices on the network can still resolve this machine's name, if File Sharing was ever enabled.

2.2.8 Functionality and Features

1. Full remote control of an Acroname device.
2. Telemetry and status information on the host computer's entire usb tree.
3. Simultaneous access from multiple client devices.
4. Extends Acroname's Brainstem protocol, the shared core of all Acroname products.
5. Convenient JSON response format.
6. Mirrors the existing Brainstem interface providing users a new, convenient way to control their devices.

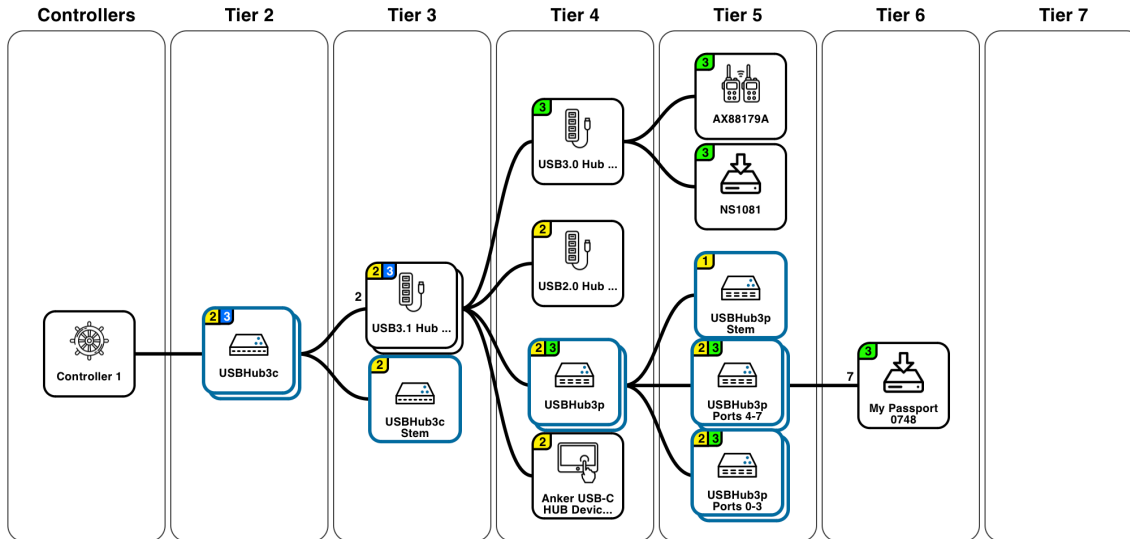
2.2.9 Audio-Video Conferencing Solutions

1. **Peripheral Management:** BrainD can efficiently handle USB peripherals such as microphones, cameras, and speakers.
2. **Enhanced Quality:** Proper management of USB devices can lead to improved audio and video quality during conferences. BrainD can highlight bandwidth allocation and ensure that devices and installations work optimally.
3. **Remote Diagnostics:** Troubleshooting issues with connected devices becomes more straightforward with BrainD by providing diagnostic information and assisting in resolving technical problems remotely.

2.2.10 Test and Measurement Applications

1. **Flexibility:** The ability to dynamically add or remove devices based on specific testing needs enhances flexibility. Engineers can adapt their setups quickly and efficiently, responding to changing requirements without significant disruptions.
2. **Data Logging and Analysis:** BrainD can facilitate real-time data logging and analysis, allowing research, development, and quality control applications to monitor measurements, perform in-depth analysis, and visualize results conveniently.

2.3 ControlRoom



ControlRoom is a service and web application that lets you control, monitor, and reset USB ports in connected conference rooms through a browser. ControlRoom gives a real-time view of your system's connected USB devices for easy troubleshooting and to ensure correct installation. Remotely resolve issues that would normally require a physical unplug / plug cycle.

2.3.1 Installation

To install ControlRoom, download the appropriate [installer](#)⁹² for your operating system.

Requirements

ControlRoom requires a supported Acroname hub with the [ControlRoom add-on software feature](#)⁹³:

- [USBHub3c](#)⁹⁴ – USB-C Hub with Power Delivery Analyzer + Tester
- [USBHub3p](#)⁹⁵ – Programmable Industrial 8-port USB 5Gbps Hub
- [USBHub2x4](#)⁹⁶ – Industrial Intelligent 4-port Hi-Speed USB Hub

⁹² <https://acroname.com/controlroom>

⁹³ <https://acroname.com/store/controlroom-feature>

⁹⁴ <https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro>

⁹⁵ <https://acroname.com/store/programmable-industrial-hub-s79-usbhub-3p>

⁹⁶ <https://acroname.com/store/industrial-intelligent-4-port-hub-s77-usbhub-2x4>

Supported operating systems

ControlRoom binaries are available for the following platforms:

- Windows 10 and higher
- Intel and Apple Silicon Macs running macOS 10.15 or higher
- Linux:
 - x86_64 Ubuntu LTS 20.04, 22.04
 - arm64v8 Ubuntu LTS 20.04
 - armhf Debian Bullseye

While ControlRoom may work on earlier OS versions, support is limited to the listed versions.

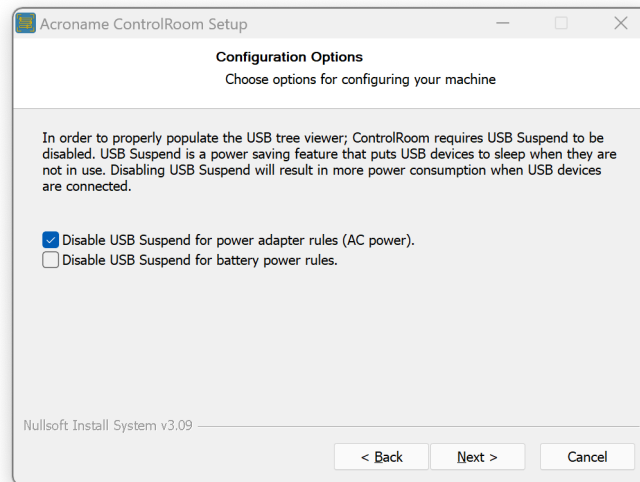
Supported browsers

The ControlRoom web application supports the latest stable versions of Safari, Brave, Edge, Chrome, and Firefox.

Install and launch

Windows

- Open the downloaded .exe file to launch the installer
- Click “Run” in the security popup
- Follow the installer prompts to accept the license



- To reliably view the USB tree, follow the prompt to disable USB suspend
- Choose an install location and start menu folder

Mac

- Open the downloaded .dmg disk image file

- Drag the ControlRoom app into the applications folder inside the disk image



Linux

- The downloaded file will be a .deb software package
- In a command line, run `sudo dpkg -i <image file>.deb` to install the ControlRoom application

Note: If dependencies are missing, the above command may fail. Run `sudo apt-get -f install` to install the dependencies, then re-run the previous install command.

- After installation, open ControlRoom from the applications selector or from the command line


Note: On Linux, the USB Tree may not enumerate all devices on the system if the user does not have write permissions to `/dev/bus/usb`.

2.3.2 Usage

Getting Started

To get started with ControlRoom, first confirm that it's *installed*, then connect a supported Acroname device with a valid [ControlRoom License](https://acroname.com/store/controlroom-license)⁹⁷. This guide will use a [USBHub3c](https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro)⁹⁸.

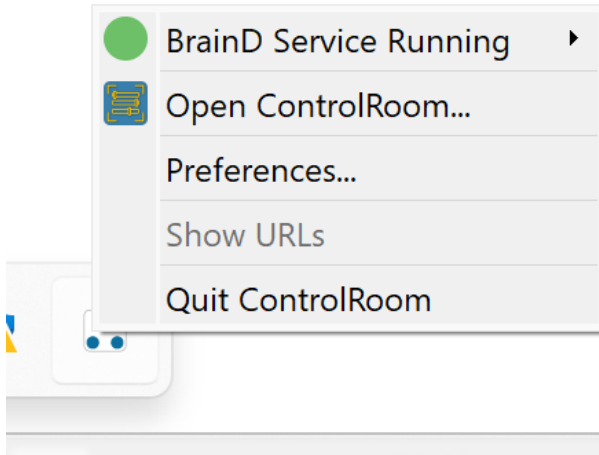
Windows

- Start ControlRoom by selecting it in the Start Menu under “Acroname”
- Acroname’s 3-dot icon () will appear in the System Tray, or in Hidden Icons
- Right click to pop up the ControlRoom menu

⁹⁷ <https://acroname.com/store/controlroom-feature>

⁹⁸ <https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro>

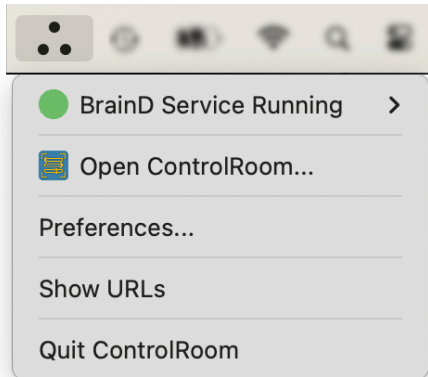
- In the ControlRoom popup menu, select “Open ControlRoom” to launch the viewer in a browser:



ControlRoom popup menu (Windows)

Mac

- Launch the ControlRoom app from the applications folder
- Acroname's 3-dot icon (⋮) will appear in the Mac menu bar
- Click to pop up the ControlRoom menu
- In the ControlRoom popup menu, select “Open ControlRoom” to launch the viewer in a browser:



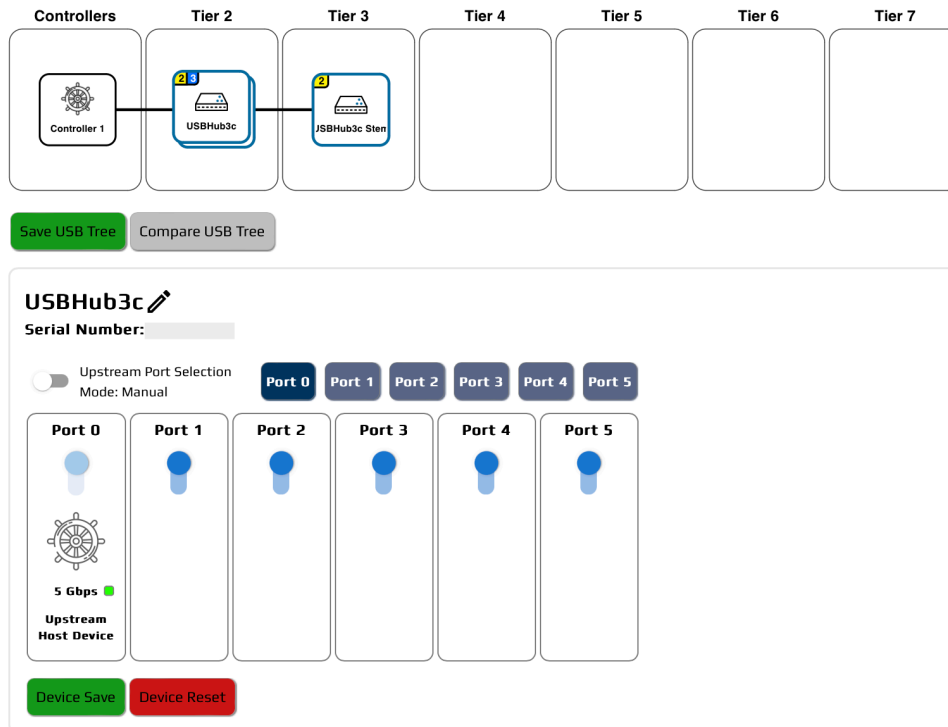
ControlRoom popup menu (Mac)

ControlRoom viewer web app



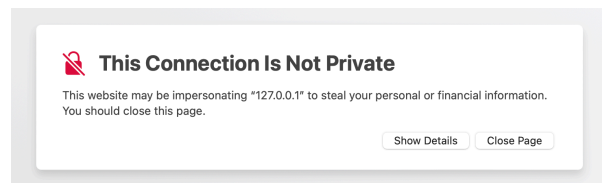
ControlRoom

USB Tree

*ControlRoom view with one USBHub3c with no attached devices*



The ControlRoom viewer web app shows the USB tree and port control panels for each connected Acroname device.

Note: If ControlRoom is configured to use HTTP Secure, a warning may pop up if using a self-signed certificate. Select show details and “visit this website” to continue.

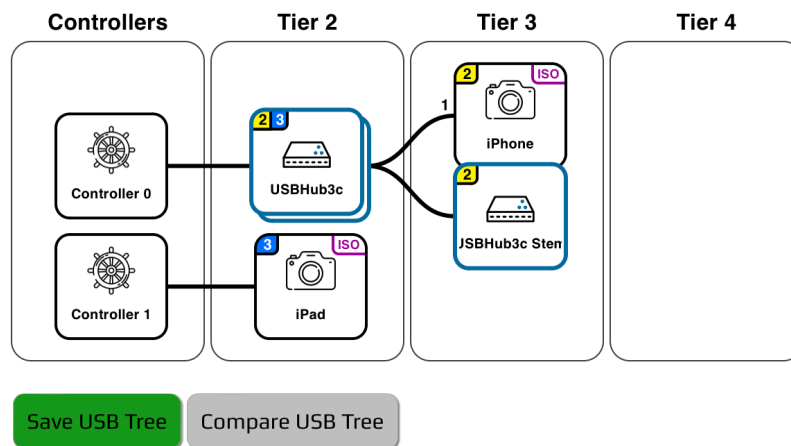


USB tree

The top of the page contains the USB Tree section. This section shows all of the USB devices connected to the system, as well as the hierarchy of the devices. The tree will continuously update as devices are connected and disconnected. See [the USB Tree section](#) for more information.

In this example, USBHub3c is the hub, USBHub3c Stem is a microcontroller responsible for controlling the hub and communicating with ControlRoom. If a box shows “2” and “3” (), then the device is connected to the host via both USB 2 and USB 3. If just a “2” is showing (), the device is only connected via USB 2.

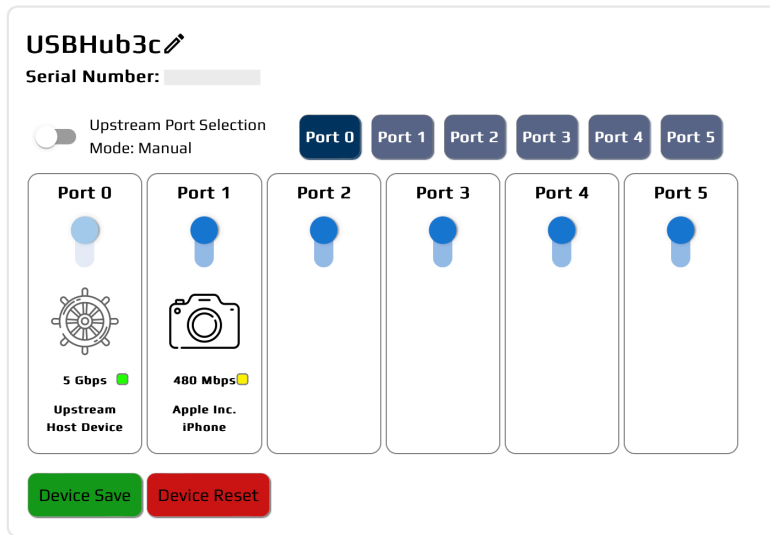
USB Tree



USB tree with an iPhone connected

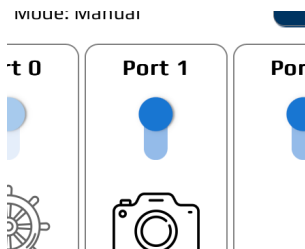
To demonstrate, connect a USB Device into USBHub3c Port 1. It will appear in the tree as a child of the USBHub3c. For this example an iPhone is connected, and the device icon shows that it is connected by USB 2 and the “ISO” mark indicates that it is capable of isochronous transfers such as streaming video and audio.

Port control panel



Port control panel for USBHub3c with an iPhone connected

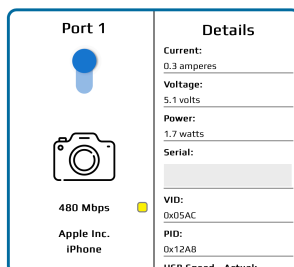
Below the USB Tree are panels representing each connected Acroname device. This section allows a user to power ports on and off, view attached devices, and more. See [the Port Control section](#) for more information.



Port Toggle

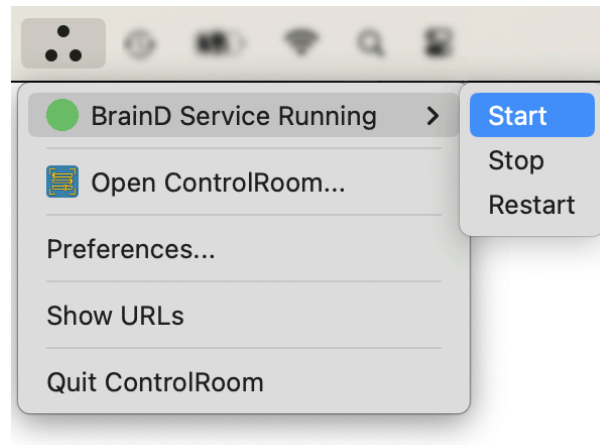
To demonstrate, disable the device on USBHub3c Port 1 by toggling the blue enable switch. The device will be removed from the USB Tree.

Detailed information about each connected device can be seen by clicking on the device icon below each port power switch:



Taskbar menu

The ControlRoom background application enables simple configuration and manipulation of the application service through a taskbar menu.



ControlRoom taskbar menu

BrainD Service [Running or Stopped] – shows whether the BrainD service that communicates with Acroname devices is running:

- Start – starts the BrainD service. This is required for ControlRoom to operate
- Stop – stops the BrainD service
- Restart – restarts the BrainD service. All active connections will terminate

Open ControlRoom – launches the ControlRoom web app in a browser

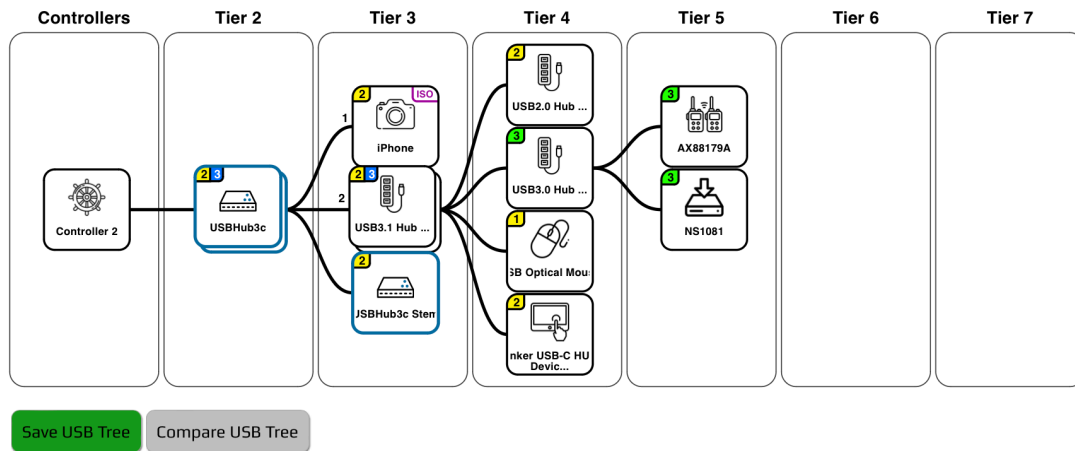
Preferences – opens the [settings and configuration dialog](#)

Show URLs – displays all broadcasted URLs that can be used to access the local ControlRoom session

Quit ControlRoom – closes and quits the ControlRoom taskbar and stops the BrainD service

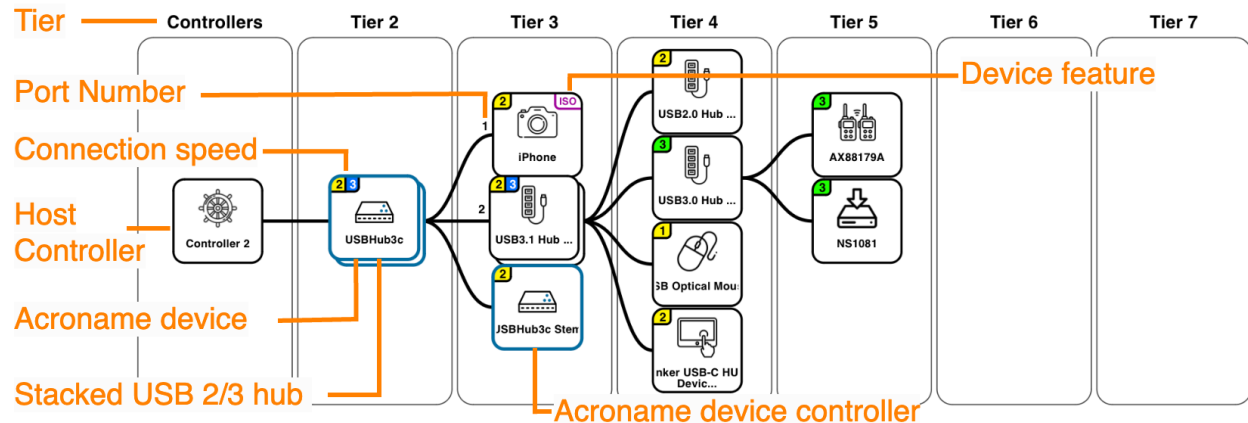
USB Tree

USB Tree



The USB Tree is a great way to visualize the USB device tree for the host processor. USB has the concept of tiers: the host and root hub are on the first tier. A device connected to the root hub is on the second tier. If a hub is added in between the device and the root hub, the device moves down to the third tier. Up to 5 non-root hubs can be added in series for a maximum of 7 USB tiers (5 Hubs + root hub + device).

The USB Tree provides a simple visualization of the 7 hub tiers along with each device and how it is connected back to the host controller.

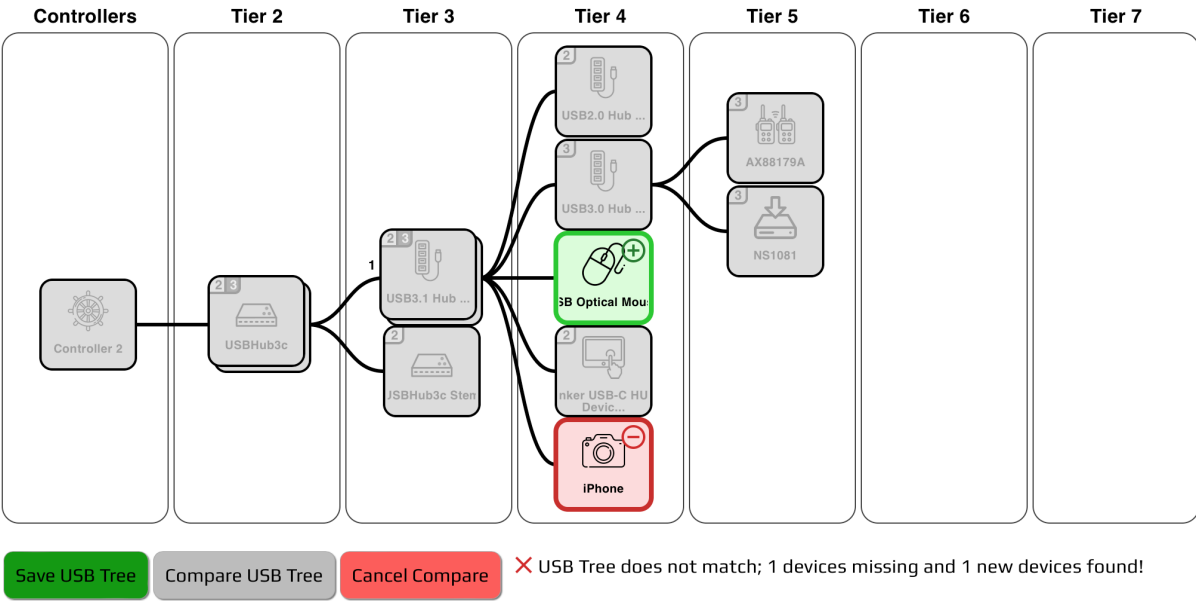


Label	Description
Tier	Depth of USB tree, up to 7
Port number	Physical port of the Acroname product the USB device is connected to (Only available when connected to Acroname products)
Connection speed	3 = USB 3 (blue = 10 Gb/s, green = 5 Gb/s) 2 = USB 2 480 Mb/s 1 = USB 1
Host controller	Host controller node, tier 1
Acroname device	Acroname devices are outlined in blue to for easy indentification
Stacked USB 2/3 hub	Hubs with both USB 2 and USB 3 connections are shown as stacks
Acroname device controller (stem)	Responsible for controlling the hub and communicating with ControlRoom
Device feature	USB device feature. ISO indicates that the device is capable of isochronous transfers such as streaming video and audio

USB Tree Comparisons

The USB Tree can be saved as a PDF with embedded data. The current tree can then be compared with a saved tree to easily identify changes.

USB Tree



- Save USB Tree** – saves to a PDF file. Saved USB Tree PDF files can be used as recipes for room designs
- Compare the USB Tree** – uploads a saved USB Tree PDF file for dynamic comparison with the current tree. USB devices that match the loaded USB Tree recipe will show a gray background. Missing devices will be shown in red and new devices in green. If a device has been moved to a different port, it will appear twice, once in green as new, and once in red as missing
- Cancel Compare** – cancels the USB Tree comparison
- Summary** – if the current tree does not match, a summary will be displayed indicating how many devices are new, and how many devices are missing

Port Control

The Port control panels represent each connected Acroname device, providing detailed control and information per port.

Editable Acroname device name – click the pencil icon to change the friendly name, click *device save* to store the change to the device internal memory. Default – product family name

Serial number – unique to the Acroname device

Upstream port selection – toggle to enable manual or auto (lowest port number priority) host port selection. In manual mode, the upstream port can be selected directly by clicking the dark blue port name buttons

Port name – name of Acroname device port. Default – numerical value of port

Device class icon – unique for each device class

Port toggle – independently enables and disables each port. Up = on


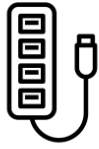

USB device speed – nominal device connection speed

USB MFG / device name – manufacturer and device name descriptors

Device save – stores changes to the hub friendly name and port enabled / disabled toggles. Saved changes will persist through power cycles

Device reset – reboots the Acroname device, temporarily disconnecting and reconnecting all connected USB devices

Port Details Expanded

Port 2	Details
	Voltage: 5.1 volts
	Power: 2.5 watts
10 Gbps 	Serial: 000000001
VIA Labs, Inc. USB3.1 Hub	VID: 0x2109
	PID: 0x0822
	USB Speed - Actual: 10 Gbps

Expanded port view with details

Click the USB device icon to expand each port view for realtime details:

- Current
- Voltage
- Power
- Serial number
- Vendor and Product ID
- Actual and maximum USB speeds
- Device manufacturer and device name
- Cable information (if advertised):
 - Cable current max

- Cable voltage max
 - Cable speed max
 - Cable orientation
 - Cable type
-

2.3.3 Advanced

For more advanced ControlRoom options, see configuration, logging, and security.

Configuration

Preferences

To change the default settings of ControlRoom, the desktop application provides a simple dialog window that enables users to change the HTTP server settings and specify *SSL/TLS Certificates*. To open the dialog, click on the taskbar menu and select “Preferences”.

Basic settings

Basic ControlRoom settings

Config File(s) – optional path to load JSON *configuration files*

Start BrainD at login – toggles BrainD service to start automatically

Allow remote access – lets other computers view the local USB tree information over a network connection

Use HTTP Secure – *Enables SSL/TLS* on client connections.

Restore Defaults – sets configuration to factory defaults

Cancel – closes the preferences window without saving

Save – saves application settings, restart ControlRoom to apply

Advanced settings

Advanced ControlRoom settings

Server Address – IP address to allow remote browsers to connect and view the local ControlRoom instance. Default is 127.0.0.1

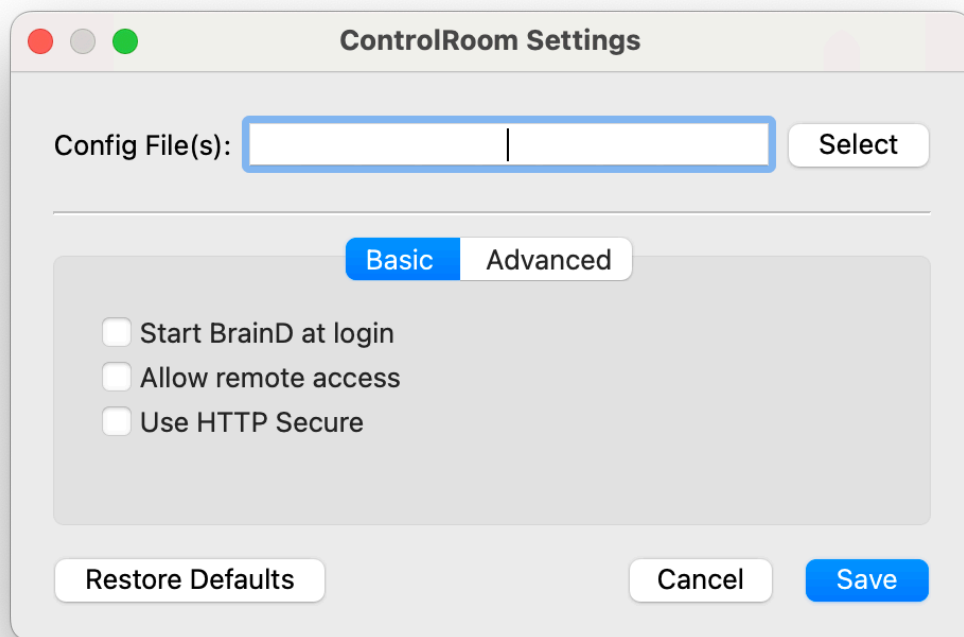
Server Port – server port for remote connections. Default is 9005

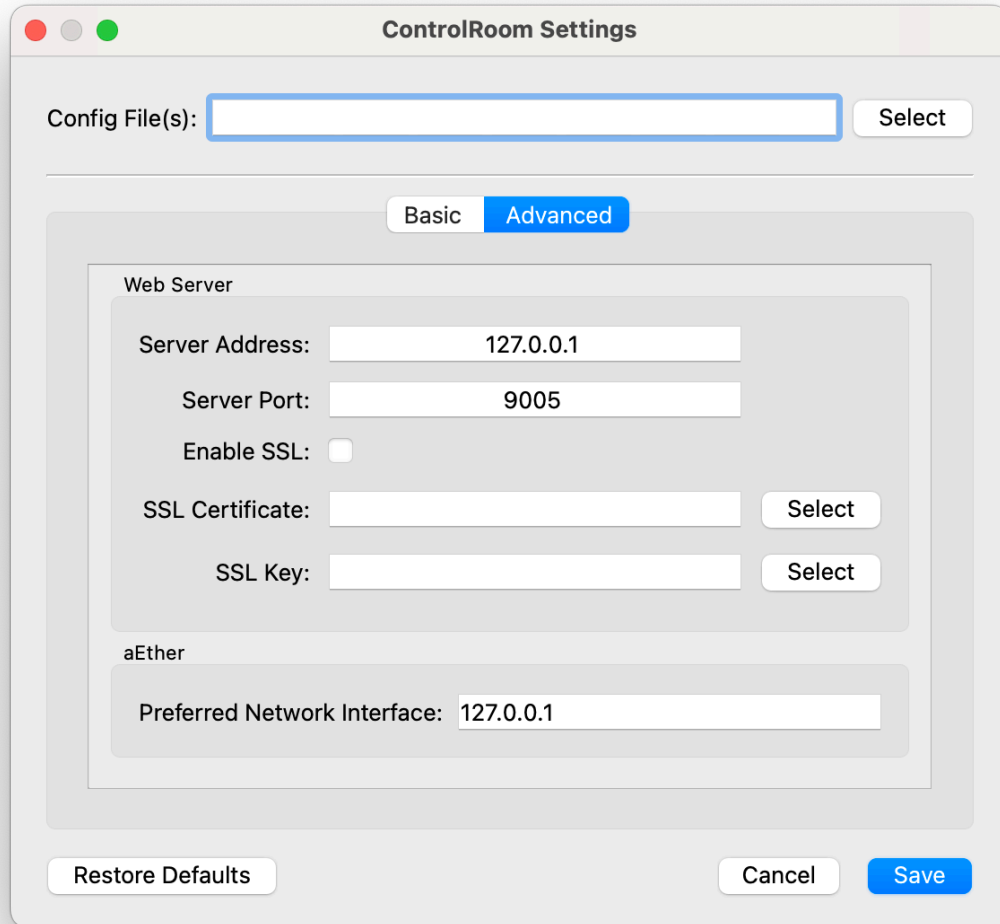
Enable SSL – *enables SSL/TLS* on client connections.

SSL Certificate – *SSL/TLS Certificate (.cert) file*. If not specified, a self-signed cert will be generated.

SSL Key – *SSL/TLS Key (.key) file*. If not specified, a self-signed cert will be generated.

Preferred Network Interface – IP address to allow remote applications to connect over aEther. Default is 127.0.0.1





The image shows a macOS-style dialog box titled "ControlRoom Settings". At the top, there are three window control buttons (red, yellow, green). Below the title bar, there is a text field labeled "Config File(s):" followed by a "Select" button. A horizontal line separates this from the main settings area. Inside the main area, there are two tabs: "Basic" and "Advanced", with "Advanced" currently selected. The "Advanced" tab contains two sections: "Web Server" and "aEther". The "Web Server" section has five fields: "Server Address" (127.0.0.1), "Server Port" (9005), "Enable SSL" (unchecked checkbox), "SSL Certificate" (text field with a "Select" button), and "SSL Key" (text field with a "Select" button). The "aEther" section has one field: "Preferred Network Interface" (127.0.0.1). At the bottom of the dialog, there are three buttons: "Restore Defaults", "Cancel", and "Save".

ControlRoom Settings

Config File(s):

Basic Advanced

Web Server

Server Address:

Server Port:

Enable SSL: ☐

SSL Certificate:

SSL Key:

aEther

Preferred Network Interface:

Configuration Files

ControlRoom allows for more fine-grained control of the backend service through the use of JSON configuration files. This configuration includes settings for [logging](#), polling rates, and web server configuration.

The BrainD service will read the following configuration file for the current operating system:

1. **Windows:** %AppData%\Acroname\BrainD\config\config.json
2. **Mac:** ~/Library/Application Support/Acroname/BrainD/config/config.json
3. **Linux:** ~/.acroname/BrainD/config/config.json

Each entry in the config.json file will be merged into the default configuration object, overriding each field that is listed. In addition, if any configuration files are specified in the “Config File(s)” entry of the Settings menu, these files will be read and merged as well. The default configuration object is defined as the following:

```
{
  "logging": {
    "loggers": {
      "default": "debug"
    },
    "sinks": {
      "consoleSinks": [
        {
          "enable": true,
          "level": "debug",
          "enableColors": true,
          "colorPattern": "\u001B[0m[\u001B[32m%Y-%m-%d %H:%M:%S.%e\
\u001B[0m] [%^%l%$] [\u001B[35m%n\u001B[0m:\u001B[33m%#\u001B[0m] %v",
          "noColorPattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v"
        }
      ],
      "fileSinks": [
        {
          "enable": true,
          "level": "info",
          "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
          "path": {
            "win": "%AppData%\Acroname\BrainD\log",
            "mac": "~/Library/Logs/Acroname/BrainD",
            "lin": ~/.acroname/BrainD/log"
          },
          "baseFileName": "braind.log",
          "maxFiles": 5,
          "maxFileSize": 5242880
        }
      ]
    },
    "flushInterval": 3000
  },
  "httpServer": {
    "enable": false,
    "address": "0.0.0.0",
    "port": 9005,
    "threadCount": 1
  },
  "httpsServer": {
    "enable": true,
```

(continues on next page)

(continued from previous page)

```
    "address": "0.0.0.0",
    "port": 9006,
    "threadCount": 1,
    "sslCert": "",
    "sslKey": ""
  },
  "pollingRates": {
    "acronameDevicesState": 300,
    "devices": 300,
    "brainstem": 1000
  }
}
```

Note: The `config['httpServer']` and `config['httpsServer']` configuration objects will have no effect if the ControlRoom desktop application is used. The value in the Settings dialog takes priority.

Examples

Add an additional log file output:

```
{
  "logging": {
    "sinks": {
      "fileSinks": [
        {
          "enable": true,
          "level": "info",
          "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
          "path": {
            "win": "C:\\\\Users\\username\\Desktop",
            "mac": "~/Desktop",
            "lin": "~/Desktop"
          },
          "baseFileName": "custom-log-file.log"
        }
      ]
    }
  }
}
```

Increase the log flush rate from 3000ms to 500ms, and reduce the USB tree polling rate from 300ms to 1000ms:

```
{
  "logging": {
    "flushInterval": 500
  },
  "pollingRates": {
    "devices": 1000
  }
}
```

Logging

ControlRoom generates runtime logging to aid in debugging and understanding for many events. These log files are stored by default in the following path for each operating system:

1. Windows: %AppData%\Acroname\BrainD\log
2. Mac: ~/Library/Logs/Acroname/BrainD
3. Linux: ~/.acroname/BrainD/log

By default, these log files will rotate when they reach 5MB in size, up to a maximum of five log files. After this point, old log files will be removed to make room for new ones.

For information on how to adjust the logging parameters or specify new log files, see [Configuration](#).

Security

SSL Certificates

SSL (Secure Sockets Layer) is a fundamental technology for securing web applications by encrypting data, ensuring data integrity, authenticating servers, and enhancing user trust.

SSL layers encrypt the data transmitted between a user's web browser and server. This encryption ensures that the data remains unreadable and secure even if intercepted by malicious actors. SSL includes mechanisms to ensure data integrity during transmission using cryptographic hash functions to detect unauthorized modifications or tampering. In the event of altered data during transit, the recipient can see and reject the modified data.

ControlRoom allows loading a user's SSL certificate through the [settings and configuration dialog](#). Many external service providers can provide an SSL certificate; one may already have an SSL certificate for ControlRoom installations.

Use of a self-signed certificate may cause a warning to appear when accessing the ControlRoom webpage or BrainD REST endpoints. This is caused by the browser failing to authenticate the received certificate against a trusted certificate authority.



This Connection Is Not Private

This website may be impersonating "127.0.0.1" to steal your personal or financial information. You should go back to the previous page.

Go Back

Safari warns you when a website has a certificate that is not valid. This may happen if the website is misconfigured or an attacker has compromised your connection.

To learn more, you can [view the certificate](#). If you understand the risks involved, you can [visit this website](#).

OpenSSL is an option that one can use to create a self-signed SSL certificate. The following command may be used to generate a self-signed certificate:

Bash

```
openssl req -new -x509 -days 365 -nodes -keyout example.key -out example.crt
```

By default, ControlRoom will serve an insecure web page over HTTP, without any certificates or encryption. If insecure communications over HTTP are not desirable, the [settings and configuration dialog](#) can be used to enable HTTPS/SSL communication. If a certificate is not specified, then ControlRoom will automatically generate a self-signed certificate.

Certificates are stored in the following directory:

1. Windows: %AppData%\Acroname\BrainD\config
2. Mac: ~/Library/Application\ Support/Acroname/BrainD/config
3. Linux: ~/.acroname/BrainD/config

Platform Specific Considerations

Configure MacOS to Advertise Its Hostname

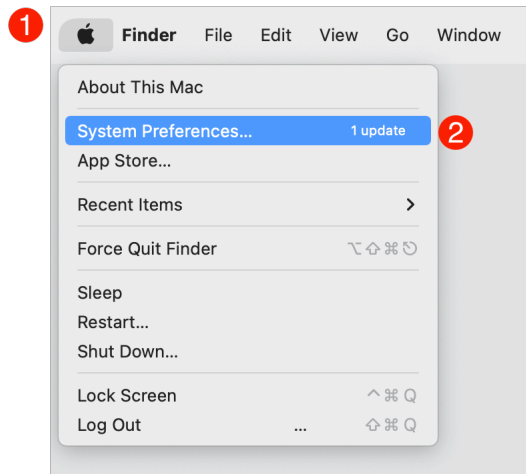
Name resolution is a process where a readable name, e.g. ElizaPC, can be resolved into a numerical IP address, e.g. 192.168.41.75. Name resolution eases server connections, by allowing usage of URLs like `https://ElizaPC` instead of a hard-to-remember, and fragile, `https://192.168.41.75`.

MacOS, in default configuration, doesn't provide resolution for the host's own name.

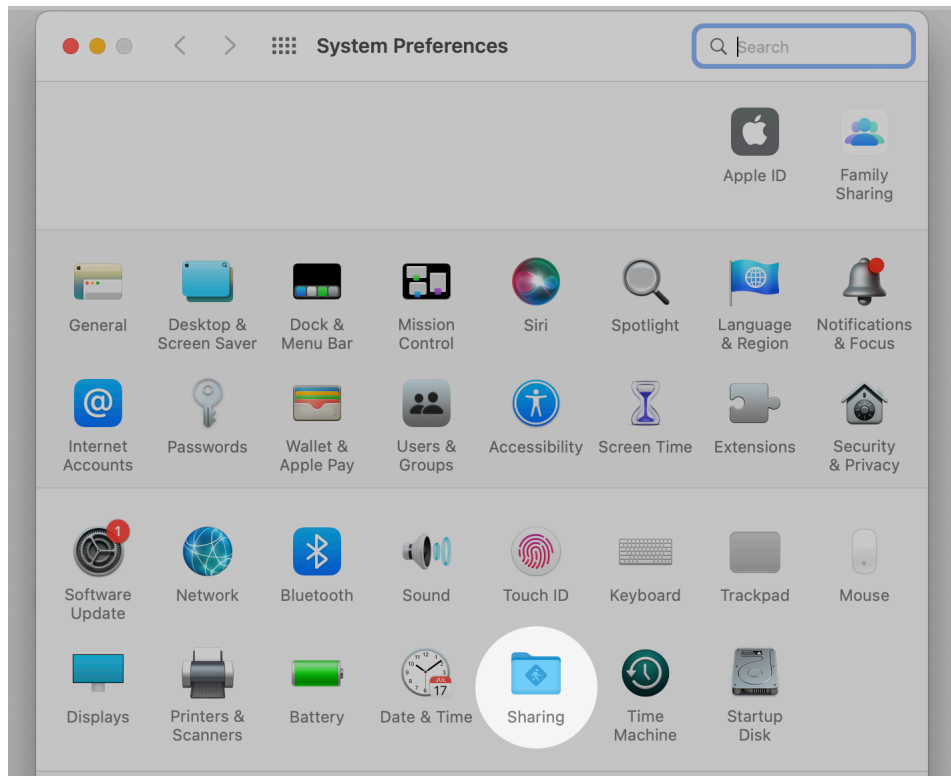
Here's how to fix that.

1. Get to System Preferences via the Apple Menu in the top left of the screen.

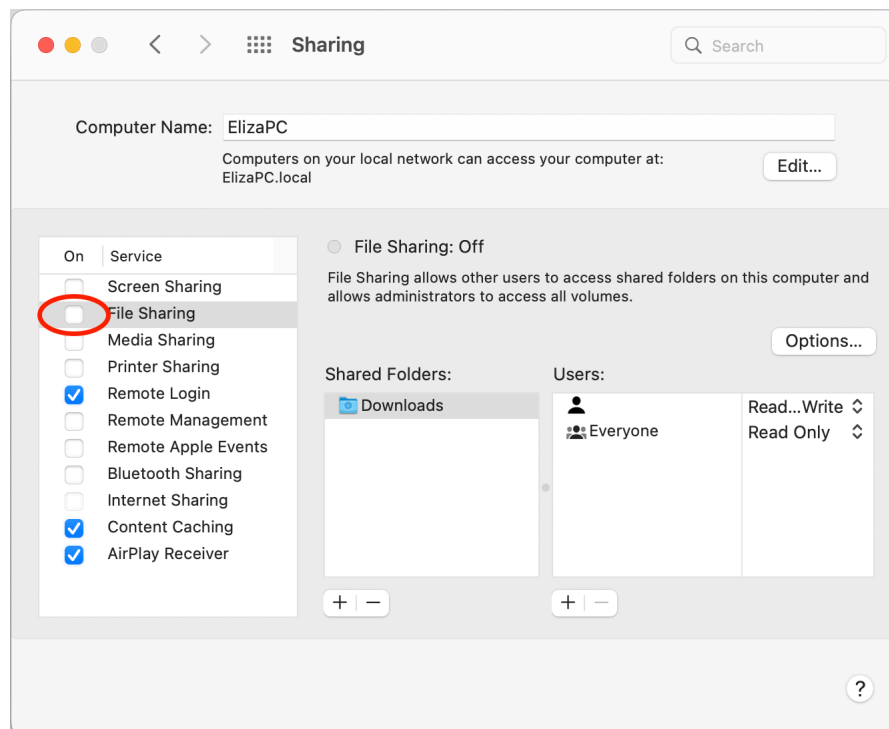
Top Left of the Screen



2. Go to the "Sharing" Panel



3. Enable File Sharing by checking this box. Other devices on the network can now resolve this machine's name. Note: File sharing does not have to remain enabled but must be enabled at least once for name resolution.



4. Configure these settings to your application's needs. This configuration is just for pictorial example. Other devices on the network can still resolve this machine's name, if File Sharing was ever enabled.

2.3.4 Features and Functions

- Visualize the USB tree in a browser in real-time
- Create device layout recipes via snapshots
- Compare the current device layout to a loaded snapshot
- View device status including:
 - Device connection speed
 - Custom devices names
 - Troubleshooting information
- Control Acroname device functions including:
 - Downstream port toggle, simulating a device unplug / plug cycle
 - Upstream port swap
 - Upstream port selection mode
 - Save settings to non-volatile storage
 - Remote Acroname device reset

2.4 Q-Sys



The Q-Sys OS and Designer Software serves as the software-based singular foundation that drives and manages a multitude of Q-Sys Products within the platform, including native software, services and hardware. Acroname's products bring unique control and telemetry of USB devices and peripherals.

The Q-Sys Designer Software's modern IT architecture and a set of development tools (called "Q-Sys Open") enable an entire Ecosystem of third-party integrations developed by approved/endorsed Q-Sys Partners as well as a worldwide community of Q-Sys programmers and developers. Acroname is part of the approved Partners.

2.4.1 Installation

Download the appropriate [Q-Sys Design Software](#)⁹⁹ for your operating system. See Q-Sys support documentation for installation instructions.

Requirements

- Q-Sys Designer Software Version 9.9 and higher is recommended when using a virtual core instances.

Q-Sys requires a supported Acroname USBHub3c with the [Serial Control add-on software feature](#)¹⁰⁰:

- [USBHub3c](#)¹⁰¹ – USB-C Hub with Power Delivery Analyzer + Tester

Install Plugin

The Q-Sys Asset Manger is the recommended method for managing plugins. To install an Acroname plugin:

1. Click the plugin icon to open the Asset Manager
2. In the Browse tab, search for `Acroname` and select the plugin that correlating the Acroname hardware.
3. Review the Description information for any additional notes or requirements.
4. Click `Install`. When the installation is complete, the Acroname product plugin will appear in the Plugins categoray within the Schematic Elements section of Q-Sys Designer.

2.4.2 Quick Start Guide

To get started with Q-Sys, follow these steps from the Q-Sys Designer Software:

1. Install Q-Sys Plugin following the [Installation Guide](#).
2. Connect an Acroname product with a valid Acroname USBHub3c with the *Serial Control add-on software feature* <<https://acroname.com/store/t99-serial>> installed. For this guide, a [USBHub3c](#).
3. Add an Acroname USBHub3c plugin to the design canvas to create rich USB based applications.

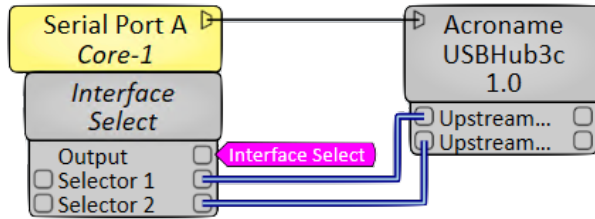
⁹⁹ <https://www.qsys.com/resources/software-and-firmware/q-sys-designer-software/>

¹⁰⁰ <https://acroname.com/store/t99-serial>

¹⁰¹ <https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro>

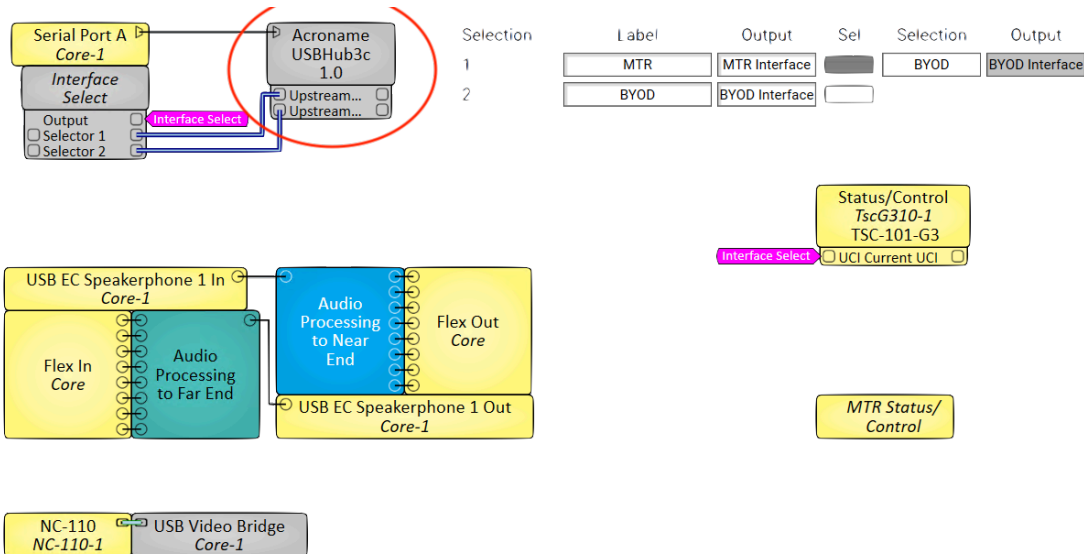
Connecting through Serial Port

The following example shows highlights that a serial port on the Core can be connected to a USBHub3c serial port.



Change Upstream Port

The following example shows how an output selector through the Q-Sys user interface can change the upstream port on the USBHub3c. Switching the upstream port allows USB devices to communicate with two different host machines, which is often found in BYOD applications and environments.

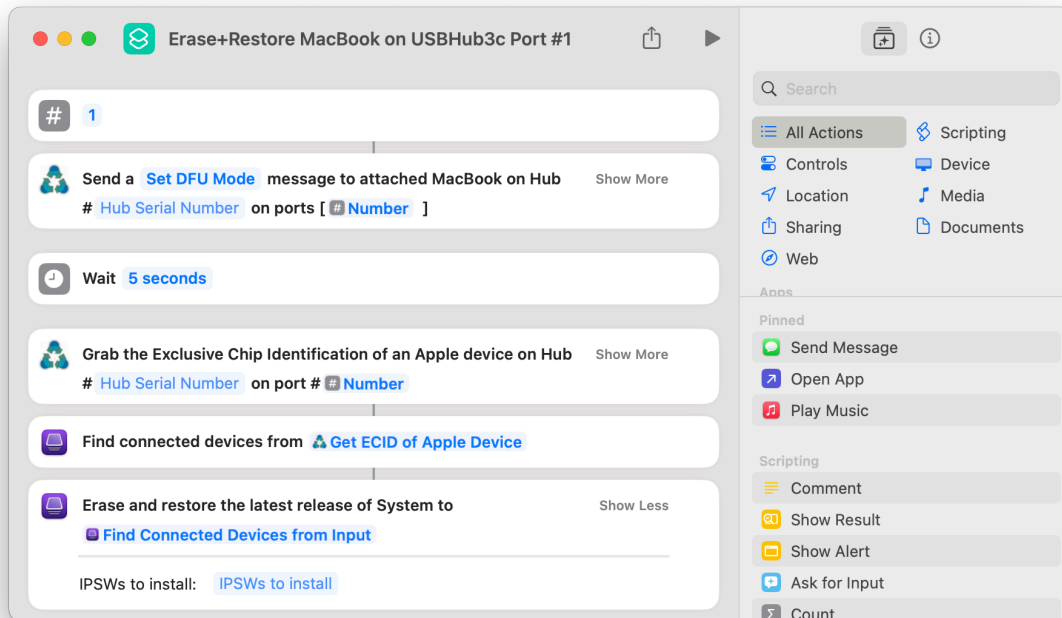


2.4.3 Usage

2.4.4 Features and Functions

- Control Acroname device functions including:
 - Downstream port toggle, simulating a device unplug / plug cycle
 - Downstream port connection status
 - Upstream port selection
 - Upstream port connection status
 - Firmware and model information

2.5 DFU Automator



Apple Devices can be put into Device Firmware Update (DFU) mode to allow for firmware updating and restoration, as well as the deployment of custom OS images. This is achieved by sending Vendor-Specific Messages (VDMs) to a DFU-capable port on the target device. VDMs are point-to-point and are not passed on by a USB Hub; however, USBHub3c can generate and send arbitrary VDMs directly to connected devices. This lets Mac Sysadmins re-image multiple Macs at once, greatly speeding up deployment.

DFU Automator is a small application that enables Apple Shortcuts to control USBHub3c's DFU functions. DFU Automator operates as a background tool, responding to commands issued by Apple Shortcuts. When combined with Apple Configurator actions, shortcuts can fully automate Mac deployment and provisioning.

Included with DFU Automator is an example shortcut showing how to DFU restore a Macbook on port 1 of USBHub3c.

2.5.1 DFU Automator Features

- Send DFU Mode or Reboot VDMs to a MacBook
- Send arbitrary VDMs (Hexstring input)
- Enable or disable hub ports
- Retrieve the ECID of an attached Apple device

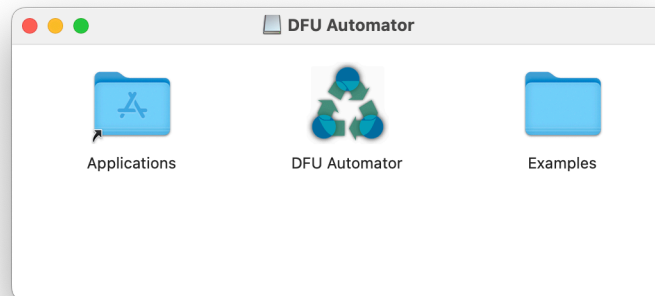
Installation and Setup

Install Apple Configurator

- Open the **App Store** on your Mac
- Search for **Apple Configurator**
- Click **Install**

Install DFU Automator

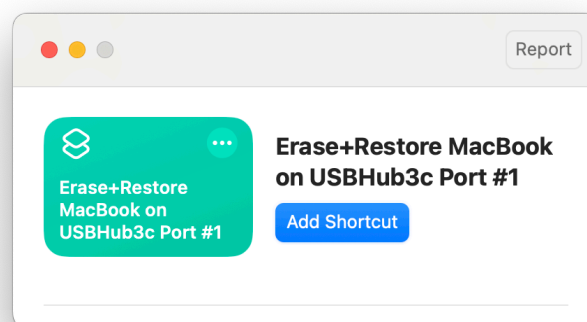
1. *Download DFU Automator* from the official source
2. Open the downloaded *.dmg* file



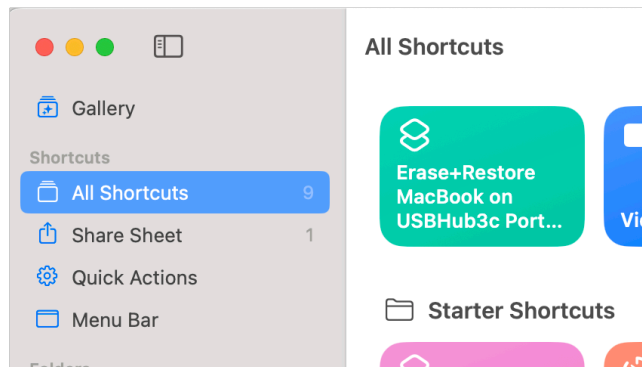
3. Drag **DFU Automator** into the **Applications** folder
4. Open DFU Automator from the **Applications** folder once to allow Shortcuts to recognize it

Adding the Example Shortcut

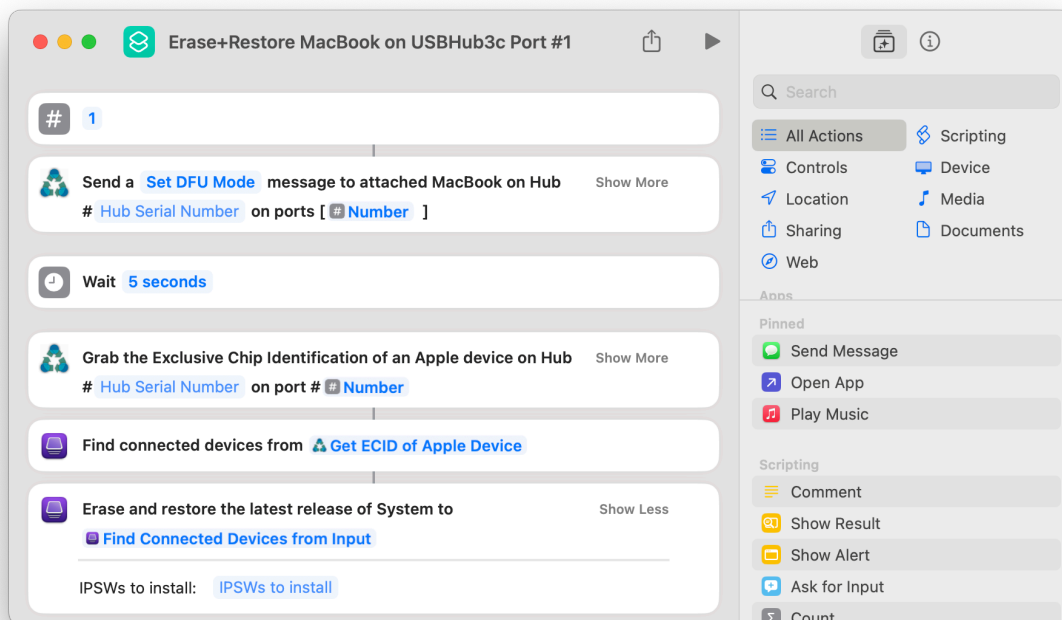
1. In the **Examples Folder**, open the shortcut titled **Erase+Restore MacBook on USBHub3c Port #1**



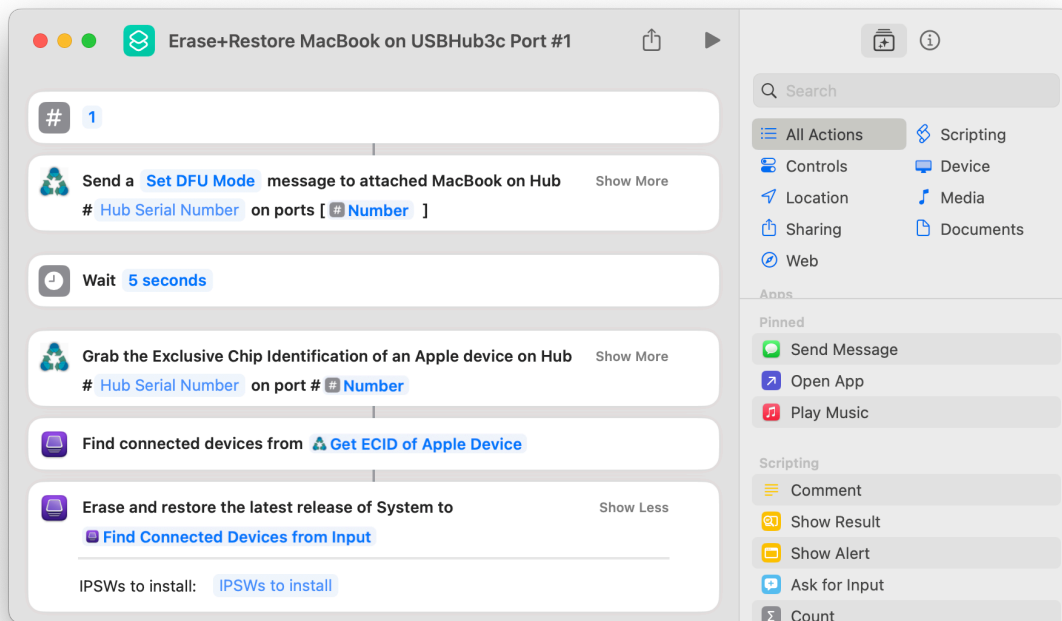
2. Click **Add Shortcut**
3. The shortcut will now appear in the **Shortcuts app** under **All Shortcuts**



4. Click the shortcut's icon (not the play button that appears on hover) to view the workflow:



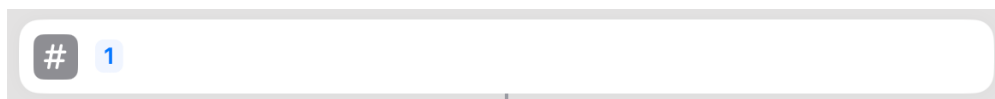
Shortcut Example



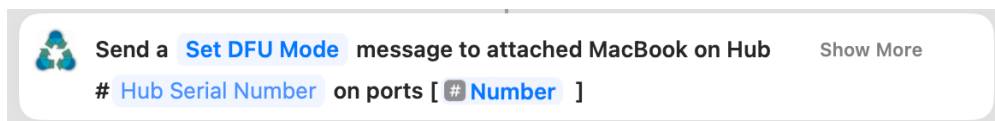
Shortcut Workflow Step-by-Step

To understand each step of the shortcut, **Control-click** an action and select **Show Info**.

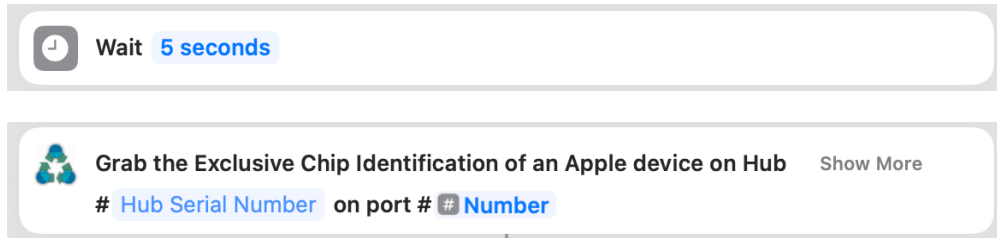
Key Actions



- Sets the USBHub3c port number for the MacBook to be updated (set to 1 by default)



- Sends a *Set DFU Mode* message on hub # *Hub Serial Number* on ports [*# Number*]
 - Tells DFU Automator to have USBHub3c send a Vendor-Defined-Message to put a MacBook on port [*# Number*] into DFU mode
 - If only one hub connected, *Hub Serial Number* can be blank
- Waits 5 seconds to give the MacBook time to enter DFU Mode



- Tells DFU Automator to read the ECID of the attached MacBook on the specified port



- Tells Apple Configurator to convert the ECID to a *Connected Device Entity* for use in other Apple Configurator shortcut actions
- Erases and restores the connected devices

Warning: All data and settings will be removed!

- *IPSWs to install:* Select additional IPSW files to install on connected devices. Configurator will choose the correct IPSW for the device being updated

Example Usage

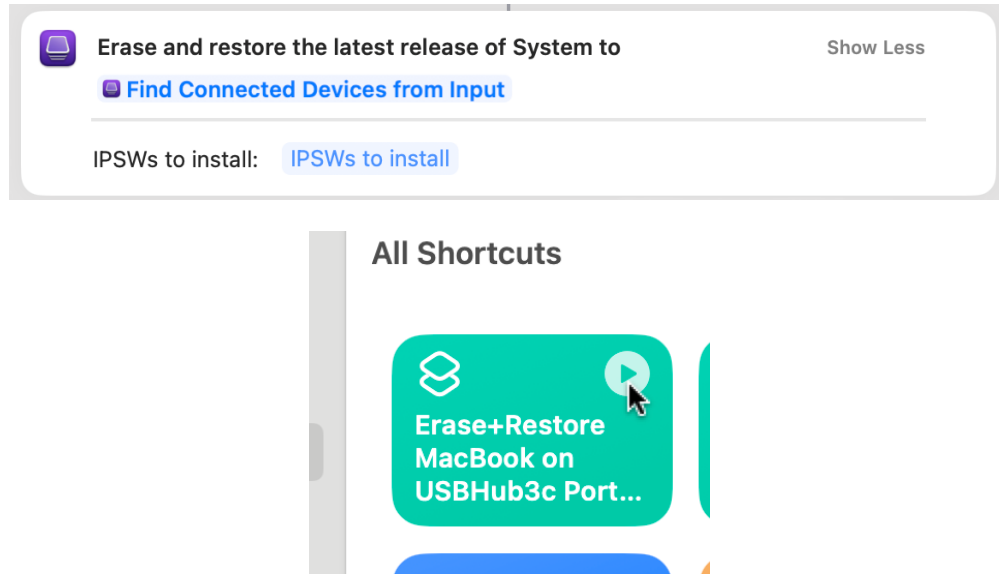
To use the example shortcut to DFU-restore a MacBook:

1. Locate the DFU-enabled USB-C port on the target Mac:
Apple Silicon MacBooks: Left side, towards the hinge
Intel MacBooks with T2 chip: Left side, away from the hinge
Desktops: Refer to [Apple Support](https://support.apple.com/en-us/108900)¹⁰² for more details
2. Connect the host Mac to **USBHub3c Port 0**
3. Connect the target MacBook's DFU-enabled port to **USBHub3c Port 1**

Warning: Target MacBook will be erased!

4. In **Shortcuts**, click the **play button** in the upper right of the shortcut icon or the expanded view of the shortcut
5. Click "Allow" to let the shortcut run actions
 - The target MacBook will chime and boot into DFU mode with a black screen
6. Approve any other privacy popups that appear:
 - Allow configurator to find devices
 - Allow the shortcut share the device ECID with Apple Configurator

¹⁰² <https://support.apple.com/en-us/108900>



8. Apple Configurator will erase and restore the MacBook on port 1

Restoring Multiple Devices Simultaneously

Shortcuts and Configurator actions can be run in parallel, allowing multiple MacBooks to be restored asynchronously. The restore shortcut can be copied and the port number can be set for each port. To use, connect a MacBook to an available USBHub3c port and manually trigger the corresponding restore shortcut matching the port number.

More information about using Apple Configurator with Shortcuts can be found [here](https://support.apple.com/en-au/guide/apple-configurator-mac/acm12a6f7b56/mac)¹⁰³.



lets users view detailed information and control settings of Acroname devices.



vice that exposes an HTTP REST interface to connected BrainStem Devices.



HubTool
is
a
util-
ity
that

BrainD
is
a
desk-
top
ser-

*Con-
trol-
Room*

¹⁰³ <https://support.apple.com/en-au/guide/apple-configurator-mac/acm12a6f7b56/mac>

browser application built on BrainD, designed for audio/video applications and USB device diagnostics.



for BrainStem Devices to be used in the Q-Sys Designer Software for audio/video applications.



helper application that enables Apple Shortcuts to control USBHub3c's Device Firmware Update (DFU) mode operations for Apple devices.

is
a
web

Q-
Sys
plug-
in
in-
ter-
face

DFU
Au-
toma-
tor
is
a

API Reference

This API reference is organized by programming language. You will find product specific documentation in the [products](#) section.

3.1 BrainStem Entities

3.1.1 Analog Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read an analog voltage (ADC) and convert this into a discrete digitized value or output a voltage value based on a desired discrete value (DAC). Analog voltage capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of analog entities and details for their capacities will be described in that module's datasheet.

Value (Get/Set)

```
analog [ index ] . getValue <= (unsigned short) value
analog [ index ] . setValue => (unsigned short) value
```

Getting Values

A BrainStem's A2D reading will always return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the measurement will get propagated up to 16 bits wide.

For example, if a 12-bit A2D engine exists in the target module's hardware, the reading will get promoted in the firmware layer by shifting up 4 bits to fill out the 16 bit value ($0x0FFF =: 0x0FFF << 4 = 0xFFF0$) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

Setting Values

The reading resolution will return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the value sent by the API will get propagated up to 16 bits wide.

For example, if a 10-bit DAC engine exists in the target module's hardware, the reading will get down shifted 5 bits to derive the 10 bit value ($0x8000 =: 0x8000 >> 5 = 0x0400$) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

Configuration (Get/Set)

```
analog [ index ] . getConfiguration <= (unsigned char) configuration
analog [ index ] . setConfiguration => (unsigned char) configuration
```

Getting Configuration

Some analog entities may be single purpose functionality or can be configured for multiple different behaviors depending on the hardware. Configuration information includes whether the entities is an input only, output only, or can be configured as either and input or output.

Setting Configuration

Analog entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most analog entities are typically as inputs, but will vary by module hardware.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

stem.analog[0].getValue(value); // gets the value of A2D channel 0 into variable value
stem.analog[3].setValue(1234); // sets the DAC on channel 3 to a value of 1234
stem.analog[0].setConfiguration(analogConfigurationInput);
```

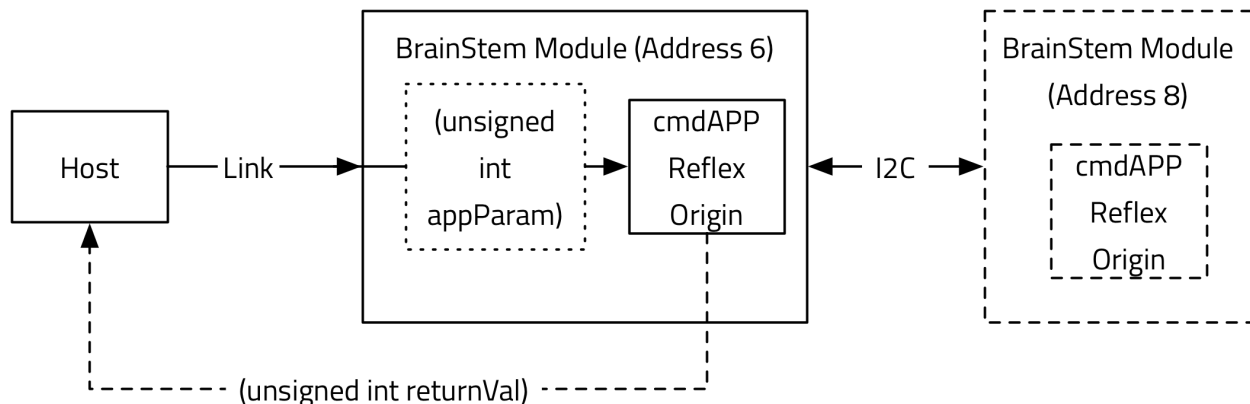
Python

```
result = stem.analog[2].getValue() # gets the value of A2D channel 2 into variable_
↪result.
print result.value
err = stem.analog[3].setValue(1234) # sets the DAC on channel 3 to a value of 1234
print err
```

3.1.2 App Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules have a unique mechanism and communication method to send host-to-stem or stem-to-stem messages that can initiate a Reflex origin to trigger if one is defined on the target module. BrainStem modules may have up to 4 different (0-3) entity app instances.



Please be aware that a Reflex file must be enabled on the target module for a call to an App entity to be successful.

Execute (non-blocking)

```
app[0] . execute => (unsigned int) appParam
```

This entities will pass the data specified in appParam to be passed into the Reflex handle. The 4 bytes are up to the implementor to mean what ever one wants them to be.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success.  
  
stem.app[0].execute(3131948783); // triggers the App reflex handle and passes 4 bytes.  
↳to it
```

Python

```
Implementation coming in future release.
```

3.1.3 Clock Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have a real time clock. This capability will be listed in the product datasheet. The clock entity allows the user to set and read the real time clock.

Year (Get/Set)

```
clock . getYear <= (unsigned short) year  
clock . setyear => (unsigned short) year
```

Gets or sets the year value of the real time clock.

Month (Get/Set)

```
clock . getMonth <= (unsigned char) month  
clock . setMonth => (unsigned char) month
```

Gets or sets the month value of the real time clock. Valid values are 1-12.

Day (Get/Set)

```
clock . getDay <= (unsigned char) day
clock . setDay => (unsigned char) day
```

Gets or sets the day value of the real time clock. Valid values are 1-31 depending on the month setting.

Hour (Get/Set)

```
clock . getHour <= (unsigned char) hour
clock . setHour => (unsigned char) hour
```

Gets or sets the hour value of the real time clock. Valid values are 0-23.

Minute (Get/Set)

```
clock . getMinute <= (unsigned char) minute
clock . setMinute => (unsigned char) minute
```

Gets or sets the minute value of the real time clock. Valid values are 0-59.

Second (Get/Set)

```
clock . getSecond <= (unsigned char) second
clock . setSecond => (unsigned char) second
```

Gets or sets the second value of the real time clock. Valid values are 0-59.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get requests fill the variable with the current clock value.

stem.clock.getYear(year);
stem.clock.setYear(year);
stem.clock.getMonth(month);
stem.clock.setMonth(month);
stem.clock.getDay(day);
stem.clock.setDay(day);
stem.clock.getHour(hour);
stem.clock.setHour(hour);
stem.clock.getMinute(minute);
stem.clock.setMinute(minute);
stem.clock.getSecond(second);
stem.clock.setSecond(second);
```

Python

```
year = stem.clock.getYear();
stem.clock.setYear(year);
month = stem.clock.getMonth();
stem.clock.setMonth(month);
day = stem.clock.getDay();
stem.clock.setDay(day);
hour = stem.clock.getHour();
stem.clock.setHour(hour);
minute = stem.clock.getMinute();
stem.clock.setMinute(minute);
second = stem.clock.getSecond();
stem.clock.setSecond(second);
```

3.1.4 Digital Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

State (Get/Set)

```
digital [ index ] . getState <= (unsigned char) state
digital [ index ] . setState => (unsigned char) state
```

Gets or Sets the digital I/O Value.

For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet.

For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

Configuration (Get/Set)

```
digital [ index ] . getConfiguration <= (unsigned char) configuration
digital [ index ] . setConfiguration => (unsigned char) configuration
```

Gets or Sets the digital pin configuration.

Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware.

Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

Available configurations for the digital entities:

Function	Typedef Constant (C++)	Typedef Constant (Python)	Val
Digital Input	digitalConfigurationInput	CONFIGURATION_INPUT	0
Digital Output	digitalConfigurationOutput	CONFIGURATION_OUTPUT	1
RCServo Input	digitalConfigurationRCServoInput	CONFIGURATION_RCSERVO_INPUT	2
RCServo Output	digitalConfigurationRCServoOutput	CONFIGURATION_RCSERVO_OUTPUT	3
High Z State	digitalConfigurationHiZ	CONFIGURATION_HIGHZ	4
Input Pull Up	digitalConfigurationInputPullUp	CONFIGURATION_INPUT_PULL_UP	0
Input No Pull	digitalConfigurationInputNoPull	CONFIGURATION_INPUT_NO_PULL	4
Input Pull Down	digitalConfigurationInputPullDown	CONFIGURATION_INPUT_PULL_DOWN	5
Signal Output	digitalConfigurationSignalOutput	CONFIGURATION_SIGNAL_OUTPUT	6
Signal Input	digitalConfigurationSignalInput	CONFIGURATION_SIGNAL_INPUT	7
Signal Counter Input	digitalConfigurationSignalCounterInput	CONFIGURATION_SIGNAL_COUNTER_INPUT	8
Link Input	digitalConfigurationLinkInput	CONFIGURATION_LINK_INPUT	9
Link Output	digitalConfigurationLinkOutput	CONFIGURATION_LINK_OUTPUT	10

Note: When using the High Z State configuration the pin and pull-ups are disconnected internally leaving the external pin floating. A get or set of the state will return in an error.

Link Channel (Get/Set)

```
digital [ index ] . getLinkChannel <= (unsigned char) channel
digital [ index ] . setLinkChannel => (unsigned char) channel
```

Configure one digital as **Link Input** and the other as **Link Output** (see the configuration table above), then set each side's link channel to the peer's entity index. Two digitals link when their `linkchannel` values match and the pair uses Link Input on one entity and Link Output on the other.

State All (Get/Set)

```
digital [ index ] . getStateAll <= (unsigned int) states
digital [ index ] . setStateAll => (unsigned int) states
```

Gets or sets the logical state of all digitals on the module in a single bit-mapped value. The number of digitals and valid bits depend on the module; see the product datasheet.

See the *RCServo Entity* for more information on its configuration.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

stem.digital[0].getState(&state); // gets the current digital state for channel 0
stem.digital[3].setState(1); // sets the digital output state to logic high on
↳channel 3
stem.digital[0].setConfiguration(digitalConfigurationInput);
```

Python

```
state = stem.digital[3].getState() # gets the value of digital channel 3 into
↳variable state
stem.digital[3].setState(1) # sets the digital on channel 3 to a logic high
```

3.1.5 Ethernet Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Ethernet Entity provides control over network configuration and monitoring for devices with Ethernet connectivity. This includes IP address management, network configuration modes, and interface settings.

Ethernet Enable/Disable (Get/Set)

```
ethernet . getEnabled <= (unsigned char) enabled
ethernet . setEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Ethernet interface. When enabled, the Ethernet interface is active and can be used for network communication.

Network Configuration (Get/Set)

```
ethernet . getNetworkConfiguration <= (unsigned char) configuration
ethernet . setNetworkConfiguration => (unsigned char) configuration
```

Returns or sets the network configuration mode for the Ethernet interface. This controls whether the interface uses DHCP or static IP configuration.

Table 1: Network Configuration Options

Value	Name	Description
0	None	No network configuration
1	Static	Static IP configuration
2	DHCP	DHCP automatic configuration

Static IPv4 Address (Get/Set)

```

ethernet . getStaticIPv4Address <= (unsigned char[]) address
ethernet . setStaticIPv4Address => (unsigned char[]) address

```

Returns or sets the static IPv4 address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the IP address (e.g., [192, 168, 1, 100] for “192.168.1.100”).

Static IPv4 Netmask (Get/Set)

```

ethernet . getStaticIPv4Netmask <= (unsigned char[]) netmask
ethernet . setStaticIPv4Netmask => (unsigned char[]) netmask

```

Returns or sets the static IPv4 netmask for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the netmask (e.g., [255, 255, 255, 0] for “255.255.255.0”).

Static IPv4 Gateway (Get/Set)

```

ethernet . getStaticIPv4Gateway <= (unsigned char[]) gateway
ethernet . setStaticIPv4Gateway => (unsigned char[]) gateway

```

Returns or sets the static IPv4 gateway address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the gateway address (e.g., [192, 168, 1, 1] for “192.168.1.1”).

IPv4 Address (Get)

```

ethernet . getIPv4Address <= (unsigned char[]) address

```

Returns the current IPv4 address of the Ethernet interface as a 4-byte array. This reflects the actual address being used, whether obtained via DHCP or set statically. Each byte represents one octet of the IP address.

IPv4 Netmask (Get)

```

ethernet . getIPv4Netmask <= (unsigned char[]) netmask

```

Returns the current IPv4 netmask of the Ethernet interface as a 4-byte array. This reflects the actual netmask being used, whether obtained via DHCP or set statically. Each byte represents one octet of the netmask.

IPv4 Gateway (Get)

```
ethernet . getIPv4Gateway <= (unsigned char[]) gateway
```

Returns the current IPv4 gateway address of the Ethernet interface as a 4-byte array. This reflects the actual gateway being used, whether obtained via DHCP or set statically. Each byte represents one octet of the gateway address.

Static IPv4 DNS Address (Get/Set)

```
ethernet . getStaticIPv4DNSAddress <= (unsigned char[]) dns  
ethernet . setStaticIPv4DNSAddress => (unsigned char[]) dns
```

Returns or sets the static IPv4 DNS server address for the Ethernet interface as a 4-byte array. This is used when the network configuration is set to static mode. Each byte represents one octet of the DNS address.

IPv4 DNS Address (Get)

```
ethernet . getIPv4DNSAddress <= (unsigned char[]) dns
```

Returns the current IPv4 DNS server address of the Ethernet interface as a 4-byte array. This reflects the actual DNS server being used, whether obtained via DHCP or set statically. Each byte represents one octet of the DNS address.

Hostname (Get/Set)

```
ethernet . getHostname <= (unsigned char[]) hostname  
ethernet . setHostname => (unsigned char[]) hostname
```

Returns or sets the hostname for the Ethernet interface. The hostname is used for network identification and can be up to 63 characters long.

MAC Address (Get)

```
ethernet . getMACAddress <= (unsigned char[]) mac
```

Returns the MAC address of the Ethernet interface. This is a unique hardware identifier for the network interface.

Interface Port (Get/Set)

```
ethernet . getInterfacePort (unsigned char) service <= (unsigned short) port  
ethernet . setInterfacePort => (unsigned char) service, (unsigned short) port
```

Returns or sets the interface port for specific services on the Ethernet interface. The service parameter specifies which service to configure, and the port parameter sets the port number for that service.

Table 2: Interface Port Services

Name	Value	Description
RestServer_HTTP	0	HTTP REST API server
RestServer_HTTPS	1	HTTPS REST API server
BrainStem_TCP	2	BrainStem TCP communication
Brain-Stem_DiscoveryRequest	3	BrainStem discovery request port
Brain-Stem_DiscoveryReply	4	BrainStem discovery reply port

Examples

C++

```
// Enable Ethernet interface
ethernet.setEnabled(1);

// Set static IP configuration
ethernet.setNetworkConfiguration(1); // Static mode
unsigned char ip[4] = {192, 168, 1, 100};
unsigned char netmask[4] = {255, 255, 255, 0};
unsigned char gateway[4] = {192, 168, 1, 1};
ethernet.setStaticIPv4Address(ip, 4);
ethernet.setStaticIPv4Netmask(netmask, 4);
ethernet.setStaticIPv4Gateway(gateway, 4);

// Get current IP address
unsigned char current_ip[4];
ethernet.getIPv4Address(current_ip, 4);

// Set hostname
ethernet.setHostname("brainstem-device");
```

Python

```
# Enable Ethernet interface
stem.ethernet.setEnabled(1)

# Set static IP configuration
stem.ethernet.setNetworkConfiguration(1) # Static mode
ip = [192, 168, 1, 100]
netmask = [255, 255, 255, 0]
gateway = [192, 168, 1, 1]
stem.ethernet.setStaticIPv4Address(ip)
stem.ethernet.setStaticIPv4Netmask(netmask)
stem.ethernet.setStaticIPv4Gateway(gateway)

# Get current IP address
current_ip = stem.ethernet.getIPv4Address()
print(current_ip.value)

# Set hostname
stem.ethernet.setHostname("brainstem-device")
```

3.1.6 Equalizer Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Equalizer entity provides a concise interface for controlling equalizer and filter settings for receivers (inputs) and transmitters (outputs). Products supporting Equalizer are capable of applying frequency dependent gain to their signals. This can allow for compensation for signal loss and degradation due to cable quality, cable length and the number of connections. It can also act as a filter implemented in hardware or firmware. Products may implement one or more equalizers; each can be configured using the Equalizer index. Allowed index values are specified in the product data sheet.

Set/Get Transmitter Configuration

```
equalizer [ index ] . getTransmitterConfig <= (unsigned char) config  
equalizer [ index ] . setTransmitterConfig => (unsigned char) config
```

The transmitter is the outgoing portion of the equalizer entity. It is responsible for generating the signal output. Generally, transmitters may have configurations which apply frequency dependent filters, broadband gain, and DC-offsets.

Set/Get Receiver Configuration

```
equalizer [ index ] . getReceiverConfig (unsigned char) channel <= (unsigned char) _  
    ↪ config  
equalizer [ index ] . setReceiverConfig (unsigned char) channel => (unsigned char) _  
    ↪ config
```

The receiver is the incoming portion of the equalizer entity. The receiver equalizer may have configurations which apply frequency dependent filters or broadband gain. Products with more than one receiver may allow individual configuration of the receivers via the channel parameter. Allowed channel and config parameter values are specified in the product data sheet.

Code Examples

C++

```
//Set Transmitter and Receiver configurations  
err = stem.equalizer[0].setTransmitterConfig(transmitterConfig);  
err = stem.equalizer[1].setTransmitterConfig(transmitterConfig);  
err = stem.equalizer[0].setReceiverConfig(eqReceiverChannel, receiverConfig);  
err = stem.equalizer[1].setReceiverConfig(eqReceiverChannel, receiverConfig);  
  
//Get Transmitter and Receiver configurations  
err = stem.equalizer[0].getTransmitterConfig(&transmitterConfig);  
err = stem.equalizer[1].getTransmitterConfig(&transmitterConfig);  
err = stem.equalizer[0].getReceiverConfig(eqReceiverChannel, &receiverConfig);  
err = stem.equalizer[1].getReceiverConfig(eqReceiverChannel, &receiverConfig);
```

Python

```
#Set Transmitter and Receiver configurations
err = stem.equalizer[0].setTransmitterConfig(transmitterConfig);
err = stem.equalizer[1].setTransmitterConfig(transmitterConfig);
err = stem.equalizer[0].setReceiverConfig(eqReceiverChannel, receiverConfig);
err = stem.equalizer[1].setReceiverConfig(eqReceiverChannel, receiverConfig);

#Get Transmitter and Receiver configurations
result = stem.equalizer[0].getTransmitterConfig();
result = stem.equalizer[1].getTransmitterConfig();
result = stem.equalizer[0].getReceiverConfig(eqReceiverChannel);
result = stem.equalizer[1].getReceiverConfig(eqReceiverChannel);
```

3.1.7 HDBaseT Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The **HDBaseT** entity provides link telemetry, remote-end USB topology discovery, and role control for the HDBaseT-USB3 link.

Identification

`stem.system.getSerialNumber()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Fills a caller-provided buffer with up to 6 bytes and reports the actual length unloaded.

`stem.hdbaset.getFirmwareVersion()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Returns a packed 32-bit value (Major: Bits 24-31; Minor: Bits 16-23; Patch: Bits 8-15; Build: Bits 0-7)

Link Status and Health

`stem.hdbaset.getState()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Returns a packed integer describing current link state.

`stem.hdbaset.getCableLength()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Perceived cable length in μm .

`stem.hdbaset.getMSEA()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

`stem.hdbaset.getMSEB()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Mean Squared Error for channel A or B in μdB .

`stem.hdbaset.getRetransmissionRate()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Number of successful messages between retransmission; 0 = no errors, otherwise higher is better.

`stem.hdbaset.getLinkUtilization()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Link utilization in milli-percent (1000 = saturation).

`stem.hdbaset.getEncodingState()` [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

Encoding state (e.g. PAM 8).

Remote USB Topology (Device Trees)

```
stem.hdbaset.getUSB2DeviceTree() [cpp] [python] [.NET] [LabVIEW]
stem.hdbaset.getUSB3DeviceTree() [cpp] [python] [.NET] [LabVIEW]
```

These calls return bytes describing the USB2/USB3 device trees at the HDBaseT endpoint.

Link Role Control

```
stem.hdbaset.getLinkRole() [cpp] [python] [.NET] [LabVIEW]
stem.hdbaset.setLinkRole() [cpp] [python] [.NET] [LabVIEW]
```

Role values are product/firmware specific (e.g., *Auto Detect*, *Leader*, *Follower*).

When in *Auto Detect*, the active role can be read using `getState()`.

3.1.8 I²C Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

BrainStem modules may have the ability to read, write data on up to two I²C busses

Read

```
i2c [ index ] . read => (unsigned char) address, (unsigned char) length <= (unsigned_
↳char*) data
```

Reads up to 26 bytes from the I²C bus given by the index. The parameters are the I²C address of the device on the bus, and the number of bytes to read. The result is the data that was read or an error.

Write

```
i2c [ index ] . write => (unsigned char) address, (unsigned char) length, (unsigned_
↳char*) data <= (unsigned char) result
```

Writes up to 26 bytes to the I²C bus given by the index. The parameters are the I²C address of the device on the bus, the number of bytes to write, and the data to write. The result is the result error condition or none.

Set Pullup

```
i2c [ index ] . setPullup => (unsigned char) bool
```

Sets software controlled pullup state on modules which have software controllable pullups. This setting is saved when a call to `system.save` is made so that Pullup settings on bus 0 can persist.

Speed (Get/Set)

```
i2c [ index ] . getSpeed <= (unsigned char) speed
i2c [ index ] . setSpeed => (unsigned char) speed
```

Returns or sets the I2C bus communication speed. The speed setting controls the clock frequency for I2C transactions on the specified bus.

Table 3: I2C Speed Options

Value	Name	Description
0	Default	Default I2C speed (typically 100 kHz)
1	100Khz	Standard I2C speed (100 kHz)
2	400Khz	Fast I2C speed (400 kHz)
3	1000Khz	Fast Plus I2C speed (1 MHz)

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.
char buff[2];
stem.i2c[0].read(0x42, 0x02, buff); // reads two from device with address 0x42.
char wrbuff[] = {0xBE, 0xEF};
stem.i2c[0].write(0x42, 0x02, wrbuff); // writes 0xBEEF to the device with address_
→0x42
stem.i2c[0].setPullup(true); //enables pullup on bus 0
stem.i2c[0].setSpeed(1); // sets I2C speed to 100 kHz (i2cSpeed_100Khz)
stem.i2c[0].setSpeed(2); // sets I2C speed to 400 kHz (i2cSpeed_400Khz)
stem.i2c[0].setSpeed(3); // sets I2C speed to 1 MHz (i2cSpeed_1000Khz)
unsigned char speed;
stem.i2c[0].getSpeed(&speed); // gets current I2C speed
```

Python

The length parameter for I²C write is not used in python.

```
result = stem.i2c[0].read(0x42, 0x02) # reads two bytes from the i2c bus. The value...
      ↳ is given in result.value
print result.value
err = stem.i2c[0].write(0x42, b'\xbe\xef') # writes b'\xbe\xef' to the i2c bus.
print err
err = stem.i2c[0].setPullup(True)
print err
err = stem.i2c[0].setSpeed(1) # sets I2C speed to 100 kHz (i2cSpeed_100Khz)
print err
speed = stem.i2c[0].getSpeed() # gets current I2C speed
print speed.value
```

3.1.9 Mux Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs.

Some mux entities can simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

Channel (Set/Get)

```
mux [ index ] . setChannel => (unsigned char) channel
mux [ index ] . getChannel <= (unsigned char) channel
```

Gets/Sets the currently selected channel

Enable/Disable (Set/Get)

```
mux [ index ] . setEnable => (unsigned char) enable
mux [ index ] . getEnable <= (unsigned char) enable
```

Enables/Disables the mux.

Get Channel Voltage (Get)

```
mux [ index ] . getChannelVoltage <= ((unsigned char) channel, (unsigned char) voltage)
```

Returns the voltage of the supplied channel.

Configuration (Get/Set)

```
mux [ index ] . getConfiguration <= (unsigned char) configuration
mux [ index ] . setConfiguration => (unsigned char) configuration
```

Gets or sets the mux operating configuration (for example default routing, split mode, or port-priority behavior on products that support it). Product-specific option values and bit layouts are described in each module's documentation.

Split Mode (Get/Set)

```
mux [ index ] . getSplitMode <= (unsigned int) split_mode
mux [ index ] . setSplitMode => (unsigned int) split_mode
```

After split-style configuration is enabled on supported hardware, the split mode word defines how functional groups are assigned to mux channels. See the product datasheet for field definitions.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get calls will fill the variable with the returned value.

err = stem.mux[0].getChannel(&channel);
err = stem.mux[0].setChannel(1);
err = stem.mux[0].setEnabled(1);
err = stem.mux[0].setEnabled(0);
err = stem.mux[0].getChannel(1, &voltage);
err = stem.mux[0].setChannel(3);
```

Python

```
result = stem.mux[0].getChannel(&channel);
print result.value
err = stem.mux[0].setChannel(1)
err = stem.mux[0].setEnabled(1)
err = stem.mux[0].setEnabled(0)
voltage = stem.mux[0].getChannel(1)
print voltage.value
err = stem.mux[0].setChannel(3)
```

3.1.10 Pointer Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Access the reflex pad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via `setOffset`. A read or write of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not. This allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

Offset (Get/Set)

```
pointer [ index ] . getOffset <= (unsigned char) Offset
pointer [ index ] . setOffset => (unsigned char) offset
```

Gets or sets the current cursor position for the pointer.

Mode (Get/Set)

```
pointer [ index ] . getMode <= (unsigned char) mode
pointer [ index ] . setMode => (unsigned char) mode
```

Get or set the pointer mode, static (0 default) or incrementing (1).

Char (Get/Set)

```
pointer [ index ] . getChar <= (unsigned char) value
pointer [ index ] . setChar => (unsigned char) value
```

Get or set a character value into the scratchpad at the current pointer offset. This will increment the pointer by 1 byte if the pointer mode is set to increment.

Short (Get/Set)

```
pointer [ index ] . getShort <= (unsigned short) value
pointer [ index ] . setShort => (unsigned short) value
```

Get or set a short value into the scratchpad at the current pointer offset. This will increment the pointer by 2 bytes if the pointer mode is set to increment.

Int (Get/Set)

```
pointer [ index ] . getInt <= (unsigned int) value
pointer [ index ] . setInt => (unsigned int) value
```

Get or set an int value into the scratchpad at the current pointer offset. This will increment the pointer by 4 bytes if the pointer mode is set to increment.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get calls will fill the variable with the returned value.

stem.pointer[0].getOffset(&offset);
stem.pointer[0].setOffset(4);
stem.pointer[0].getMode(&mode);
stem.pointer[1].setMode(1);
stem.pointer[1].getChar(&value);
stem.pointer[1].setChar(6);
stem.pointer[1].getShort(&value);
stem.pointer[1].setShort(600);
stem.pointer[1].getInt(&value);
stem.pointer[1].setInt(600000);
```

Python

```
result = stem.pointer[0].getOffset()
print result.value
err = stem.pointer[0].setOffset(4)
result = stem.pointer[0].getMode(mode)
print result.value
err = stem.pointer[1].setMode(1)
result = stem.pointer[1].getChar()
print result.value
err = stem.pointer[1].setChar(6)
result = stem.pointer[1].getShort()
result.value
err = stem.pointer[1].setShort(600)
result = stem.pointer[1].getInt()
result.value
result = stem.pointer[1].setInt(600000)
```

3.1.11 PoE Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The **PoE** entity provides Power over Ethernet control and telemetry on supported products. Use this entity to enable/disable PoE pairs, select power modes, inspect negotiated classes, and read per-pair electrical measurements and accumulated power.

- *pair* = 0 = wire pair 1/2
 - *pair* = 1 = wire pair 3/4
-

Pair Enable/Disable

```
stem.poe.getPairEnabled(pair) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
stem.poe.setPairEnabled(pair, enable) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
1 = Enabled, 0 = Disabled.
```

Power Mode and Power State

```
stem.poe.getPowerMode() \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
stem.poe.setPowerMode(value) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
stem.poe.getPowerState() \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
Power Mode (commanded): PD, PSE, Auto, Off.  
Power State (actual): PD, PSE, Off.
```

Classification and Detection

```
stem.poe.getPairSourcingClass(pair) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
stem.poe.setPairSourcingClass(pair, value) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
Sourcing class: The PoE class offered by the device when acting as a PSE.  
stem.poe.getPairRequestedClass(pair) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
Requested class: The PoE class requested by the device when acting as a PD.  
stem.poe.getPairDiscoveredClass(pair) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
Discovered class (PSE/PD): The negotiated PoE class result.  
stem.poe.getPairDetectionStatus(pair) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]  
Current per-pair detection status.
```

Table 4: Detection status values

Unknown
Short circuit
Open circuit
Low resistance
High resistance
Valid
Switch Failure

Electrical Measurements (per pair)

`stem.poe.getPairVoltage(pair) [cpp] [python] [.NET] [LabVIEW]`

Voltage: μV

`stem.poe.getPairCurrent(pair) [cpp] [python] [.NET] [LabVIEW]`

Current: μA

`stem.poe.getPairResistance(pair) [cpp] [python] [.NET] [LabVIEW]`

Resistance: $\text{m}\Omega$

`stem.poe.getPairCapacitance(pair) [cpp] [python] [.NET] [LabVIEW]`

Capacitance: nF

`stem.poe.getPairPower(pair) [cpp] [python] [.NET] [LabVIEW]`

Power: mW (approx. Voltage * Current for the pair)

Total Power (instantaneous)

`stem.poe.getTotalPower() [cpp] [python] [.NET] [LabVIEW]`

Returns the sum of both pairs' instantaneous power (mW).

Accumulated Power (energy counters)

`stem.poe.getPairAccumulatedPower(pair) [cpp] [python] [.NET] [LabVIEW]`

`stem.poe.setPairAccumulatedPower(pair, power) [cpp] [python] [.NET] [LabVIEW]`

`stem.poe.getTotalAccumulatedPower() [cpp] [python] [.NET] [LabVIEW]`

`stem.poe.setTotalAccumulatedPower(power) [cpp] [python] [.NET] [LabVIEW]`

* All values are in mWh

* Setting total or pair accumulated power resets the corresponding accumulator.

3.1.12 Port Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Port Entity provides control over the most basic items related to a USB Port. This includes actions ranging from a complete port enable and disable to the individual interface control. Voltage and current measurements are also included for devices which support the Port Entity.

Port Enable/Disable (Get/Set)

```
port [index] . getEnabled <= (unsigned char) enabled
port [index] . setEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over an entire port for a provided index (Power, Data, CC and Vconn). Values either passed in or returned are treated as boolean values.

Power Enable/Disable (Get/Set)

```
port [index] . getPowerEnabled <= (unsigned char) enabled  
port [index] . setPowerEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the power for a provided index (Vbus). Values either passed in or returned are treated as boolean values.

Data Enable/Disable (Get/Set)

```
port [index] . getDataEnabled <= (unsigned char) enabled  
port [index] . setDataEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the data lines for a provided index (High Speed (HS) and Super Speed (SS)). Values either passed in or returned are treated as boolean values.

High Speed (HS) Data Enable/Disable (Get/Set)

```
port [index] . getDataHSEnabled <= (unsigned char) enabled  
port [index] . setDataHSEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the High Speed (HS) data lines for a provided index (HS1 and HS2). Values either passed in or returned are treated as boolean values.

High Speed 1 (HS1) Data Enable/Disable (Get/Set)

```
port [index] . getDataHS1Enabled <= (unsigned char) enabled  
port [index] . setDataHS1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the High Speed 1 (HS1) data lines for a provided index. Values either passed in or returned are treated as boolean values.

High Speed 2 (HS2) Data Enable/Disable (Get/Set)

```
port [index] . getDataHS2Enabled <= (unsigned char) enabled  
port [index] . setDataHS2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the High Speed 2 (HS2) data lines for a provided index. Values either passed in or returned are treated as boolean values.

Super Speed (SS) Data Enable/Disable (Get/Set)

```
port [index] . getDataSSEnabled <= (unsigned char) enabled
port [index] . setDataSSEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Super Speed (SS) data lines for a provided index (SS1 and SS2). Values either passed in or returned are treated as boolean values.

Super Speed 1 (SS1) Data Enable/Disable (Get/Set)

```
port [index] . getDataSS1Enabled <= (unsigned char) enabled
port [index] . setDataSS1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Super Speed 1 (SS1) data lines for a provided index. Values either passed in or returned are treated as boolean values.

Super Speed 2 (SS2) Data Enable/Disable (Get/Set)

```
port [index] . getDataSS2Enabled <= (unsigned char) enabled
port [index] . setDataSS2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Super Speed 2 (SS2) data lines for a provided index. Values either passed in or returned are treated as boolean values.

Vconn Enable/Disable (Get/Set)

```
port [index] . getVconnEnabled <= (unsigned char) enabled
port [index] . setVconnEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Vconn lines for a provided index (Vconn1 and Vconn2 (only one ever exists)). Values either passed in or returned are treated as boolean values.

Vconn 1 Enable/Disable (Get/Set)

```
port [index] . getVconn1Enabled <= (unsigned char) enabled
port [index] . setVconn1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Vconn 1 lines for a provided index. Values either passed in or returned are treated as boolean values.

Vconn 2 Enable/Disable (Get/Set)

```
port [index] . getVconn2Enabled <= (unsigned char) enabled
port [index] . setVconn2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Vconn 2 lines for a provided index. Values either passed in or returned are treated as boolean values.

CC Enable/Disable (Get/Set)

```
port [index] . getCCEnabled <= (unsigned char) enabled
port [index] . setCCEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the CC lines for a provided index (CC1 and CC2). Values either passed in or returned are treated as boolean values.

CC 1 Enable/Disable (Get/Set)

```
port [index] . getCC1Enabled <= (unsigned char) enabled
port [index] . setCC1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the CC 1 lines for a provided index. Values either passed in or returned are treated as boolean values.

CC 2 Enable/Disable (Get/Set)

```
port [index] . getCC2Enabled <= (unsigned char) enabled
port [index] . setCC2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the CC 2 lines for a provided index. Values either passed in or returned are treated as boolean values.

Vbus Voltage/Current (Get)

```
port [index] . getVbusVoltage <= (unsigned int) microvolts
port [index] . getVbusCurrent <= (unsigned int) microamps
```

Provides access to the last read values of Voltage (in microvolts) and Current (in microamps) for the Vbus lines.

Vconn Voltage/Current (Get)

```
port [index] . getVconnVoltage <= (unsigned int) microvolts
port [index] . getVconnCurrent <= (unsigned int) microamps
```

Provides access to the last read values of Voltage (in microvolts) and Current (in microamps) for the Vconn lines.

Vbus Accumulated Power (Get/Reset)

```
port [index] . getVbusAccumulatedPower <= (unsigned int) milliwatthours
port [index] . resetVbusAccumulatedPower => (void)
```

Returns the accumulated power (energy) sank or sourced by the Vbus line for the given port in units of milliWatt-hours.

Vconn Accumulated Power (Get/Reset)

```
port [index] . getVconnAccumulatedPower <= (unsigned int) milliwatthours
port [index] . resetVconnAccumulatedPower => (void)
```

Returns the accumulated power (energy) sank or sourced by the Vconn line for the given port in units of milliWatt-hours.

Port Name (Get/Set)

```
port [index] . getName <= (unsigned char[]) name
port [index] . setName => (unsigned char[]) name
```

Allows for setting a friendly name to the port with a 32 character limit.

Downstream Data Speed (Get)

```
port [index] . getDataSpeed <= (unsigned int)
```

Gets the speed of the enumerated device.

Data Speed	Bit	Value	Define
1.5 Mbit/s	0	0/1	portDataSpeed_ls_1p5M_Bit
12 Mbit/s	1	0/1	portDataSpeed_fs_12M_Bit
480 Mbit/s	2	0/1	portDataSpeed_hs_480M_Bit
5 Gbit/s	3	0/1	portDataSpeed_ss_5G_Bit
10 Gbit/s	4	0/1	portDataSpeed_ss_10G_Bit
USB 2.0	6	0/1	portDataSpeed_Connected_2p0_Bit
USB 3.0	7	0/1	portDataSpeed_Connected_3p0_Bit

Allocated Power (Get)

```
port[index] . getAllocatedPower <= (unsigned int) milliwatts
```

Returns the currently allocated power for the port in milliwatts. This represents the power that has been allocated to this port from the available power budget.

Available Power (Get)

```
port[index] . getAvailablePower <= (unsigned int) milliwatts
```

Returns the available power for the port in milliwatts. This represents the power that is available for allocation to this port.

CC1/CC2 Accumulated Power (Get/Set)

```
port[index] . getCC[1|2]AccumulatedPower <= (unsigned int) milliwatt_hours  
port[index] . setCC[1|2]AccumulatedPower => (unsigned int) milliwatt_hours
```

Returns or sets the accumulated power consumption for the CC1/CC2 line in milliwatt-hours. This tracks the total energy consumed over time.

CC1/CC2 Current (Get)

```
port[index] . getCC[1|2]Current <= (unsigned int) microamps
```

Returns the current flowing through the CC1/CC2 line in microamps. This is used for USB-C power delivery monitoring.

CC1/CC2 State (Get)

```
port[index] . getCC[1|2]State <= (unsigned int) state
```

Returns the current state of the CC1/CC2 line. The state is a 2-byte value where the upper byte represents the remote device state (device on the other end of the cable) and the lower byte represents the local hub state.

Table 5: CC State Values

Value	Name	Description
0	None	No connection or state
1	Invalid	Invalid or error state
2	RpDefault	Default pull-up resistor (500/900/1500mA)
3	Rp1p5	1.5 Amp pull-up resistor
4	Rp3p0	3.0 Amp pull-up resistor
5	Rd	Pull-down resistor (sink device)
6	Ra	Audio accessory resistor
7	Managed	Managed by firmware
8	Unknown	Unknown state

CC1/CC2 Voltage (Get)

```
port[index] . getCC[1|2]Voltage <= (unsigned int) microvolts
```

Returns the voltage on the CC1/CC2 line in microvolts. This is used for USB-C power delivery monitoring.

CC Current Limit (Get/Set)

```
port[index] . getCCCurrentLimit <= (unsigned char) limit
port[index] . setCCCurrentLimit => (unsigned char) limit
```

Returns or sets the current limit enumeration for the CC lines. This sets the maximum current value that will be advertised for power delivery negotiation.

Table 6: CC Current Limit Options

Value	Name	Description
0	None	No current limit
1	Default	Default current limit (500/900mA)
2	1p5	1.5 Amp current limit
3	3p0	3.0 Amp current limit

Current Limit Mode (Get/Set)

```
port[index] . getCurrentLimitMode <= (unsigned char) mode
port[index] . setCurrentLimitMode => (unsigned char) mode
```

Returns or sets the current limit mode for the port. This controls how the current limit is applied and enforced.

Data HS Routing Behavior (Get/Set)

```
port[index] . getDataHSRoutingBehavior <= (unsigned char) behavior
port[index] . setDataHSRoutingBehavior => (unsigned char) behavior
```

Returns or sets the high-speed data routing behavior for the port. This controls how high-speed data signals are routed through the port.

Table 7: Data HS Routing Behavior Options

Value	Name	Description
0	FollowCC	Auto follow CC line routing
1	Side1	Route to side 1 only
2	Side2	Route to side 2 only
3	Shorted	Route to both sides (shorted)

Data Role (Get)

```
port[index] . getDataRole <= (unsigned char) role
```

Returns the current data role of the port. This indicates whether the port is acting as a host or device for data communication.

Data SS Routing Behavior (Get/Set)

```
port[index] . getDataSSRoutingBehavior <= (unsigned char) behavior
port[index] . setDataSSRoutingBehavior => (unsigned char) behavior
```

Returns or sets the super-speed data routing behavior for the port. This controls how super-speed data signals are routed through the port.

Table 8: Data SS Routing Behavior Options

Value	Name	Description
0	FollowCC	Auto follow CC line routing
1	Side1	Route to side 1 only
2	Side2	Route to side 2 only

Port Errors (Get)

```
port[index] . getErrors <= (unsigned int) errors
```

Returns the current error status for the port. The error value is a bitfield indicating various port-related error conditions.

HS Boost (Get/Set)

```
port[index] . getHSBoost <= (unsigned char) boost
port[index] . setHSBoost => (unsigned char) boost
```

Returns or sets the high-speed boost setting for the port. This controls signal amplification for high-speed data transmission.

Port Mode (Get/Set)

```
port[index] . getMode <= (unsigned int) mode
port[index] . setMode => (unsigned int) mode
```

Returns or sets the operating mode for the port. The mode is a bitfield that controls various port behaviors and capabilities.

Power Limit Mode (Get/Set)

```
port[index] . getPowerLimitMode <= (unsigned char) mode
port[index] . setPowerLimitMode => (unsigned char) mode
```

Returns or sets the power limit mode for the port. This controls how the power limit is applied and enforced.

Power Mode (Get/Set)

```
port[index] . getPowerMode <= (unsigned char) mode
port[index] . setPowerMode => (unsigned char) mode
```

Returns or sets the power mode for the port. This controls the power delivery behavior and capabilities.

SBU1 Voltage (Get)

```
port[index] . getSBU1Voltage <= (unsigned int) microvolts
```

Returns the voltage on the SBU1 line in microvolts. SBU (Sideband Use) lines are used for alternate modes in USB-C.

SBU2 Voltage (Get)

```
port[index] . getSBU2Voltage <= (unsigned int) microvolts
```

Returns the voltage on the SBU2 line in microvolts. SBU (Sideband Use) lines are used for alternate modes in USB-C.

Port State (Get)

```
port[index] . getState <= (unsigned int) state
```

Returns the current state of the port. The state is a bitfield indicating the status of various port components and features.

Voltage Setpoint (Get/Set)

```
port[index] . getVoltageSetpoint <= (unsigned int) millivolts
port[index] . setVoltageSetpoint => (unsigned int) millivolts
```

Returns or sets the voltage setpoint for the port in millivolts. This controls the target voltage for the port's power output.

Vbus Accumulated Power (Set)

```
port[index] . setVbusAccumulatedPower => (unsigned int) milliwatt_hours
```

Sets the accumulated power consumption for the Vbus line in milliwatt-hours. This is used to reset or initialize the power tracking.

Vconn Accumulated Power (Set)

```
port[index] . setVconnAccumulatedPower => (unsigned int) milliwatt_hours
```

Sets the accumulated power consumption for the Vconn line in milliwatt-hours. This is used to reset or initialize the power tracking.

3.1.13 Power Delivery Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface. This Entity will allow you to directly access the vast landscape of PD.

When the capabilities of a PD system are fully realized everything in the system is “smart”. That includes the device, the host and even the cable. All of these elements contain electronics that identify themselves and what they are capable of doing. Because of this complexity it is important to align on a few terms that will be used throughout this Entity.

Partner This refers to the side of the PD connection in question. The possible options for this parameter are.

- **Local** Indicates the context/perspective of the Acriname device you are communicating with through a BrainStem connection.
- **Remote** The context/perspective of anything other than the Acriname device.

Partner Type	Value	Define
Local	0	powerdeliveryPartnerLocal
Remote	1	powerdeliveryPartnerRemote

Power Role Indicates the direction of power. This value is typically used in the context of a “Partner”. i.e. The remote partner is sinking, which would mean the local partner is sourcing. The possible options for this context are:

- **Sink** Indicates that the partner is taking power in/from.
- **Source** Indicates that the partner is providing power out/to.

Power Roles are also used in the context of what a port is capable of doing.

- **Sink** Device is capable of consuming power.
- **Source** Device is capable of producing power.
- **Sink/Source** Device is capable of both consuming or producing power. Dual Role Port (DRP)

Power Role	Value	Define
Disabled	0	powerdeliveryPowerRoleDisabled
Source	1	powerdeliveryPowerRoleSource
Sink	2	powerdeliveryPowerRoleSink
Source/Sink	3	powerdeliveryPowerRoleSourceSink

Power Data Objects (PDO)

- PDO's define what a device is capable of doing in the world of Power Delivery. PDO's are bit packed integers defined by the PD Specification which vary in meaning based on the type of PDO.

Request Data Objects (RDO)

- RDO's are the final agreement after successful Power Delivery negotiations. This RDO is always sent by the sinking device and is the result of the sources advertised PDO's and the needs/requirements of the sinking device. Only one RDO exists per valid connection.

Connection State (Get)

```
pd[x] . getConnectionState => (unsigned char) state
```

Gets the type of connection as defined by the Power Delivery Specification. The most common connections types are: Not Attached, Sourcing and Sinking.

Power Data Object (Get/Set)

```
pd[x] . getPowerDataObject => (unsigned int) pdo
pd[x] . setPowerDataObject <= (unsigned int) pdo
```

Gets and Sets the PDO for a given pd[x] instance, partner and power role.

For any one connection there are 4 locations in which PDOs exist: Remote Sink, Remote Source, Local Sink, and Local Source. Within each of PDO locations up to 7 PDOs can be defined.

Set calls are only allowed on Local Partner assuming the BrainStem device supports this feature.

Number of Power Data Objects (Get)

```
pd[x] . getNumberOfPowerDataObjects => (unsigned int) pdoCount
```

As previously stated 7 PDO's can be defined per location; however, it is only required that there be 1. This API allows you to get the number of PDO's available for a given partner and power role.

Reset Power Data Objects (Set)

```
pd[x] . resetPowerDataObjectToDefault => (void)
```

Resets the local partner PDO for a given power role and index.

Power Data Object List (Get)

```
pd[x] . getPowerDataObjectList => (unsigned int [MAX_PDOS]) list
```

Returns a list of all PDOs for a given pd[x] instance. This is equivalent to calling getPowerDataObject on all possible configurations.

Power Data Objects Enabled (Get/Set)

```
pd[x] . getPowerDataObjectEnabled => (unsigned char) enable  
pd[x] . setPowerDataObjectEnabled <= (unsigned char) enable
```

Acroname products which support this feature can selectively enable and disable its local PDO's. In that, if the local source location has 7 PDO's, the user could disable all but the first PDO from being advertised by disabling them.

Power Data Object Enabled List (Get)

```
pd[x] . getPowerDataObjectEnabledList => (unsigned char) enableList
```

Convenience function to getPowerDataObjectEnabled. Returns a bit packed representation of the PDO enabled status.

Request Data Object (Get/Set)

```
pd[x] . getRequestDataObject => (unsigned int) rdo  
pd[x] . setRequestDataObject <= (unsigned int) rdo
```

Gets and Sets the RDO for a given pd[x] instance and partner

Set calls are only possible on a local sinking partner assuming the BrainStem device supports this feature.

Power Role (Get/Set)

```
pd[x] . getPowerRole => (unsigned char) role  
pd[x] . setPowerRole <= (unsigned char) role
```

The power role defines the type of PD connections the device supports. Devices can be disabled, sinking, sourcing or dual role ports (capable of sinking or sourcing).

Power Role Preferred (Get/Set)

```
pd[x] . getPowerRolePreferred => (unsigned char) role
pd[x] . setPowerRolePreferred <= (unsigned char) role
```

Dual role port typically have a preference of whether they are sinking or sourcing. For instance battery powered devices typically prefer to sink power since they have a finite amount of battery power; however, many of them can source power if requested to do so.

Cable Voltage Maximum (Get)

```
pd[x] . getCableVoltageMax => (unsigned char) voltage
```

Returns the maximum amount of voltage the attached cable is capable of handling. This information is defined in the emark of the cable and is used during PD negotiations for PDO compatibility.

Cable Current Maximum (Get)

```
pd[x] . getCableCurrentMax => (unsigned char) current
```

Returns the maximum amount of current the attached cable is capable of handling. This information is defined in the emark of the cable and is used during PD negotiations for PDO compatibility.

Cable Speed Maximum (Get)

```
pd[x] . getCableSpeedMax => (unsigned char) speed
```

Returns the maximum speed the attached cable is capable of handling. This information is defined in the emark of the cable.

Cable Type (Get)

```
pd[x] . getCableType => (unsigned char) cable
```

Returns whether the cable is active or passive and if it is emarked.

Table 9: Cable Type Options

Value	Name	Description
0	Invalid	Invalid or unknown cable type
1	Passive	Passive cable (no active electronics)
2	Active	Active cable (contains active electronics)

Cable Orientation (Get)

```
pd[x] . getCableOrientation => (unsigned char) orientation
```

Indicates which side of the connection is being used for PD negotiations. This is based on physical CC strapping within the cable.

Request (Set)

```
pd[x] . request <= (unsigned char) request
```

Allows access to specific requests which are built into the PD specification. It is important to remember that these are requests and are not guaranteed to occur. Examples are resets, power, data, and VCONN role swaps.

Table 10: Requests

Request	Value	Define
Hard Reset	0	pdRequestHardReset
Soft Reset	1	pdRequestSoftReset
Data Reset	2	pdRequestDataReset
Power Role Swap	3	pdRequestPowerRoleSwap
Power Fast Role Swap	4	pdRequestPowerFastRoleSwap
Data Role Swap	5	pdRequestDataRoleSwap
Vconn Swap	6	pdRequestVconnSwap
Sink GoTo Minimum	7	pdRequestSinkGoToMinimum
Remote Source Power Data Objects	8	pdRequestRemoteSourcePowerDataObjects
Remote Sink Power Data Objects	9	pdRequestRemoteSinkPowerDataObjects
Remote Source Extended Capabilities	10	pdRequestRemoteSourceExtendedCapabilities
Remote Sink Extended Capabilities	11	pdRequestRemoteSinkExtendedCapabilities
Status	12	pdRequestStatus
PPS Status	13	pdRequestPPSStatus
Battery Capabilities	14	pdRequestBatteryCapabilities
Battery Status	15	pdRequestBatteryStatus
Manufacturer Info Sop	16	pdRequestManufacturerInfoSop
Manufacturer Info Sop'	17	pdRequestManufacturerInfoSopp
Manufacturer Info Sop''	18	pdRequestManufacturerInfoSoppp
Discover Identity Sop	19	pdRequestDiscoverIdentitySop
Discover Identity Sop'	20	pdRequestDiscoverIdentitySopp
Discover Identity Sop''	21	pdRequestDiscoverIdentitySoppp
Revision	22	pdRequestRevision

Request Status (Get)

```
pd[x] . requestStatus => (unsigned char) status
```

Returns the most recent status for a given pd[x] instance. This is usually paired with the request command since they are not guaranteed and are asynchronous.

Flag Mode (Get/Set)

```
pd[x] . getFlagMode => (unsigned char) mode
pd[x] . setFlagMode <= (unsigned char) mode
```

Allows get and set of a flag configuration for a given USB Power Delivery Flag. The following flags can be configured to the following different modes:

Table 11: Flags

Flag	Value	Define
Dual Role Data	1	pdFlagDualRoleData
Dual Role Power	2	pdFlagDualRolePower
Unconstrained Power	3	pdFlagUnconstrainedPower
Suspend Possible	4	pdFlagSuspendPossible
USB Com Possible	5	pdFlagUSBComPossible
Unchunked Message Support	6	pdFlagUnchunkedMessageSupport
Higher Capability	7	pdFlagHigherCapability
Capability Mismatch	8	pdFlagCapabilityMismatch
Giveback Flag	9	pdFlagGivebackFlag

Table 12: Modes

Mode	Value	Description
Disabled	0	Flag will always report 0
Enabled	1	Flag will always report 1
Auto	2	Flag will show 0 or 1 correctly according to the rest of the hubs state/config

Override (Get/Set)

```
pd[x] . getOverride => (unsigned int) overrides
pd[x] . setOverride <= (unsigned int) overrides
```

Returns or sets the override configuration for the Power Delivery connection. Overrides allow bypassing certain Power Delivery protocol behaviors for testing or special applications.

Table 13: Override Options

Bit	Name	Description
0	CableCurrent	overrides the cable current limiting to 3A unless it's an emarked cable
1	PortPower	Overrides the port power budgeting and just allows full power always
2	AutoDiscovery	Overrides the auto discovery feature. With override true the hub will only establish a basic power connection and wont ask for additional vendor information.
3	DataPathDisable	Override normal disabling of the data path when the PD Data Role and port configured data role differ

Link State (Get)

```
pd[x] . getLinkState => (unsigned int) bitmask
```

Returns a bitmask describing PD link-layer state. See product and release notes for bit definitions.

Attach Time Elapsed (Get)

```
pd[x] . getAttachTimeElapsed => (unsigned int [2]) elapsed
```

Returns how long the port has been attached, as two unsigned integers: seconds, then microseconds.

Power Role Capabilities (Get)

```
pd[x] . getPowerRoleCapabilities => (unsigned char) capabilities
```

Returns the power roles that the local partner may advertise (for example from CC strapping).

Data Role Capabilities (Get)

```
pd[x] . getDataRoleCapabilities => (unsigned char) capabilities
```

Returns the data roles that the local partner may advertise.

Protocol Revision Maximum (Get/Set)

```
pd[x] . getProtocolRevisionMax => (unsigned int) revision
pd[x] . setProtocolRevisionMax <= (unsigned int) revision
```

Gets or sets the maximum USB Power Delivery protocol revision used as a negotiation ceiling (for example Rev 2 vs Rev 3 constants such as `pdProtocolRevision2` and `pdProtocolRevision3` in protocol headers). This value constrains how the stack advertises protocol revision during negotiation; it is **not** the product hardware revision or firmware build number. Only valid API revision values for your release should be written; invalid values may be rejected or clamped by firmware.

Peak Current Configuration (Get/Set)

```
pd[x] . getPeakCurrentConfiguration => (unsigned char) configuration
pd[x] . setPeakCurrentConfiguration <= (unsigned char) configuration
```

Gets or sets the peak current configuration for the local source partner. Peak current configuration describes allowable overload tolerance relative to the advertised maximum current.

Fast Role Swap Current (Get/Set)

```
pd[x] . getFastRoleSwapCurrent => (unsigned char) current
pd[x] . setFastRoleSwapCurrent <= (unsigned char) current
```

Gets or sets the fast-role-swap current parameter for the local sink: the current the sink needs available to complete a successful fast role swap.

3.1.14 Rail Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Rail entity provides power control to connected devices on some modules. Check the module datasheet to determine if the module has this capability.

the Rail entity controls power provided to downstream devices, it has the ability to enable and disable power, can read voltage on the rail, and provides current consumption information on some modules. There are additional capabilities that certain modules provide which enhance basic power delivery through Kelvin sensing, or by bringing online separate power management functionality.

Certain modules may provide more than one power rail. These are independently controlled and can be accessed via the entity index.

Current (Get)

```
rail[ index ] . getCurrent <= (int) microamps
```

Returns the current consumption of the device attached to the rail. This can be a positive or negative value, and is reported in microamps.

Current Limit (Get/Set)

```
rail [ index ] . getCurrentLimit <= (int) microamps
rail [ index ] . setCurrentLimit => (int) microamps
```

Available on some modules, check your module datasheet. This control gets or sets the maximum current draw for the given power rail in microamps.

Temperature (Get)

```
rail [ index ] . getTemperature <= (int) microcelsius
```

Some modules have a rail temperature measurement. This command gets the current rail temperature in microcelsius.

Enable (Get/Set)

```
rail [ index ] . setEnable => (unsigned char) enable  
rail [ index ] . getEnable <= (unsigned char) enable
```

Setting Enable

Some rails can be enabled or disabled. The enable value is treated as a boolean 1 will enable the rail and 0 will disable it. Check the module datasheet to determine if this functionality is available for the given rail.

Getting Enable

If a rail can be enabled or disabled, getting the Enable setting will return a 1 if the rail is enabled or 0 otherwise.

Voltage (Get/Set)

```
rail [ index ] . setVoltage => (int) microvolts  
rail [ index ] . getVoltage <= (int) microvolts
```

Some rails are variable voltage rails, and users can set the rails to supply voltage at range of voltage values. Check the module datasheet for the rail voltage limits, and settings.

Setting Rail Voltage

Setting this value will cause the rail to supply the requested voltage, if it is within the settings defined in the datasheet.

Getting Rail Voltage

Getting this value will return the current voltage setpoint for the rail in microvolts. If the given rail is fixed, it returns the fixed voltage setting for the given rail.

Kelvin Sensing (Get/Set)

```
rail [ index ] . setKelvinSensingEnable => (unsigned char) enable  
rail [ index ] . getKelvinSensingEnable <= (unsigned char) enable
```

Some rails have kelvin sensing capabilities. See the module datasheet for more information about using kelvin sensing in your application.

Setting Kelvin Sensing mode

Setting this value to 1 will enable Kelvin sensing on this rail.

Getting Kelvin Sensing mode

Getting this value will return whether kelvin sensing is enabled on the rail. 1 is enabled 0 is disabled.

Kelvin Sensing State (Get)

```
rail [ index ] . getKelvinSensingState <= (unsigned char) state
```

When a rail is capable of Kelvin sensing, under certain error conditions kelvin sensing may be disabled by the system. This command returns the current kelvin sensing state of the rail, either enabled or disabled.

Operational Mode (Get/Set)

```
rail [ index ] . setOperationalMode => (unsigned char) mode
rail [ index ] . getOperationalMode <= (unsigned char) mode
```

Certain modules have multiple power regulation stages that can affect the behavior of the supplied rail voltage and current. This command sets and gets the preferred mode of operation for the given rail. Check the module datasheet for details on the capabilities and behavior of these operational modes.

Operational State (Get)

```
rail [ index ] . getOperationalState <= (unsigned char) mode
```

When a rail is capable of multiple operational modes, getting this value will return the current operational state of the rail, this can indicate error conditions, or a certain operational mode if the rail is in an automatic behavior.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get commands fill the variable with the returned value.

stem.rail[0].getCurrent(microamps);
stem.rail[0].setCurrentLimit(limit);
stem.rail[0].getCurrentLimit(limit);
stem.rail[0].getTemperature(microcelsius);
stem.rail[0].setEnabled(1); //enables rail.
stem.rail[0].getEnable(bEnable);
stem.rail[1].setVoltage(2000000); // set rail to 2 volts.
stem.rail[1].getVoltage(microvolts);
stem.rail[0].setKelvinSensingEnable(1); // enable kelvin sensing.
stem.rail[0].getKelvinSensingEnable(bEnabled);
stem.rail[0].getKelvinSensingState(bEnabled);
stem.rail[0].setOperationalMode(auto);
stem.rail[0].getOperationalMode(mode);
stem.rail[0].getOperationalState(state);
```

Python

```
microamps = stem.rail[0].getCurrent()
print microamps.value
stem.rail[0].setCurrentLimit(limit)
limit = stem.rail[0].getCurrentLimit()
print limit.value
temperature = stem.rail[0].getTemperature()
print temperature.value
stem.rail[0].setEnabled(1) //enables rail.
bEnable = stem.rail[0].getEnable()
print bEnable.value
stem.rail[0].setVoltage(2000000) // set rail to 2 volts.
microvolts = stem.rail[0].getVoltage(microvolts)
print microvolts.value
stem.rail[0].setKelvinSensingEnable() // enable kelvin sensing.
bEnabled = stem.rail[0].getKelvinSensingEnable()
print bEnabled.value
bEnabled = stem.rail[0].getKelvinSensingState()
print bEnabled.value
stem.rail[0].setOperationalMode(0, auto);
mode = stem.rail[0].getOperationalMode(0);
print mode.value
state = stem.rail[0].getOperationalState(0);
print state.value
```

3.1.15 RCServo Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The RCServo entity provides a pulsed signal based on the RC servo standard. This consist of a period lasting 20ms with a high pulse between 1-2ms. The time high corresponds to a specific position determined by the servo being used. For example if you are using a 90 degree servo a 1.5ms pulse will correspond to the 45 degrees. 1ms and 2ms pulses will correspond to 0 and 90 degree positions respectively.

The RCServo entity is an overload to the [Digital Entity](#) and therefor requires proper configuration of the Digital entity before the RCServo entity can be enabled.

Note: Not all BrainStem modules will have this capability.

Set/Get Enable

```
servo [ index ] . getEnable <= (unsigned char) enable
servo [ index ] . setEnable => (unsigned char) enable
```

This functions gets/sets the RCServo function for a given pin (pending, it has been properly configured in the digital entity). At a firmware level this enables/disables the timers.

Set/Get Position

```
servo [ index ] . getPosition <= (unsigned char) position
servo [ index ] . setPosition => (unsigned char) position
```

This functions gets/sets the RCServo position. For outputs this will return the currently set position; however, for inputs it will return the value seen at the pin pending the pulse is valid. If the pulse or period are invalid a zero will be returned along with the error code aErrRange.

The default range is: 64 (1ms) - 192 (2ms). For example when working with a 90 degree servo setting the position to 64 will give you 0 degrees and 192 will give you 90 degrees.

Note: getPosition() will return the original setPosition() regardless of the reverse settings.

Set/Get Reverse

```
servo [ index ] . getReverse <= (unsigned char) reverse
servo [ index ] . setReverse => (unsigned char) reverse
```

This functions gets/sets the reverse (invert) option in the RCServo Class.

Given a setPosition of 64 the servo pulse will be 1ms; however, if you reverse it the value will now be treated as 192.

Aligning the Digital and RCServo Entities

Digital Entity	Servo Entity	Pin Number	Assignment
digital[0]	servo[0]	Pin 0	RCServo Input
digital[1]	servo[1]	Pin 1	RCServo Input
digital[2]	servo[2]	Pin 2	RCServo Input
digital[3]	servo[3]	Pin 3	RCServo Input
digital[4]	servo[4]	Pin 4	RCServo Output
digital[5]	servo[5]	Pin 5	RCServo Output
digital[6]	servo[6]	Pin 6	RCServo Output
digital[7]	servo[7]	Pin 7	RCServo Output

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

//Output
//Set digital pin 8 as an RCServo output.
err = stem.digital[8].setConfiguration(digitalConfigurationRCServoOutput);
//Enable the servo channel
```

(continues on next page)

(continued from previous page)

```

err = stem.servo[4].setEnabled(1);
//Set servo to middle/neutral position
err = stem.servo[4].setPosition(128);

//Input
//Set digital pin 0 as an RCServo input.
err = stem.digital[0].setConfiguration(digitalConfigurationRCServoInput);
//Enable the servo channel
err = stem.servo[0].setEnabled(1);
//Set servo to middle/neutral position
err = stem.servo[4].getPosition(&pPosition);

```

Python

```

# All commands return aErr values when errors are encountered and aErrNone on
# success.

#Output
#Set digital pin 8 as an RCServo output.
err = stem.digital[8].setConfiguration(CONFIGURATION_RCSERVO_OUTPUT)
#Enable the servo channel
err = stem.servo[4].setEnabled(1)
#Set servo to middle/neutral position
err = stem.servo[4].setPosition(128)

#Input
#Set digital pin 0 as an RCServo input.
err = stem.digital[0].setConfiguration(CONFIGURATION_RCSERVO_INPUT)
#Enable the servo channel
err = stem.servo[0].setEnabled(1)
#Set servo to middle/neutral position
err = stem.servo[0].getPosition(&pPosition)

```

3.1.16 Relay Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Relay entity is a simple class which allows the enabling and disabling of a specified relay.

Channel Enable (Get/Set)

```

relay [ index ] . setEnable => (unsigned char) enable
relay [ index ] . getEnable <= (unsigned char) enable

```

Enables the relay channel for the specified index

Get Voltage (Get)

```
relay [ index ] . getVoltage <= (unsigned char) voltage
```

Returns the voltage of the specified index.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get calls will fill the variable with the returned value.

err = stem.relay[0].setEnabled(1);
err = stem.relay[1].setEnabled(1);

err = stem.relay[0].getEnable(&enable);
err = stem.relay[1].getEnable(&enable);

err = stem.relay[0].getVoltage(&voltage);
err = stem.relay[1].getVoltage(&voltage);

err = stem.relay[0].setEnabled(0);
err = stem.relay[1].setEnabled(0);
```

Python

```
err = stem.relay[0].setEnabled(1);
err = stem.relay[1].setEnabled(1);

result = stem.relay[0].getEnable()
print result.value

result = stem.relay[1].getEnable()
print result.value

voltage = stem.relay[0].getVoltage();
print voltage.value

voltage = stem.relay[1].getVoltage();
print voltage.value

err = stem.relay[0].setEnabled(0);
err = stem.relay[1].setEnabled(0);
```

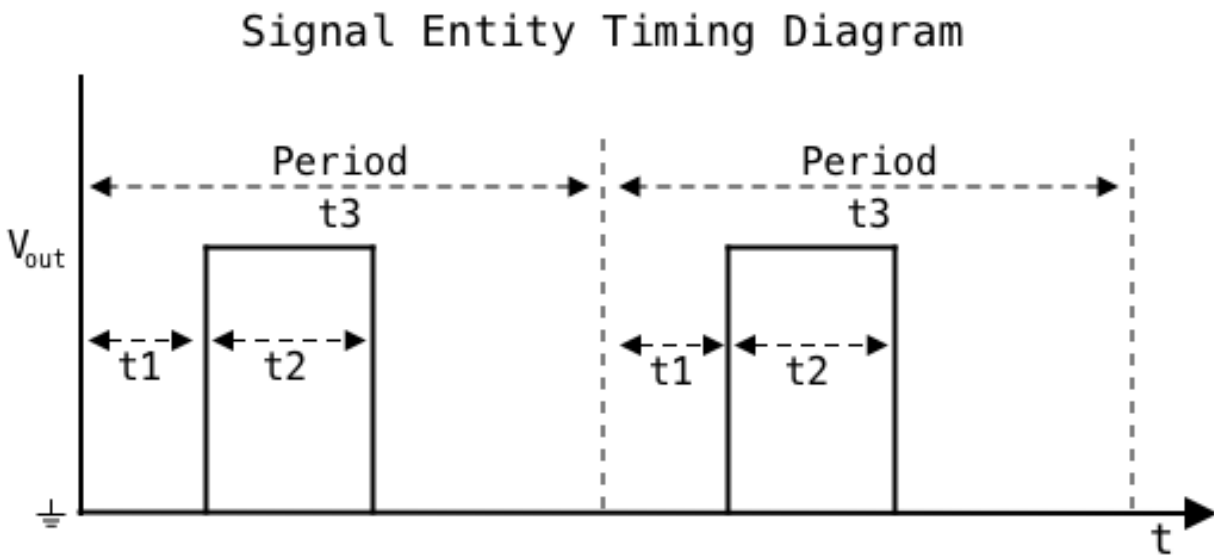
3.1.17 Signal Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

SignalClass. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Signal entity to Digital entity mapping varies from device to device. Please refer to the datasheet.

Timing



Set/Get enable

```
signal [ index ] . getEnable <= (unsigned char) enable  
signal [ index ] . setEnable => (unsigned char) enable
```

Enables the Signal Entity for a given index.

Set/Get T3 Time

```
signal [ index ] . getT3Time <= (unsigned int) t3_nsec  
signal [ index ] . setT3Time => (unsigned int) t3_nsec
```

The T3 time defines the period of the waveform in nano seconds.

Set/Get T2 Time

```
signal [ index ] . getT2Time <= (unsigned int) t2_nsec
signal [ index ] . setT2Time => (unsigned int) t2_nsec
```

The T2 time defines the high period of the waveform in nano seconds.

Set/Get invert

```
signal [ index ] . getInvert <= (unsigned char) invert
signal [ index ] . setInvert => (unsigned char) invert
```

Inverts the meaning of the T2 time. When inverted the T2 time will represent the time in nano seconds that the waveform is low.

Code Examples

C++

```
//Setup 10Hz Signal Output with 50% Duty Cycle
err = stem.digital[0].setConfiguration(digitalConfigurationSignalOutput);
err = stem.signal[0].setT2Time(50000000);
err = stem.signal[0].setT3Time(100000000);
err = stem.signal[0].setEnabled(1);

//Setup Signal as input and calculate the duty cycle.
err = stem.digital[4].setConfiguration(digitalConfigurationSignalInput);
err = stem.signal[4].getT2Time(&t2Time);
err = stem.signal[4].getT3Time(&t3Time);
double dutyCycle = ((double)t2Time / t3Time) * 100;
```

Python

```
#Setup 10Hz Signal Output with 50% Duty Cycle
err = stem.digital[0].setConfiguration(digitalConfigurationSignalOutput);
err = stem.signal[0].setT2Time(50000000);
err = stem.signal[0].setT3Time(100000000);
err = stem.signal[0].setEnabled(1);

#Setup Signal as input and calculate the duty cycle.
err = stem.digital[4].setConfiguration(digitalConfigurationSignalInput);
t2Time = stem.signal[4].getT2Time();
t3Time = stem.signal[4].getT3Time();
dutyCycle = (t2Time.value / t3Time.value) * 100;
```

3.1.18 Store Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

Get Slot State (Get)

```
store [ index ] . getSlotState <= (unsigned char) state
```

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

Load Slot (Write)

```
store [ index ] . loadSlot => (slot, byte buffer, buffer length)
```

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

Unload Slot (Read)

```
store [ index ] . unloadSlot <= (slot, byte buffer, max buffer size, length read)
```

This command reads the slot in the given store into the byte buffer. The length

will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

Slot Enable (Set)

```
store [ index ] . slotEnable => (unsigned char) slot
```

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

Slot Disable (Set)

```
store [ index ] . slotDisable => (unsigned char) slot
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

Slot Capacity (Get)

```
store [ index ] . slotCapacity (unsigned char) slot <= (unsigned short) capacity
```

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

Slot Size (Get)

```
store [ index ] . slotSize (unsigned char) slot <= (unsigned short) size
```

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

stem.store[0].getSlotState(3, state); // gets the state of slot 3 in the internal
↳store.
stem.store[0].loadSlot(3, buffer, length); // loads the data in buffer.
stem.store[1].unloadSlot(0, buffer, 300, length); // unloads at most 300 bytes from
↳the 1st RAM slot.
stem.store[0].enableSlot(1);
stem.store[0].disableSlot(1);
stem.store[0].getSlotCapacity(1, size); // gets the max size of the slot.
Stem.store[0].getSlotSize(1, size); // gets the current size of the data in the slot.
```

Python

```
res = stem.store[0].getSlotState(3) #res.value is the state of slot 3 in the internal
↳store
stem.store[0].loadSlot(3, buffer, length) # loads length bytes from buffer to slot 3
res = stem.store[1].unloadSlot(0) # res.value is a tuple of (str|bytes|int) of the
↳data in slot 0 and the length
stem.store[0].enableSlot(3)
stem.store[0].disableSlot(3)
res = stem.store[0].getCapacity(1) #res.value is the max size of the slot
res = stem.store[0].getSize(1) #res.value is the current size of the data in slot 1
```

3.1.19 System Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Every BrainStem module includes a single system entity. The system entity allows the retrieval and manipulation of configuration settings like the module address and input voltage, control over the user LED, as well as other functionality.

System save

```
system . save => (void)
```

BrainStem configuration settings are stored in volatile memory until the save command is executed. Settings such as the BootSlot, and changes to the Module or Router address will not persist across resets unless followed by a call to:

System reset (Set)

```
system . reset => (void)
```

Calling `system.reset()` will reset the BrainStem module just as if the reset button were pressed.

User LED

```
system . setLED => (unsigned char) state  
system . getLED <= (unsigned char) state
```

Gets or Sets the state of the User LED. Setting LED with a value of 1 turns the User LED on and setting it to 0 turns it off.

LED Brightness (Get/Set)

```
system . setLEDMaxBrightness => (unsigned char) value  
system . getLEDMaxBrightness <= (unsigned char) value
```

Gets or Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels.

Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Boot Slot (Get/Set)

```
system . setBootSlot => (unsigned char) slot
system . getBootSlot <= (unsigned char) slot
```

BrainStem modules can be configured to enable a reflex file at boot. The reflex file must be loaded into a slot in the internal store. Setting the boot slot to the value 255 will disable on boot functionality. For more information about stores and slots please see the store section of the reference manual. For more information about reflexes please see the Reflex section of the manual.

Input Voltage (Get)

```
system . getInputVoltage <= (unsigned int) inputVoltage
```

The input voltage system command is a read only command and will return the input supply voltage of the BrainStem module in micro volts.

Serial Number (Get)

```
system . getSerialNumber <= (unsigned int) serialNumber
```

Read only command that returns the unique module serial number. The returned value is an unsigned int. In Acroname UI applications the serial number is generally represented as an 8 character Hexadecimal number.

BrainStem Model (Get)

```
system . getModel <= (unsigned char) BrainStem model.
```

Read only command that returns the model of the BrainStem module.

Hardware Version (Get)

```
system . getHardwareVersion <= (unsigned int) Hardware Version.
```

Read only command that returns the hardware version of the module. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

Version (Get)

```
system . getVersion <= (unsigned int) version number.
```

Read only command that returns the version number of the BrainStem firmware. This is a packed format. The `aVersion.h` C API can represent this version in a human readable manner. The format of the version number is 3 digits separated by ..

```
major . minor . patch
```

Module Address (Get)

```
system . getModule <= (unsigned char) module
```

The module address is the number used to address the module on the BrainStem network and from the host. This is a combination of the module base address, any software offset that is applied and any hardware module offset.

Module Base Address (Get)

```
system . getModuleBaseAddress <= (unsigned char) module
```

The module base address is the default or base address of the module, before any offsets are applied.

Module Software Offset (Set/Get)

```
system . getModuleSoftwareOffset <= (unsigned char) software offset  
system . setModuleSoftwareOffset => (unsigned char) software offset
```

The module software offset is added to the module's base address and any hardware offsets to determine the final module address of the module. This setting is not applied until saved and the module has been reset.

Module Hardware Offset (Get)

```
system . getModuleHardwareOffset <= (unsigned char) module hardware offset
```

MTM BrainStems have a set of module offset pins which will adjust the module address via hardware. See the data sheet for your MTM module for more information about these hardware settings. The module offset command is a read only command that returns the offset that will be added to the base module address and any software offset to determine the operating address of the MTM BrainStem module. Changes to the hardware offset are applied when the Device is reset.

Router Address (Get/Set)

```
system . setRouter => (unsigned char) module  
system . getRouter <= (unsigned char) module
```

The BrainStem router address refers to the BrainStem module address of the module that will coordinate communication with the host system. This setting is not applied until it is saved and the module has been reset.

Changing the router address can have negative consequences for communicating with the BrainStem network. Please see the appendix on the BrainStem Network setup for more information.

- [Appendix: Brainstem Universal Entity Interface](#)
- [Appendix: The BrainStem Communication Protocol](#)

HeartBeat Interval (Get/Set)

```
system . setHBInterval => (unsigned char) interval
system . getHBInterval <= (unsigned char) interval
```

Gets or sets the heartbeat interval to control the amount of heartbeat traffic. This value is set at approximately 1/50th of a second resolution. Heartbeat packets are handled by the underlying system, and are indicated on the brainstem by the blinking green heartbeat LED. UI applications also have Heartbeat indicators. Default value is 12.

System Name (Get/Set)

```
system . getName <= (unsigned char[]) name
system . setName => (unsigned char[]) name
```

Allows for setting a friendly name to the device with a 32 character limit.

System Build (Get)

```
system . getBuild <= (unsigned int) build
```

Returns the build number of the current firmware. This is a unique identifier for the specific firmware build.

System Errors (Get)

```
system . getErrors <= (unsigned int) errors
```

Returns the current system error status. The error value is a bitfield indicating various system-level error conditions.

Input Current (Get)

```
system . getInputCurrent <= (unsigned int) milliamps
```

Returns the current input current to the system in milliamps. This represents the current being drawn from the power source.

Input Power Behavior (Get/Set)

```
system . getInputPowerBehavior <= (unsigned char) behavior
system . setInputPowerBehavior => (unsigned char) behavior
```

Returns or sets the input power behavior configuration. This controls how the system responds to different power input conditions.

Input Power Behavior Config (Get/Set)

```
system . getInputPowerBehaviorConfig <= (unsigned int[]) config  
system . setInputPowerBehaviorConfig => (unsigned int[]) config
```

Returns or sets the detailed input power behavior configuration. This provides fine-grained control over power input behavior.

Input Power Source (Get)

```
system . getInputPowerSource <= (unsigned char) source
```

Returns the current input power source. Indicates which power source is currently being used to power the system.

Link Interface (Get/Set)

```
system . getLinkInterface <= (unsigned char) interface  
system . setLinkInterface => (unsigned char) interface
```

Returns or sets the link interface configuration. This controls how the system communicates with external devices.

Maximum Temperature (Get)

```
system . getMaximumTemperature <= (unsigned int) microcelsius
```

Returns the maximum temperature recorded by the system in microcelsius. This represents the highest temperature the system has reached.

Minimum Temperature (Get)

```
system . getMinimumTemperature <= (unsigned int) microcelsius
```

Returns the minimum temperature recorded by the system in microcelsius. This represents the lowest temperature the system has reached.

System Power Limit (Get)

```
system . getPowerLimit <= (unsigned int) milliwatts
```

Returns the current system power limit in milliwatts. This represents the maximum power the system is configured to consume.

System Power Limit Max (Get/Set)

```
system . getPowerLimitMax <= (unsigned int) milliwatts  
system . setPowerLimitMax => (unsigned int) milliwatts
```

Returns or sets the maximum system power limit in milliwatts. This sets the upper bound for system power consumption.

System Power Limit State (Get)

```
system . getPowerLimitState <= (unsigned int) state
```

Returns the current power limit state. This indicates whether the system is operating within, approaching, or exceeding power limits.

Protocol Features (Get)

```
system . getProtocolFeatures <= (unsigned int) features
```

Returns the protocol features supported by the system. This is a bitfield indicating which protocol features are available.

Router Address Setting (Get)

```
system . getRouterAddressSetting <= (unsigned char) address
```

Returns the router address setting. This indicates the configured router address for the system.

System Temperature (Get)

```
system . getTemperature <= (unsigned int) microcelsius
```

Returns the current system temperature in microcelsius. This represents the real-time temperature of the system.

Unregulated Current (Get)

```
system . getUnregulatedCurrent <= (unsigned int) milliamps
```

Returns the current unregulated current in milliamps. This represents the current being drawn from the unregulated power supply.

Unregulated Voltage (Get)

```
system . getUnregulatedVoltage <= (unsigned int) millivolts
```

Returns the current unregulated voltage in millivolts. This represents the voltage of the unregulated power supply.

System Uptime (Get)

```
system . getUptime <= (unsigned int) seconds
```

Returns the system uptime in seconds. This represents how long the system has been running since the last reset.

Log Events (Get)

```
system . logEvents => (void)
```

Initiates logging of system events to a designated slot. This command triggers the system to save event logs to memory.

Reset Device to Factory Defaults (Set)

```
system . resetDeviceToFactoryDefaults => (void)
```

Resets the device to its factory default configuration. This command restores all settings to their original factory state.

Route to Me (Get/Set)

```
system . routeToMe <= (unsigned char) enabled  
system . routeToMe => (unsigned char) enabled
```

Returns or sets the “route to me” configuration. When enabled, this allows the system to receive routed packets.

Code Examples

C++

```
// Get requests fill the parameter with the current system value upon success.  
// All commands return aErr values when errors are encountered and aErrNone on  
// success.  
  
stem.system.save();  
stem.system.reset();  
stem.system.setLED(1);  
stem.system.getLED(state);  
stem.system.setLEDMaxBrightness(255);
```

(continues on next page)

(continued from previous page)

```

stem.system.getLEDMaxBrightness(value);
stem.system.setBootSlot(5);
stem.system.getBootSlot(slot);
stem.system.getInputVoltage(voltage);
stem.system.getModule(address);
stem.system.getRouter(address);
stem.system.setRouter(6);
stem.system.getModuleBaseAddress(address);
stem.system.setModuleSoftwareOffset(16);
stem.system.getModuleSoftwareOffset(offset);
stem.system.getModuleHardwareOffset(offset);
stem.system.getSerialNumber(serialNumber);
stem.system.getModel(model);
stem.system.getHardwareVersion(hardwareVersion);
stem.system.getVersion(version);
stem.system.getHBInterval(interval);
stem.system.setHBInterval(interval);

```

Python

```

stem.system.save()
stem.system.reset()
stem.system.setLED(1)
state = stem.system.getLED()
print state.value
stem.system.setLEDMaxBrightness(255);
brightness = stem.system.getLEDMaxBrightness();
print brightness.value
stem.system.setBootSlot(5)
slot = stem.system.getBootSlot()
print slot.value
inputVoltage = stem.system.getInputVoltage()
print inputVoltage.value
module = stem.system.getModule()
print module.value
address = stem.system.getModuleBaseAddress();
print address.value
stem.system.setModuleSoftwareOffset(16);
offset = stem.system.getModuleSoftwareOffset();
print offset.value
offset = stem.system.getModuleHardwareOffset();
print offset.value
serialNumber = stem.system.getSerialNumber()
print serialNumber.value
model = stem.system.getModel()
hardwareVersion = stem.system.getHardwareVersion()
print hardwareVersion.value
version = stem.system.getVersion()
print brainstem.version.get_version_string(version.value)
hbInterval = stem.system.getHBInterval()
print hbInterval.value
stem.system.setHBInterval(12)

```

3.1.20 Temperature Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

Certain modules have a temperature measurement available. The temperature entity gives access to these measurements. Check your module datasheet to see if your module has a temperature entity.

Temperature (Get)

```
temperature [ index ] . getTemperature => (int) microcelsius
```

Returns a temperature measurement in microcelsius.

Temperature Value (Get)

```
temperature [ index ] . getValue <= (int) microcelsius
```

Returns the current temperature measurement in microcelsius. This is the same as `getTemperature` but follows the standard naming convention.

Temperature Maximum (Get)

```
temperature [ index ] . getValueMax <= (int) microcelsius
```

Returns the maximum temperature recorded since the last power cycle in microcelsius. This tracks the highest temperature the module has reached.

Temperature Minimum (Get)

```
temperature [ index ] . getValueMin <= (int) microcelsius
```

Returns the minimum temperature recorded since the last power cycle in microcelsius. This tracks the lowest temperature the module has reached.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success. Get commands fill the variable with the returned value.  
  
stem.temperature[0].getTemperature(microcelsius);  
stem.temperature[0].getValue(currentTemp);  
stem.temperature[0].getValueMax(maxTemp);  
stem.temperature[0].getValueMin(minTemp);
```


Python

```
microcelsius = stem.temperature[0].getTemperature();
print microcelsius.value
currentTemp = stem.temperature[0].getValue();
print currentTemp.value
maxTemp = stem.temperature[0].getValueMax();
print maxTemp.value
minTemp = stem.temperature[0].getValueMin();
print minTemp.value
```

3.1.21 Timer Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The Timer entity provides simple scheduling for events in the reflex system. BrainStem modules generally contain between 4 and 8 timers depending on the module. The most common usage is to write a timer reflex and load and enable it on the BrainStem module, then an expiration can be set for the timer, and this reflex code will be executed when the timer expires.

Timers have two modes, single which executes just once and repeat which executes until the expiration is set to zero or the mode is changed to single.

Expiration (Get/Set)

```
timer [ index ] . getExpiration <= (unsigned int) microseconds
timer [ index ] . setExpiration => (unsigned int) microseconds
```

Gets or sets the next expiration for this timer in microseconds. If zero, the timer is not currently set to expire in the future.

Mode (Get/Set)

```
timer [ index ] . getMode <= (unsigned char) mode
timer [ index ] . setMode => (unsigned char) mode
```

Gets or sets the current timer mode. 1 for repeat mode and 0 for single mode.

When in repeat mode an expiration will occur every n microseconds when n is the expiration setting of the timer. To stop a repeat timer, set its expiration to 0.

When in single mode (The default) setting a non-zero expiration will cause the timer to trigger a single time after the expiration setting in microseconds. If a timer is set, resetting its expiration to zero will clear the timer, and no reflex code will be triggered.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get commands fill the variable with the returned value.

stem.timer[0].getExpiration(uSecs);
stem.timer[0].setExpiration(1000000); // Sets the timer for 1 second in the future.
stem.timer[0].getMode(mode);
stem.timer[0].setMode(timerModeRepeat) // timerModeRepeat is a convenience define.
```

Python

```
uSecs = stem.timer[3].getExpiration()
stem.timer[3].setExpiration(1000000) # Sets the timer for 1 second in the future.
mode = stem.timer[3].getMode()
stem.timer[3].setMode(timerModeRepeat) // timerModeRepeat is a convenience define.
```

3.1.22 UART Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The UART entity is a class which allows a user to configure and control an arbitrary stream of serial data. These streams can represent a number of different transports, including a RS232 external interface, a virtual COM port, and onboard UART interfaces from system components. This entity allows each transport to be configured as either an endpoint, or as a passthrough between transports, similar to a switchboard of a telephone operator.

Products may implement any number of UART entities for the specific capabilities of that product. Consult the product specific reference material for details about the exact capabilities of the product.

Channel Enable (Get/Set)

```
uart [ index ] . setEnable => (unsigned char) enable
uart [ index ] . getEnable <= (unsigned char) enable
```

Enables the uart channel for the specified index.

Change Linked Channel (Get/Set)

```
uart [ index ] . setLinkChannel => (unsigned char) index
uart [ index ] . getLinkChannel <= (unsigned char) index
```

Sets a UART channel to pass data to and from another UART channel. If the channel index is set to the current index, the UART channel will operate as an endpoint with a specific protocol.

Note: Channel linking must be set up on both ends of the connection. That is, setting Channel A's link channel to Channel B will not pass data through until Channel B's link channel is also set to channel A.

Change Protocol (Get/Set)

```
uart [ index ] . setProtocol => (unsigned char) protocol
uart [ index ] . getProtocol <= (unsigned char) protocol
```

Sets the protocol of the UART when operating as an endpoint. The protocols supported by the API are listed below. Note that not all products or indexes are required to support all protocols. The list of protocols that may be assigned can be queried using `getCapableProtocols` and `getAvailableProtocols`.

Enumera- tion	Name	Description
0	Undefined	Entity will drop incoming data and will not send data
1	Extron Responder	Entity will respond to incoming Extron-like commands (see product details)
2	BrainStem Transport	Connection can be opened to this entity from the BrainStem API
3	Extron Initiator	Entity will initiate Extron-like commands on certain events (see product details)
4	Reserved	Reserved for future use
5	Reserved	Reserved for future use
6	Loopback	Entity will write back any received data

```
uart [ index ] . getCapableProtocols => (unsigned int) protocols
```

Returns an integer containing a bitmask where each bit represents a protocol enumeration value that the channel may be able to be configured with. This does not guarantee that attempting to set the protocol will succeed, as there may not be sufficient resources to assign to this channel.

```
uart [ index ] . getAvailableProtocols => (unsigned int) protocols
```

Returns an integer containing a bitmask where each bit represents a protocol enumeration value that the channel both supports and has an available resource to assign for this protocol.

Table 14: UART Protocol Values

Value	Name	Description
0	Undefined	Disabled/Undefined protocol
1	Extron Responder	Extron Compatible Protocol (backward compatibility)
2	Brainstem Transport	Brainstem Transport protocol
6	Loopback	Loopback mode for testing

Note: Protocol availability may vary by device. Use `getCapableProtocols()` and `getAvailableProtocols()` to determine which protocols are supported on a specific device and channel.

Set Link Channel (Get/Set)

```
uart [ index ] . setLinkChannel => (unsigned char) channel
uart [ index ] . getLinkChannel <= (unsigned char) channel
```

Sets or gets the index of another UART Entity that should be linked to this UART. If set to the index of this entity, the channel will not be linked. If set to the index of another UART entity, data will be sent between the two UART entities with no additional processing.

Note: Linking VCOM to Hardware UART/RS232: A common use case is to link a VCOM (Virtual COM port) channel to the hardware UART/RS232 connector. This allows USB-based serial communication to be transparently forwarded to the physical RS232 interface. For example, linking UART[1] (VCOM) to UART[0] (hardware RS232) will forward all data received on the VCOM channel directly to the RS232 connector, and vice versa. This is useful for remote serial access or bridging USB serial connections to physical serial devices.

Set Stop Bits (Get/Set)

```
uart [ index ] . setStopBits => (unsigned char) stopBits
uart [ index ] . getStopBits <= (unsigned char) stopBits
```

Sets or gets the UART stop bit configuration.

Table 15: Stop Bits Values

Value	Description
0	1 Stop Bit
1	1.5 Stop Bits
2	2 Stop Bits

Set Parity (Get/Set)

```
uart [ index ] . setParity => (unsigned char) parity
uart [ index ] . getParity <= (unsigned char) parity
```

Sets or gets the UART parity configuration.

Table 16: Parity Values

Value	Description
0	No Parity Bit
1	Odd Parity Bit
2	Even Parity Bit
3	Mark Parity Bit
4	Space Parity Bit

Set Data Bits (Get/Set)

```
uart [ index ] . setDataBits => (unsigned char) dataBits
uart [ index ] . getDataBits <= (unsigned char) dataBits
```

Sets or gets the number of data bits per character for the UART channel.

Set Flow Control (Get/Set)

```
uart [ index ] . setFlowControl => (unsigned char) flowControl
uart [ index ] . getFlowControl <= (unsigned char) flowControl
```

Sets or gets the UART flow control configuration as a bitmask. Multiple flow control methods can be enabled simultaneously.

Table 17: Flow Control Bits

Bit	Description
0	RTS/CTS Enable
1	DSR/DTR Enable
2	XON/XOFF Enable

Get Capable Protocols

```
uart [ index ] . getCapableProtocols <= (unsigned int) protocols
```

Returns a bitmask indicating which protocols are capable of being used on this UART channel. The value of the protocol is mapped to the bit index (e.g., protocol 0 is bit 0, protocol 1 is bit 1, etc.).

Get Available Protocols

```
uart [ index ] . getAvailableProtocols <= (unsigned int) protocols
```

Returns a bitmask indicating which protocols are currently available for use on this UART channel. This takes into account resource availability (e.g., whether Brainstem Transport resources are available). The value of the protocol is mapped to the bit index (e.g., protocol 0 is bit 0, protocol 1 is bit 1, etc.).

Change Line Coding (Get/Set)

These APIs are all used to set the UART line coding parameters. Note that not all parameters may be changed for certain indexes, nor may all options for line coding be specified.

```
uart [ index ] . setBaudRate => (unsigned int) rate
uart [ index ] . getBaudRate <= (unsigned int) rate
```

Sets the baud rate of the transport in bits per second.

Note: A baudrate of 0 will perform automatic baudrate selection if supported on the specified index.

```
uart [ index ] . setParity => (unsigned char) parity
uart [ index ] . getParity <= (unsigned char) parity
```

Sets the parity bit configuration:

Enumeration	Name	Description
0	uartParity_None_Value	Parity bit is not used
1	uartParity_Odd_Value	Parity bit is set to 1 if the number of 1s in the data bits is odd
2	uartParity_Even_Value	Parity bit is set to 1 if the number of 1s in the data bits is even
3	uartParity_Mark_Value	Forces parity bit to a binary 1
4	uartParity_Space_Value	Forces parity bit to a binary 0

```
uart [ index ] . setStopBits => (unsigned char) stopBits
uart [ index ] . getStopBits <= (unsigned char) stopBits
```

Sets the stop bit configuration:

Enumeration	Name	Number of Stop Bits
0	uartStopBits_1_Value	1
1	uartStopBits_1p5_Value	1.5
2	uartStopBits_2_Value	2

```
uart [ index ] . setDataBits => (unsigned char) dataBits
uart [ index ] . getDataBits <= (unsigned char) dataBits
```

Sets the number of bits in a single payload (e.g. 7, 8, 9)

```
uart [ index ] . setFlowControl => (unsigned char) flowControl
uart [ index ] . getFlowControl <= (unsigned char) flowControl
```

Sets the flow control configuration:

Bit	Name	Description
0	uartFlowControl_RTS_CTS_Bit	Enable RTS/CTS Flow Control
1	uartFlowControl_DSR_DTR_Bit	Enable DSR/DTR Flow Control
2	uartFlowControl_XON_XOFF_Bit	Enable XON/XOFF Flow Control

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get calls will fill the variable with the returned value.

err = stem.uart[0].setEnabled(1);
err = stem.uart[1].setEnabled(1);

err = stem.uart[0].getEnable(&enable);
err = stem.uart[1].getEnable(&enable);

err = stem.uart[0].setBaudRate(115200);
err = stem.uart[0].setProtocol(1); // Extron Compatible Protocol

err = stem.uart[0].setEnabled(0);
err = stem.uart[1].setEnabled(0);

// Loopback Protocol Example
// Configure UART[0] for loopback testing
err = stem.uart[0].setEnabled(1);
err = stem.uart[0].setBaudRate(115200);
err = stem.uart[0].setProtocol(6); // Loopback protocol
// Data sent to UART[0] will be echoed back

// VCOM to Hardware UART Link Example
// Link VCOM channel (UART[1]) to hardware RS232 (UART[0])
err = stem.uart[0].setEnabled(1); // Enable hardware UART
err = stem.uart[1].setEnabled(1); // Enable VCOM channel
err = stem.uart[1].setLinkChannel(0); // Link VCOM to hardware UART
err = stem.uart[0].setLinkChannel(1); // Link hardware UART to VCOM
// Data received on VCOM will be forwarded to RS232, and vice versa
```

Python

```
err = stem.uart[0].setEnabled(1)
err = stem.uart[1].setEnabled(1)

result = stem.uart[0].getEnable()
print result.value

result = stem.uart[1].getEnable()
print result.value

err = stem.uart[0].setBaudRate(115200)
err = stem.uart[0].setProtocol(1) # Extron Compatible Protocol

err = stem.uart[0].setEnabled(0)
err = stem.uart[1].setEnabled(0)

# Loopback Protocol Example
# Configure UART[0] for loopback testing
err = stem.uart[0].setEnabled(1)
```

(continues on next page)

(continued from previous page)

```

err = stem.uart[0].setBaudRate(115200)
err = stem.uart[0].setProtocol(6)  # Loopback protocol
# Data sent to UART[0] will be echoed back

# VCOM to Hardware UART Link Example
# Link VCOM channel (UART[1]) to hardware RS232 (UART[0])
err = stem.uart[0].setEnabled(1)  # Enable hardware UART
err = stem.uart[1].setEnabled(1)  # Enable VCOM channel
err = stem.uart[1].setLinkChannel(0)  # Link VCOM to hardware UART
# Data received on VCOM will be forwarded to RS232, and vice versa

```

3.1.23 USB Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USB Entity provides the software control interface for USB related features. This entity is supported by BrainStem products which have programmatically controlled USB features.

Port Enable/Disable (Set)

```

usb . setPortEnable => (unsigned char) channel
usb . setPortDisable => (unsigned char) channel

```

Enables or Disables the given downstream channel. This call enables or disables data and power together for the given channel.

Data Enable/Disable (Set)

```

usb . setDataEnable => (unsigned char) channel
usb . setDataDisable => (unsigned char) channel

```

Enables or Disables data only for given downstream channel. This call enables or disables the usb data (+) and data (-) lines for the given channel.

Calls to this command have no side effects on the power connections for the channel. If power was enabled before the call then it will still be enabled after the call to setDataEnable/Disable.

High Speed Data Enable/Disable (Set)

```

usb . setHiSpeedDataEnable => (unsigned char) channel
usb . setHiSpeedDataDisable => (unsigned char) channel

```

Enables or Disables Hi Speed data only for given downstream channel. This call enables or disables the usb data (+) and data (-) lines for the given channel.

Calls to this command have no side effects on the power connections for the channel. If power was enabled before the call then it will still be enabled after the call to setHiSpeedDataEnable/Disable.

Super Speed Data Enable/Disable (Set)

```
usb . setSuperSpeedDataEnable => (unsigned char) channel
usb . setSuperSpeedDataDisable => (unsigned char) channel
```

Enables or Disables Super Speed (3.0) data only for given downstream channel. This call enables or disables the usb data (+) and data (-) lines for the given channel.

Calls to this command have no side effects on the power connections for the channel. If power was enabled before the call then it will still be enabled after the call to setSuperSpeedDataEnable/Disable.

Power Enable/Disable (Set)

```
usb . setPowerEnable => (unsigned char) channel
usb . setPowerDisable => (unsigned char) channel
```

Enables or Disables power only for given downstream channel. This call enables or disables the usb power connection for the given channel.

Calls to this command have no side effects on the data connections for the channel. If data was enabled before the call then it will still be enabled after the call to setPowerEnable/Disable.

port Voltage/Current (Get)

```
usb . getPortVoltage (unsigned char) channel <= (unsigned int) microvolts
usb . getPortCurrent (unsigned char) channel <= (unsigned int) microamps
```

Returns the last read values for Voltage (in microvolts) and Current (in microamps) for the given channel.

Hub Mode (Get/Set)

```
usb . getHubMode <= (unsigned int) state
usb . setHubMode => (unsigned int) state
```

Gets/Sets the hubs mode in the form of a big mapped representation. See the product datasheet for state mapping. Usually represents the downstream ports power and data lines enable/disable state.

Hub State (Get)

Note: This function has been removed in version 2.5. This functionality is moved to [Port State](#).

Hub Error Status (Get)

Note: This function has been removed in version 2.5. This functionality is moved to [Port Error](#).

Clear Port Error Status (Set)

```
usb . clearPortErrorStatus => (unsigned char) channel
```

Clears the error status for the given channel

Upstream Mode (Get/Set)

```
usb . getUpstreamMode <= (unsigned char) mode
usb . setUpstreamMode => (unsigned char) mode
```

Gets/Sets the mode of the upstream USB ports. The mode parameter can be defined as the following:

Value	Definitions	Hub Upstream Mode Descriptions
0	usbUpstreamModePort0	Force upstream port 0 to be selected
1	usbUpstreamModePort1	Force upstream port 1 to be selected
2	usbUpstreamModeAuto	Automatically detect upstream port
255	usbUpstreamModeNone	Disconnect both upstream ports

Upstream State (Get)

```
usb . getUpstreamState <= (unsigned char) state
```

Gets the upstream switch state for the USB upstream ports. Returns none if no ports are plugged in, port 0 if the mode is set correctly and a cable is plugged into port 0, and port 1 if the mode is set correctly and a cable is plugged into port 1

Enumeration Delay (Get/Set)

```
usb . getEnumerationDelay <= (unsigned int) ms_delay
usb . setEnumerationDelay => (unsigned int) ms_delay
```

Gets/Sets the inter-port enumeration delay in milliseconds. The enumeration delay sequentially enables data and power to downstream ports after the defined delay time. After setting and saving this parameter all downstream ports will be initially disabled upon system power-on or reset. Similarly, if there is no upstream connection, all downstream ports will be disabled. When an upstream connection is applied, or after the system boots, the system will wait for the defined delay time and enable the lowest port number. The system will then wait for the defined delay time and then enable the next highest port. This behavior repeats until all ports are enabled.

Inconsistent behavior from race conditions may occur if enumeration delay is used in conjunction with Reflex programs which also manipulate the downstream port states. Care should be taken to ensure no conflicts between the enumeration delay and Reflex programs.

Note: This setting should be saved with a `stem.system.save()` call.

Upstream Boost Mode (Get/Set)

```
usb . getUpstreamBoostMode <= (unsigned char) setting
usb . setUpstreamBoostMode => (unsigned char) setting
```

Gets/Sets the upstream boost mode. Boost mode increase the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Modes: 0 = no boost, 1 = 4% boost, 2 = 8% boost, 3 = 12% boost.

Note: This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Down Stream Boost Mode (Get/Set)

```
usb . getDownstreamBoostMode <= (unsigned char) setting
usb . setDownstreamBoostMode => (unsigned char) setting
```

Gets/Sets the Downstream boost mode. Boost mode increase the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Modes: 0 = no boost, 1 = 4% boost, 2 = 8% boost, 3 = 12% boost.

Note: This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Port Current Limit (Get/Set)

```
usb . setPortCurrentLimit => (unsigned char) channel, (unsigned int) microamps
usb . getPortCurrentLimit (unsigned char) channel <= (unsigned int) microamps
```

Gets/Sets the current limit for the downstream channel. There are a number of settings for current limits ranging from 100 mAmps to 2.5 amps. See the USB hub datasheet for specific settings information.

Port Mode setting (Get/Set)

```
usb . setPortMode => (unsigned char) channel, (unsigned char) mode  
usb . getPortMode (unsigned char) channel <= (unsigned char) mode
```

Gets/Sets the Port mode for the channel specified. The portmode is a bitmapped setting. Device specific mode options are listed in the data-sheet. There is a unified listing of all port mode bits at *usbPortMode* within *usb_entity*.

Port State (Get)

```
usb . getPortState (unsigned char) channel <= (unsigned char) mode
```

Gets the Port state for the channel specified. State options for the device are listed in the device data-sheet.

Port Error (Get)

```
usb . getPortError (unsigned char) channel <= (unsigned char) mode
```

Gets the Port error status for the channel specified. Error status for the device are listed in the device data-sheet.

System Temperature (Get)

Note: This function has been removed in version 2.5. This functionality is moved to [temperature](#).

Connect Mode setting (Get/Set)

```
usb . setConnectMode => (unsigned char) channel, (unsigned char) mode  
usb . getConnectMode (unsigned char) channel <= (unsigned char) mode
```

Gets/Sets the connect mode for the channel specified. Check the device datasheet for more information regarding the use of this function.

CC1/CC2 Enable/Disable setting (Get/Set)

```
usb . setCC[1|2]Enable => (unsigned char) channel, (unsigned char) bEnable  
usb . getCC[1|2]Enable (unsigned char) channel <= (unsigned char) bEnable
```

Gets or sets the enabled status of the CC1/CC2 lines.

CC1/CC2 Current (Get)

```
usb . getCC[1|2]Current (unsigned char) channel <= (unsigned char) microAmps
```

Gets the current on the CC1/CC2 line in microAmps.

CC1/CC2 Voltage (Get)

```
usb . getCC[1|2]Voltage (unsigned char) channel <= (unsigned char) microVolts
```

Gets the voltage on the CC1/CC2 lines in microVolts.

SBU Enable/Disable setting (Get/Set)

```
usb . setSBUEnable => (unsigned char) channel, (unsigned char) bEnable
usb . getSBUEnable (unsigned char) channel <= (unsigned char) bEnable
```

Gets or sets the enabled status of the SBU lines.

Cable Flip (Get/Set)

```
usb . getCableFlip (unsigned char) channel <= (unsigned char) flipped
usb . setCableFlip => (unsigned char) channel, (unsigned char) flipped
```

Gets or sets the cable flip state for the channel specified. This indicates whether the USB-C cable is flipped and affects signal routing. Change the orientation of the common side to Mux side cable connection.

Alt Mode Configuration (Get/Set)

```
usb . getAltModeConfig (unsigned char) channel <= (unsigned int) configuration
usb . setAltModeConfig => (unsigned char) channel, (unsigned int) configuration
```

Gets or sets the alternate mode configuration for the channel specified. This controls which alternate mode is active on the port.

Downstream Data Speed (Get)

```
usb . getDownstreamDataSpeed (unsigned char) channel <= (unsigned char) speed
```

Returns the data speed of the downstream device for the channel specified. This indicates the USB speed at which the connected device is operating.

Table 18: Downstream Data Speed Options

Value	Name	Description
0	Unknown	Unknown or not detected speed
1	Hi-Speed	USB 2.0 Hi-Speed (480 Mbps)
2	SuperSpeed	USB 3.0 SuperSpeed (5 Gbps)
3	Low-Speed	USB 1.1 Low-Speed (1.5 Mbps)

SBU1/SBU2 Voltage (Get)

```
usb . getSBU[1|2]Voltage (unsigned char) channel <= (unsigned int) microvolts
```

Returns the voltage on the SBU[1|2] line in microvolts for the channel specified. SBU (Sideband Use) lines are used for alternate modes in USB-C.

SBU Enable (Get/Set)

```
usb . getSBUEnable (unsigned char) channel <= (unsigned char) enabled
usb . setSBUEnable => (unsigned char) channel, (unsigned char) enabled
```

Gets or sets the enabled status of the SBU lines for the channel specified. When enabled, the SBU lines are active for alternate mode communication.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get commands fill the variable with the returned value.

stem.usb.setPortEnable(1);
stem.usb.setPortDisable(2);
stem.usb.setDataEnable(0);
...
```

Python

```
stem.usb.setPortEnable(1)
stem.usb.setPortDisable(2)
stem.usb.setDataEnable(0)
stem.usb.setDataDisable(1)
stem.usb.setPowerEnable(0)
stem.usb.setPowerDisable(0)
microamps = stem.usb.getPortCurrent(0)
print microamps.value
microvolts = stem.usb.getPortVoltage(0)
print microvolts.value
```

(continues on next page)

(continued from previous page)

```
stem.usb.setPortCurrentLimit(0, limit_setting)
state = stem.usb.getHubState()
print state.value
...
```

3.1.24 USB System Entity

API Documentation: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[CCA\]](#) [\[REST\]](#)

The USBSystem class provides high level control of the lower level *Port Entity*

Upstream Connection (Get/Set)

```
usbssystem . setUpstream => (unsigned char) enable
usbssystem . getUpstream <= (unsigned char) enable
```

Many acroname products have multiple upstream port selections. This function is used to access and control that functionality.

Upstream Connection HighSpeed (Get/Set)

```
usbssystem . setUpstreamHS => (unsigned char) enable
usbssystem . getUpstreamHS <= (unsigned char) enable
```

Many acroname products have multiple upstream port selections, some even have the ability to move just the HighSpeed or SuperSpeed signals. This function is used to access and control that functionality for the HighSpeed signals only.

Upstream Connection SuperSpeed (Get/Set)

```
usbssystem . setUpstreamSS => (unsigned char) enable
usbssystem . getUpstreamSS <= (unsigned char) enable
```

Many acroname products have multiple upstream port selections, some even have the ability to move just the HighSpeed or SuperSpeed signals. This function is used to access and control that functionality for the SuperSpeed signals only.

Enumeration Delay (Get/Set)

```
usbssystem . getEnumerationDelay <= (unsigned int) ms_delay
usbssystem . setEnumerationDelay => (unsigned int) ms_delay
```

Gets/Sets the inter-port enumeration delay in milliseconds. The enumeration delay sequentially enables data and power to downstream ports after the defined delay time. After setting and saving this parameter all downstream ports will be initially disabled upon system power-on or reset. Similarly, if there is no upstream connection, all downstream ports will be disabled. When an upstream connection is applied, or after the system boots, the system will wait for the defined delay time and enable the lowest port number. The system will then wait for the defined delay time and then enable the next highest port. This behavior repeats until all ports are enabled.

Inconsistent behavior from race conditions may occur if enumeration delay is used in conjunction with Reflex programs which also manipulate the downstream port states. Care should be taken to ensure no conflicts between the enumeration delay and Reflex programs.

Enabled List (Get/Set)

```
usbsystem . getEnabledList <= (unsigned int) list  
usbsystem . setEnabledList => (unsigned int) list
```

The enabled list function provides state and control over all lower ports enables. It is equivalent to calling get/set enabled from the [PortClass](#) on all ports at once. The returned variable is in a bit mapped format. Please see the product data sheet for specific bit meanings.

Mode List (Get/Set)

```
usbsystem . getModeList <= (unsigned int [NUM_PORTS]) list  
usbsystem . setModeList => (unsigned int [NUM_PORTS]) list
```

The mode list function gives you access and control to all lower level port modes. It is equivalent to calling get/set mode from the [PortClass](#) on all ports at once.

State List (Get)

```
usbsystem . getStateList <= (unsigned int [NUM_PORTS]) list
```

The state list function reads all lower level port states in one call. It is equivalent to calling get state from the [PortClass](#) on each port.

Power Behavior (Get/Set)

```
usbsystem . getPowerBehavior <= (unsigned char) behavior  
usbsystem . setPowerBehavior => (unsigned char) behavior
```

The power behavior controls how power will be allocated to each lower level port. This behavior comes into play when the requested power of the system exceeds the available power. i.e. first come first server, even distribution, priority list. See the product datasheet for specific implementations.

Power Behavior Config (Get/Set)

```
usbsystem . getPowerBehaviorConfig <= (unsigned int) config  
usbsystem . setPowerBehaviorConfig => (unsigned int) config
```

Some power behaviors require a list of parameters in order to operate. For instance in priority list mode the user can supply a list of port indexes to priorities for power. This feature is product specific and users should consult the manual for further details.

Data Role Behavior (Get/Set)

```
usbsystem . getDataRoleBehavior <= (unsigned char) behavior
usbsystem . setDataRoleBehavior => (unsigned char) behavior
```

Some Type-C ports are capable of being dual role ports (DRP). Meaning they are capable of being either a host or a device. The behavior defined here will determine if that is allowed, what happens if it is, and what occurs when a host goes away. Examples are: first come first serve, priority list, static/fixed selection, etc. See the product datasheet for specific implementations.

Data Role Behavior Config (Get/Set)

```
usbsystem . getDataRoleBehaviorConfig <= (unsigned int) config
usbsystem . setDataRoleBehaviorConfig => (unsigned int) config
```

Many of the data role behaviors require a list of parameters in order to operate. For instance in a static/fixed mode the config would indicate what port is the upstream connection.

Data HighSpeed Max Datarate (Get/Set)

```
usbsystem . setDataHSMMaxDatarate <= (unsigned int) config
usbsystem . getDataHSMMaxDatarate => (unsigned int) config
```

The Max Datarate APIs will limit the device to a maximum specified datarate for the specific signal set. This API modifies the max datarate on the USB HighSpeed signals.

Enumeration	Name	Description
0	None	Configure HighSpeed Signals to no connection
1	Low Speed	Configure HighSpeed Signals to a maximum datarate of 1.5Mbps
2	Full Speed	Configure HighSpeed Signals to a maximum datarate of 12Mbps
3	High Speed	Configure HighSpeed Signals to a maximum datarate of 480Mbps

Data SuperSpeed Max Datarate (Get/Set)

```
usbsystem . setDataSSMaxDatarate <= (unsigned int) config
usbsystem . getDataSSMaxDatarate => (unsigned int) config
```

The Max Datarate APIs will limit the device to a maximum specified datarate for the specific signal set. This API modifies the max datarate on the USB SuperSpeed signals.

Enumeration	Name	Description
0	None	Configure SuperSpeed Signals to no connection
1	Super Speed	Configure SuperSpeed Signals to a maximum datarate of 5Gbps
2	Super Speed Plus	Configure SuperSpeed Signals to a maximum datarate of 10Gbps

Override (Get/Set)

```
usbsystem . getOverride <= (unsigned int) config  
usbsystem . setOverride => (unsigned int) config
```

The system inherently goes towards compliant behavior, in some conditions you may not want compliant behavior and this is what the override bit field allows. There are the following override bits that can be set.

Bit	Name	Description
0	Auto Vbus Toggle Disable	This bit is used to disable the auto vbus toggle behavior on re-enumeration of the upstream port.
1	Vbus Detect Disable	This bit is used to disable the requirement of an upstream connection for enabling the hub chip.

3.2 Python API Reference

Welcome to the BrainStem Python API reference documentation. This documentation covers the Python Acroname BrainStem module. This reference assumes that you understand the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation.

- [BrainStem Overview](#)
- [BrainStem Terminology](#)
- [Getting Started with the BrainStem.](#)

Next check out the python [Getting Started](#) section.

3.2.1 Getting (Quickly) Started

The BrainStem python package allows you to interact with a collection of BrainStem modules from python. The API is similar to both the C++ and Reflex API's, with a few significant differences. The remainder of this section details the structure and functionality of the python API.

Most modern operating systems come with all the tools needed to immediately install the BrainStem python libraries and create python based applications. As such, simply download the [latest development package](#)¹⁰⁴, and then use pip to install the library.

```
#> cd <path to extracted download>/development/python
#> pip install brainstem-*.whl
```

If you see errors from these commands, check the requirements and details below.

Requirements

The brainstem python package is currently compatible with the following versions of Python:

- **Python 2.7**
 - It is recommended that the python version be at least 2.7.9.
- **Python 3**
 - Minimum supported Python 3 version is 3.6

pip

The brainstem python package is installed via a platform specific wheel. To install these wheels you need a relatively up to date version of pip and setuptools. If you don't have pip installed you can install it by following the instructions at;

<https://pip.pypa.io/en/latest/installing.html>

If you do have pip installed it may be helpful to update pip. To do so run the following command from your command line. You may need to have administrator privileges on macOS and Linux. Instructions for updating pip can be found at;

<https://pip.pypa.io/en/latest/installing/#upgrade-pip>

¹⁰⁴ <https://acroname.com/software/brainstem-development-kit>

libffi

The Brainstem python library relies on libffi, on macOS and Windows this is generally available via pip. On Linux you may need to install libffi via your distro's package manager.

Python development headers

Also on Linux, you may need to install the development package for python via your distro's package manager before you can install.

CentOS package manager

On CentOS and yum based distros the following command will install the required packages.

```
$> sudo yum install libffi-devel python-devel
```

Installation

Install the python package.

Note: '#>' indicates that the command must be run with admin privileges on MacOS and Linux, either via sudo or su.

```
#> pip install brainstem-*.whl
```

If you need to uninstall the library, the easiest way to do so is with pip.

```
$> pip uninstall brainstem
```

A Tour of the Python Example

To run the example, go to Development/python in the “BrainStem2 Development Kit” package and type:

```
$> python brainstem_example.py
```

The example requires that you have a USB BrainStem link module connected to your host computer. If you see the following message, you probably don't have a module connected:

```
Creating USB stem and connecting to first module found  
Could not find a module.
```

Once the example starts running, it will connect to the first USBStem it finds connected to your computer and then blink the user LED on the module.

```
$> python brainstem_example.py  
Creating USB stem and connecting to first module found  
Connecting to Module with serial number: 0x40F5849A  
Flashing the user LED
```

The following is a brief introduction interacting with the brainstem via the python interactive interpreter. The first step is to import some modules that we'll need later. There are multiple ways to import the brainstem package. For this example we will use the simplest method.

```
>>> import brainstem
```

See the *Package Structure* <package> section of the python reference for more information about the brainstem package, and the modules it includes.

Next we discover a USBStem module, and connect to it.

```
>>> spec = brainstem.discover.findFirstModule(brainstem.link.Spec.USB)
>>> print spec
LinkType: USB(serial: 0x40F5849A, module: 0)
>>> stem = brainstem.stem.USBStem()
>>> stem.connect(0x40F5849A)
```

Information about specific modules can be found in the *Modules* <Modules> section.

Now that we have created a *USBStem*, we can turn on the user LED using the *system* entity:

```
>>> stem.system.setLED(1)
```

Finally lets blink the LED in a loop.

```
>>> from time import sleep
>>> for i in range(0,100):
...     err = stem.system.setLED(i % 2)
...     if err != 0:
...         print "error %d"% err
...         break
...     sleep(0.5)
...
>>>
```

As you can see the call to setLED returns an error value. In this case that is an error value, that will be 0 on success and some other number if there is an error. The brainstem library generally avoids raising exceptions, and instead passes information via result objects, or result error codes. More information about these errors, and the result object can be found in the *Result* <result> section of the python reference

Help is available from within the python interpreter, calling help() on a stem or other object will yield context specific documentation.

```
>>> import brainstem
>>> help(brainstem.stem.USBStem)
Help on class USBStem in module brainstem.stem:

class USBStem(brainstem.module.Module)
| Concrete Module implementation for 40Pin and MTM USBStem modules
|
| USBStem modules contain Analogs, Digital IO's, and I2C entities
| in addition to the system entity.
|
| Method resolution order:
|     USBStem
|
| ...
```

Enjoy!

The Acroname Team.

Support

If you are having issues, please let us know. We have a mailing list located at: support@acroname.com

3.2.2 Acroname Modules

Quick Access:

- *USBHub3c*
- *USBHub3p*
- *USBHub2x4*
- *USBCSwitchPro*
- *USBCSwitch*
- *MTMDAQ2*
- *MTMEtherStem*
- *MTMIOSerial*
- *MTMLOAD1*
- *MTMPM1*
- *MTMRelay*
- *MTMUSBStem*
- *MTMDAQ1*
- *EtherStem*
- *USBStem*

Each type of BrainStem module is represented by a corresponding concrete Module implementation. The following classes are instantiated to allow communication through to the corresponding BrainStem module hardware.

The instantiation and subsequent connection to each module is as follows

```
>>> stem = USBStem()  
# 0XXXXXXXXX is the serial number of the module.  
>>> stem.connect(0XXXXXXXXX)
```

Connecting to the BrainStem module can take multiple forms, the simplest way to connect when you know the module's serial number is to call connect with the serial number, as in the above code snippet. If you don't know the serial number of the module, you can perform a discovery of the modules currently connected and print that information. For details of the connection functions API please see [Connections](#) in this reference.

USBHub3c

```
class brainstem.stem.USBHub3c (address=6, enable_auto_networking=True, model=0)
```

Concrete Module implementation for the USBHub3c.

The module contains the USB entity as well as the following.

Entities:

- system
- app[0-3]
- pointers[0-3]
- store[0-2]
- temperature[0-2]
- timer[0-7]
- hub
- hub.port[0-7]
- rail[0-6]
- pd[0-7]
- usb
- uart[0-1]

Useful Constants:

- BASE_ADDRESS (6)
- NUMBER_OF_STORES (2)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_USB_PORTS (8)
- NUMBER_OF_UARTS (2)
- NUMBER_OF_RAILS (7)
- STORE_INTERNAL_INDEX (0)
- STORE_RAM_INDEX (1)
- STORE_EEPROM_INDEX (2)
- PORT_ID_CONTROL_INDEX (6)
- PORT_ID_POWER_C_INDEX (7)
- NUMBER_OF_PORTS (8)

```
class Hub (module, index)
```

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

i2c

usb entity adds minimal legacy support

[Back to the top](#)

USBHub3p

class brainstem.stem.**USBHub3p** (*address=6, enable_auto_networking=True, model=0*)

Concrete Module implementation for the USBHub3p.

The module contains the USB entity as well as the following.

Entities:

- system
- app[0-3]
- pointers[0-3]
- usb
- store[0-1]
- temperature
- timer[0-7]

Useful Constants:

- BASE_ADDRESS (6)
- NUMBER_OF_STORES (2)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_DOWNSTREAM_USB (8)
- NUMBER_OF_UPSTREAM_USB (2)
- NUMBER_OF_PORTS (12)

Bit defines for port state UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if (state & brainstem.BIT(aUSBHUB3P_USB_VBUS_ENABLED))

- aUSBHUB3P_USB_VBUS_ENABLED (0)

- aUSBHUB3P_USB2_DATA_ENABLED (1)
- aUSBHUB3P_USB3_DATA_ENABLED (3)
- aUSBHUB3P_USB_SPEED_USB2 (11)
- aUSBHUB3P_USB_SPEED_USB3 (12)
- aUSBHUB3P_USB_ERROR_FLAG (19)
- aUSBHUB3P_USB2_BOOST_ENABLED (20)
- aUSBHUB3P_DEVICE_ATTACHED (23)

Bit defines for port error UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if (error & brainstem.BIT(aUSBHUB3P_ERROR_VBUS_OVERCURRENT))

- aUSBHUB3P_ERROR_VBUS_OVERCURRENT (0)
- aUSBHUB3P_ERROR_VBUS_BACKDRIVE (1)
- aUSBHUB3P_ERROR_HUB_POWER (2)
- aUSBHUB3P_ERROR_OVER_TEMPERATURE (3)

class Hub (*module, index*)

connect (*serial_number, **kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

[Back to the top](#)

USBHub2x4

class brainstem.stem.USBHub2x4 (*address=6, enable_auto_networking=True, model=0*)

Concrete Module implementation for the USBHub2x4.

The module contains the USB entity as well as the following.

Entities:

- system
- app[0-3]
- pointer[0-3]
- usb
- mux
- store[0-1]
- temperature
- timer[0-7]

Useful Constants:

- `BASE_ADDRESS` (6)
- `NUMBER_OF_STORES` (3)
- `NUMBER_OF_INTERNAL_SLOTS` (12)
- `NUMBER_OF_RAM_SLOTS` (1)
- `NUMBER_OF_TIMERS` (8)
- `NUMBER_OF_APPS` (4)
- `NUMBER_OF_POINTERS` (4)
- `NUMBER_OF_DOWNSTREAM_USB` (4)
- `NUMBER_OF_UPSTREAM_USB` (2)
- `NUMBER_OF_PORTS` (6)

Bit defines for port error UInt32 use `brainstem.BIT(X)` from `aDefs.h` to get bit value. i.e if (error & `brainstem.BIT(aUSBHUB2X4_USB_VBUS_ENABLED)`)

- `aUSBHUB2X4_USB_VBUS_ENABLED` (0)
- `aUSBHUB2X4_USB2_DATA_ENABLED` (1)
- `aUSBHUB2X4_USB_ERROR_FLAG` (19)
- `aUSBHUB2X4_USB2_BOOST_ENABLED` (20)
- `aUSBHUB2X4_DEVICE_ATTACHED` (23)
- `aUSBHUB2X4_CONSTANT_CURRENT` (24)

Bit defines for port error UInt32 use `brainstem.BIT(X)` from `aDefs.h` to get bit value. i.e if (error & `brainstem.BIT(aUSBHUB3P_ERROR_VBUS_OVERCURRENT)`)

- `aUSBHUB2X4_ERROR_VBUS_OVERCURRENT` (0)
- `aUSBHUB2X4_ERROR_OVER_TEMPERATURE` (3)
- `aUSBHub2X4_ERROR_DISCHARGE` (4)

class `Hub` (*module, index*)

connect (*serial_number, **kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

USBCSwitchPro

```
class brainstem.stem.USBCSwitchPro (address=16, enable_auto_networking=True,  
                                     model=0)
```

Concrete Module implementation for the USBCSwitchPro.

The module contains the USB entity as well as the following.

Entities:

- system
- rail[0]
- store[0-2]
- temperature[0-4]
- i2c[0]
- usb
- uart[0-1]
- port[0-5]
- pd[0-5]
- mux
- digital[0]

Useful Constants:

- BASE_ADDRESS (16)
- NUMBER_OF_STORES (3)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_TEMPERATURES (5)
- NUMBER_OF_I2CS (1)
- NUMBER_OF_USB (1)
- NUMBER_OF_UARTS (2)
- NUMBER_OF_PORTS (6)
- NUMBER_OF_POWER_DELIVERY_PORTS (6)
- NUMBER_OF_MUXS (1)
- NUMBER_OF_DIGITALS (1)
- PORT_ID_0 (0)
- PORT_ID_1 (1)
- PORT_ID_2 (2)
- PORT_ID_3 (3)
- PORT_ID_COMMON (4)
- PORT_ID_CONTROL (5)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

[Back to the top](#)

USBCSwitch

```
class brainstem.stem.USBCSwitch (address=6, enable_auto_networking=True,  
                                model=0)
```

Concrete Module implementation for the USBC-Switch.

The module contains the USB entity as well as the following.

Entities:

- system
- usb
- mux
- store[0-1]
- equalizer[0-1]

Useful Constants:

- BASE_ADDRESS (6)
- NUMBER_OF_STORES (3)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_USB (1)
- NUMBER_OF_MUXS (1)
- NUMBER_OF_EQUALIZERS (2)

Bit defines for port state UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if (state & brainstem.BIT(usbPortStateVBUS))

- usbPortStateVBUS (0)
- usbPortStateUSB2A (1)
- usbPortStateUSB2B (2)
- usbPortStateSBU (3)
- usbPortStateSS1 (4)
- usbPortStateSS2 (5)
- usbPortStateCC1 (6)

- usbPortStateCC2 (7)
- usbPortStateCCFlip (14)
- usbPortStateSSFlip (15)
- usbPortStateSBUFlip (16)
- usbPortStateUSB2Flip (17)
- usbPortStateCC1Inject (26)
- usbPortStateCC2Inject (27)
- usbPortStateCC1Detect (28)
- usbPortStateCC2Detect (29)
- usbPortStateCC1LogicState (30)
- usbPortStateCC2LogicState (31)
- TRANSMITTER_2P0_40mV (0)
- TRANSMITTER_2P0_60mV (1)
- TRANSMITTER_2P0_80mV (2)
- TRANSMITTER_2P0_0mV (3)
- MUX_1db_COM_0db_900mV (0)
- MUX_0db_COM_1db_900mV (1)
- MUX_1db_COM_1db_900mV (2)
- MUX_0db_COM_0db_900mV (3)
- MUX_0db_COM_0db_1100mV (4)
- MUX_1db_COM_0db_1100mV (5)
- MUX_0db_COM_1db_1100mV (6)
- MUX_2db_COM_2db_1100mV (7)
- MUX_0db_COM_0db_1300mV (8)
- LEVEL_1_2P0 (0)
- LEVEL_2_2P0 (1)
- LEVEL_1_3P0 (0)
- LEVEL_2_3P0 (1)
- LEVEL_3_3P0 (2)
- LEVEL_4_3P0 (3)
- LEVEL_5_3P0 (4)
- LEVEL_6_3P0 (5)
- LEVEL_7_3P0 (6)
- LEVEL_8_3P0 (7)
- LEVEL_9_3P0 (8)
- LEVEL_10_3P0 (9)

- LEVEL_11_3P0 (10)
- LEVEL_12_3P0 (11)
- LEVEL_13_3P0 (12)
- LEVEL_14_3P0 (13)
- LEVEL_15_3P0 (14)
- LEVEL_16_3P0 (15)
- EQUALIZER_CHANNEL_BOTH (0)
- EQUALIZER_CHANNEL_MUX (1)
- EQUALIZER_CHANNEL_COMMON (2)
- NO_DAUGHTERCARD (0)
- PASSIVE_DAUGHTERCARD (1)
- REDRIVER_DAUGHTERCARD (2)
- UNKNOWN_DAUGHTERCARD (3)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

[Back to the top](#)

MTMDAQ2

class brainstem.stem.**MTMDAQ2** (*address=10, enable_auto_networking=True, model=0*)

Concrete Module implementation for MTM-DAQ-2 module

MTM-DAQ-2 modules contain contain the following entities:

- system
- app[0-3]
- digital[0-1]
- analog[0-19]
- i2c[0]
- pointer[0-3]
- store[0-1]
- timer[0-7]

Useful Constants:

- BASE_ADDRESS (10)
- NUMBER_OF_STORES (2)

- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_DIGITALS (2)
- NUMBER_OF_ANALOGS (20)
- NUMBER_OF_I2C (1)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- ANALOG_RANGE_P0V064N0V064 (0)
- ANALOG_RANGE_P0V64N0V64 (1)
- ANALOG_RANGE_P0V128N0V128 (2)
- ANALOG_RANGE_P1V28N1V28 (3)
- ANALOG_RANGE_P1V28N0V0 (4)
- ANALOG_RANGE_P0V256N0V256 (5)
- ANALOG_RANGE_P2V56N2V56 (6)
- ANALOG_RANGE_P2V56N0V0 (7)
- ANALOG_RANGE_P0V512N0V512 (8)
- ANALOG_RANGE_P5V12N5V12 (9)
- ANALOG_RANGE_P5V12N0V0 (10)
- ANALOG_RANGE_P1V024N1V024 (11)
- ANALOG_RANGE_P10V24N10V24 (12)
- ANALOG_RANGE_P10V24N0V0 (13)
- ANALOG_RANGE_P2V048N0V0 (14)
- ANALOG_RANGE_P4V096N0V0 (15)
- ANALOG_BULK_CAPTURE_MAX_HZ (500000)
- ANALOG_BULK_CAPTURE_MIN_HZ (1)

connect (*serial_number*, ****kwargs**)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

MTMEtherStem

```
class brainstem.stem.MTMEtherStem (address=4, enable_auto_networking=True,  
                                     model=0)
```

Concrete Module implementation for MTM EtherStem modules

USBStem modules contain the following entities:

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]
- servo[0-7]
- store[0-2]
- timer[0-7]

Useful Constants:

- BASE_ADDRESS (4)
- NUMBER_OF_STORES (3)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_SD_SLOTS (255)
- NUMBER_OF_ANALOGS (4)
- DAC_ANALOG_INDEX (3)
- FIXED_DAC_ANALOG (False)
- NUMBER_OF_DIGITALS (15)
- NUMBER_OF_I2C (2)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- NUMBER_OF_SERVOS (8)
- NUMBER_OF_SERVO_OUTPUTS (4)
- NUMBER_OF_SERVO_INPUTS (4)
- ANALOG_BULK_CAPTURE_MAX_HZ (200000)
- ANALOG_BULK_CAPTURE_MIN_HZ (7000)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

MTMIOSerial

```
class brainstem.stem.MTMIOSerial (address=8, enable_auto_networking=True,  
                                model=0)
```

Concrete Module implementation for MTM-IO-Serial module

MTM-IO-SERIAL modules contain contain the following entities:

- system
- app[0-3]
- digital[0-8]
- i2c[0]
- pointer[0-3]
- servo[0-7]
- signal[0-4]
- store[0-1]
- temperature
- timer[0-7]
- uart[0-3]
- rail[0-2]

Useful Constants:

- BASE_ADDRESS (8)
- NUMBER_OF_STORES (2)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_DIGITALS (8)
- NUMBER_OF_I2C (1)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- NUMBER_OF_UART (1)

- `NUMBER_OF_RAILS` (3)
- `NUMBER_OF_SERVOS` (8)
- `NUMBER_OF_SERVO_OUTPUTS` (4)
- `NUMBER_OF_SERVO_INPUTS` (4)
- `NUMBER_OF_SIGNALS` (5)
- `NUMBER_OF_USB` (1)
- `NUMBER_OF_USB_PORTS` (4)
- `NUMBER_OF_PORTS` (5)
- `aMTMIO SERIAL_USB_VBUS_ENABLED` (0)
- `aMTMIO SERIAL_USB2_DATA_ENABLED` (1)
- `aMTMIO SERIAL_USB_ERROR_FLAG` (19)
- `aMTMIO SERIAL_USB2_BOOST_ENABLED` (20)
- `aMTMIO SERIAL_ERROR_VBUS_OVERCURRENT` (0)

class `Hub` (*module, index*)

connect (*serial_number, **kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

MTMLOAD1

class `brainstem.stem.MTMLOAD1` (*address=14, enable_auto_networking=True, model=0*)

Concrete Module implementation for MTM-LOAD-1 module

MTM-LOAD-1 modules contain contain the following entities:

- `system`
- `app[0-3]`
- `digital[0-3]`
- `i2c[0]`
- `pointer[0-3]`
- `store[0-1]`
- `timer[0-7]`
- `rail[0]`
- `temperature`

Useful Constants:

- `BASE_ADDRESS` (14)
- `NUMBER_OF_STORES` (2)
- `NUMBER_OF_INTERNAL_SLOTS` (12)
- `NUMBER_OF_RAM_SLOTS` (1)
- `NUMBER_OF_DIGITALS` (2)
- `NUMBER_OF_I2C` (1)
- `NUMBER_OF_POINTERS` (4)
- `NUMBER_OF_TIMERS` (8)
- `NUMBER_OF_APPS` (4)
- `NUMBER_OF_RAILS` (2)
- `NUMBER_OF_TEMPERATURES` (1)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

MTMPM1

class `brainstem.stem.MTMPM1` (*address=6, enable_auto_networking=True, model=0*)

Concrete Module implementation for MTM-PM-1 module

MTM-PM-1 modules contain contain the following entities:

- `system`
- `app[0-3]`
- `digital[0-1]`
- `i2c[0]`
- `pointer[0-3]`
- `store[0-1]`
- `timer[0-7]`
- `rail[0-1]`
- `temperature`

Useful Constants:

- `BASE_ADDRESS` (6)
- `NUMBER_OF_STORES` (2)

- `NUMBER_OF_INTERNAL_SLOTS` (12)
- `NUMBER_OF_RAM_SLOTS` (1)
- `NUMBER_OF_DIGITALS` (2)
- `NUMBER_OF_I2C` (1)
- `NUMBER_OF_POINTERS` (4)
- `NUMBER_OF_TIMERS` (8)
- `NUMBER_OF_APPS` (4)
- `NUMBER_OF_RAILS` (2)
- `NUMBER_OF_TEMPERATURES` (1)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

MTMRelay

class `brainstem.stem.MTMRelay` (*address=12, enable_auto_networking=True, model=0*)

Concrete Module implementation for MTM-RELAY module

MTM-RELAY modules contain contain the following entities:

- `system`
- `app[0-3]`
- `digital[0-3]`
- `i2c[0]`
- `pointer[0-3]`
- `store[0-1]`
- `timer[0-7]`
- `relay[0-3]`
- `temperature`

Useful Constants:

- `BASE_ADDRESS` (12)
- `NUMBER_OF_STORES` (2)
- `NUMBER_OF_INTERNAL_SLOTS` (12)
- `NUMBER_OF_RAM_SLOTS` (1)
- `NUMBER_OF_DIGITALS` (4)

- `NUMBER_OF_I2C` (1)
- `NUMBER_OF_POINTERS` (4)
- `NUMBER_OF_TIMERS` (8)
- `NUMBER_OF_APPS` (4)
- `NUMBER_OF_RELAYS` (4)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

MTMUSBStem

```
class brainstem.stem.MTMUSBStem (address=4, enable_auto_networking=True,  
                                model=0)
```

Concrete Module implementation for MTM USBStem modules

MTMUSBStem modules contain the following entities:

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]
- servo[0-7]
- signal[0-4]
- store[0-2]
- timer[0-7]

Useful Constants:

- `BASE_ADDRESS` (4)
- `NUMBER_OF_STORES` (3)
- `NUMBER_OF_INTERNAL_SLOTS` (12)
- `NUMBER_OF_RAM_SLOTS` (1)
- `NUMBER_OF_SD_SLOTS` (255)
- `NUMBER_OF_ANALOGS` (4)

- DAC_ANALOG_INDEX (3)
- FIXED_DAC_ANALOG (True)
- NUMBER_OF_DIGITALS (15)
- NUMBER_OF_I2C (2)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- NUMBER_OF_SERVOS (8)
- NUMBER_OF_SERVO_OUTPUTS (4)
- NUMBER_OF_SERVO_INPUTS (4)
- NUMBER_OF_SIGNALS (5)
- ANALOG_BULK_CAPTURE_MAX_HZ (200000)
- ANALOG_BULK_CAPTURE_MIN_HZ (7000)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

[Back to the top](#)

MTMDAQ1

class brainstem.stem.**MTMDAQ1** (*address=10, enable_auto_networking=True, model=0*)

Concrete Module implementation for MTM-DAQ-1 module

MTM-DAQ-1 modules contain contain the following entities:

- system
- app[0-3]
- digital[0-1]
- analog[0-19]
- i2c[0]
- pointer[0-3]
- store[0-1]
- timer[0-7]

Useful Constants:

- BASE_ADDRESS (10)
- NUMBER_OF_STORES (2)

- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_DIGITALS (2)
- NUMBER_OF_ANALOGS (20)
- NUMBER_OF_I2C (1)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- ANALOG_RANGE_P0V064N0V064 (0)
- ANALOG_RANGE_P0V64N0V64 (1)
- ANALOG_RANGE_P0V128N0V128 (2)
- ANALOG_RANGE_P1V28N1V28 (3)
- ANALOG_RANGE_P1V28N0V0 (4)
- ANALOG_RANGE_P0V256N0V256 (5)
- ANALOG_RANGE_P2V56N2V56 (6)
- ANALOG_RANGE_P2V56N0V0 (7)
- ANALOG_RANGE_P0V512N0V512 (8)
- ANALOG_RANGE_P5V12N5V12 (9)
- ANALOG_RANGE_P5V12N0V0 (10)
- ANALOG_RANGE_P1V024N1V024 (11)
- ANALOG_RANGE_P10V24N10V24 (12)
- ANALOG_RANGE_P10V24N0V0 (13)
- ANALOG_RANGE_P2V048N0V0 (14)
- ANALOG_RANGE_P4V096N0V0 (15)

connect (*serial_number*, ****kwargs**)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

EtherStem

```
class brainstem.stem.EtherStem (address=2, enable_auto_networking=True,  
                                model=0)
```

Concrete Module implementation for 40Pin EtherStem modules

EtherStem modules contain the following entities:

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]
- servo[0-7]
- store[0-2]
- timer[0-7]

Useful Constants:

- BASE_ADDRESS (2)
- NUMBER_OF_STORES (3)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_SD_SLOTS (255)
- NUMBER_OF_ANALOGS (4)
- DAC_ANALOG_INDEX (3)
- FIXED_DAC_ANALOG (False)
- NUMBER_OF_DIGITALS (15)
- NUMBER_OF_I2C (2)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- NUMBER_OF_SERVOS (8)
- NUMBER_OF_SERVO_OUTPUTS (4)
- NUMBER_OF_SERVO_INPUTS (4)
- ANALOG_BULK_CAPTURE_MAX_HZ (200000)
- ANALOG_BULK_CAPTURE_MIN_HZ (7000)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

[Back to the top](#)

USBStem

class brainstem.stem.**USBStem** (*address=2, enable_auto_networking=True, model=0*)

Concrete Module implementation for 40Pin USBStem modules

USBStem modules contain contain the following entities:

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]
- servo[0-7]
- store[0-2]
- timer[0-7]

Useful Constants:

- BASE_ADDRESS (2)
- NUMBER_OF_STORES (3)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_SD_SLOTS (255)
- NUMBER_OF_ANALOGS (4)
- DAC_ANALOG_INDEX (3)
- FIXED_DAC_ANALOG (False)
- NUMBER_OF_DIGITALS (15)
- NUMBER_OF_I2C (2)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)

- `NUMBER_OF_APPS` (4)
- `NUMBER_OF_SERVOS` (8)
- `NUMBER_OF_SERVO_OUTPUTS` (4)
- `NUMBER_OF_SERVO_INPUTS` (4)
- `ANALOG_BULK_CAPTURE_MAX_HZ` (200000)
- `ANALOG_BULK_CAPTURE_MIN_HZ` (7000)

connect (*serial_number*, ***kwargs*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

[Back to the top](#)

3.2.3 Package Structure

The BrainStem package consists of a number of modules, which together form the BrainStem python API.

brainstem.module

A module that provides base classes for BrainStem Modules and Entities.

The Module and Entity classes are designed to be extended for specific types of BrainStem Modules and Entities. For more information about Brainstem Modules and Entities, please see the [Terminology](#)¹⁰⁵ section of the [Acroname BrainStem Reference](#)¹⁰⁶

brainstem.stem

Provides specific module instances, and entity functionality.

The Module and Entity classes contained in this module provide the core API functionality for all of the Brainstem modules. For more information about possible entities please see the [Entity](#)¹⁰⁷ section of the [Acroname BrainStem Reference](#)¹⁰⁸

¹⁰⁵ <https://acroname.com/reference/brainstem/terms.html>

¹⁰⁶ <https://acroname.com/reference>

¹⁰⁷ <https://acroname.com/reference/api/entities>

¹⁰⁸ <https://acroname.com/reference>

brainstem.link

A module that provides a Spec class for specifying a connection to a BrainStem module.

A Spec instance fully describes a connection to a brainstem module. In the case of USB based stems this is simply the serial number of the module. For TCPIP based stems this is an IP address and TCP port.

For more information about links and the Brainstem network see the [Acroname BrainStem Reference](#)¹⁰⁹

brainstem.discover

A module that provides methods for discovering brainstem modules over USB and TPCIP.

The discovery module provides an interface for locating BrainStem modules accross multiple transports. It provides a way to find all modules for a give transport as well as specific modules by serial number, or first found. The result of a call to one of the discovery functions is either a list of brainstem.link.Spec objects, or a single brainstem.link.Spec.

The Discovery module allows users to find specific brainstem devices via their serial number, or a list of all devices connected to the host via usb or on the same subnet via TCP/IP. In all cases a *Spec* object is returned with connection details for the device. In addition do connection details, the BrainStem model is returned. This model is one of a list of BrainStem device model numbers which are accessible via the *defs* module.

A typical interactive python session finding all connected USB modules might look like the following.

```
>> import brainstem >> module_list = brainstem.discover.findAllModules(brainstem.link.Spec.USB)
>> print [str(s) for s in module_list] ['Model: 4 LinkType: USB(serial: 0xCB4A3B25, module: 0)',
'Model: 13 LinkType: USB(serial: 0x40F5849A, module: 0)']
```

For an overview of links, discovery and the Brainstem network see the [Acroname BrainStem Reference](#)¹¹⁰

brainstem.defs

A module that provides defines and constants useful for working with the python library.

brainstem.result

A module that provides a result class for returning results of UEI commands.

Results consist of an error attribute and a value attribute. If the error attribute is set to NO_ERROR, then the result value is the response to the UEI command that was sent.

For more information about return values for commands and UEI's see the [Acroname BrainStem Reference](#)¹¹¹

¹⁰⁹ <https://acroname.com/reference>

¹¹⁰ <https://acroname.com/reference>

¹¹¹ <https://acroname.com/reference>

brainstem.version

Provides version access utilities.

3.2.4 Definitions

A module that provides defines and constants useful for working with the python library.

```
brainstem.defs.model_info(model)
```

Get Model information.

Parameters

model (*int*) – One of the model numbers, i.e from stem.system.getModel().

Returns

String containing model information.

```
brainstem.defs.model_name(model)
```

Get Model Name.

Parameters

model (*int*) – One of the model numbers, i.e from stem.system.getModel().

Returns

A string containing model name.

3.2.5 Discovery

A module that provides methods for discovering brainstem modules over USB and TPCIP.

The discovery module provides an interface for locating BrainStem modules accross multiple transports. It provides a way to find all modules for a give transport as well as specific modules by serial number, or first found. The result of a call to one of the discovery functions is either a list of brainstem.link.Spec objects, or a single brainstem.link.Spec.

The Discovery module allows users to find specific brainstem devices via their serial number, or a list of all devices connected to the host via usb or on the same subnet via TCP/IP. In all cases a *Spec* object is returned with connection details for the device. In addition do connection details, the BrainStem model is returned. This model is one of a list of BrainStem device model numbers which are accessible via the *defs* module.

A typical interactive python session finding all connected USB modules might look like the following.

```
>> import brainstem >> module_list = brainstem.discover.findAllModules(brainstem.link.Spec.USB)
>> print [str(s) for s in module_list] ['Model: 4 LinkType: USB(serial: 0xCB4A3B25, module: 0)',
'Model: 13 LinkType: USB(serial: 0x40F5849A, module: 0)']
```

For an overview of links, discovery and the Brainstem network see the [Acroname BrainStem Reference](https://acroname.com/reference)¹¹²

```
class brainstem.discover.DeviceNode
```

Python representation of DeviceNode_t (C structure)

- **hub_serial_number** (uint32_t): Serial number of the Acroname hub where the device was found.
- **hub_port** (uint8_t): Port of the Acroname hub where the device was found.
- **id_vendor** (uint16_t): Manufactures Vendor ID of the downstream device.

¹¹² <https://acroname.com/reference>

- `id_product` (`uint16_t`): Manufactures Product ID of the downstream device.
- **speed (enumeration): The devices downstream device speed.**
 - Unknown (0)
 - Low Speed (1)
 - Full Speed (2)
 - High Speed (3)
 - Super Speed (4)
 - Super Speed Plus (5)
- `product_name` (string): USB string descriptor.
- `manufacture` (string): USB string descriptor.
- `serial_number` (string): USB string descriptor.

`brainstem.discover.findAllModules` (*transports*, *network_interface=0*, *buffer_length=128*)

Return a list of Specs for all modules found on the transports given.

Transports can be presented as a list, and the results would be a list of all modules found for those transports. TCPIP modules take a little longer to find due to the Multicast and gather necessary for finding modules on the local network segment.

Parameters

- **transports** (*int or list(int)*) – A list of transports or a single transport.
- **network_interface** (*unsigned int*) – The network interface to use for the discovery.
- **buffer_length** (*unsigned int*) – The length of the buffer to use for the discovery.

Returns

A list of the Spec objects for all modules found.

Return type

`list(Spec)`

`brainstem.discover.findFirstModule` (*transports*, *network_interface=0*)

Return the Spec for the first module found on the given transport.

Parameters

- **transports** (*int or list(int)*) – A list of transports or a single transport.
- **network_interface** (*unsigned int*) – The network interface to use for the discovery.

Returns

The connection spec of the first module found on the given transport.

Return type

`Spec`

`brainstem.discover.findModule` (*transports*, *serial_number*, *network_interface=0*)

Return the Spec for the module with the given serial number.

Transports can be presented as a list. TCPIP modules take a little longer to find due to the Multicast and gather necessary for finding modules on the local network segment.

Parameters

- **transports** (*int or list(int)*) – A list of transports or a single transport.
- **serial_number** (*unsigned int*) – The module serial_number to look for.
- **network_interface** (*unsigned int*) – The network interface to use for the discovery.

Returns

The connection spec for the module whose serial number is given in the args.

```
brainstem.discover.getDownstreamDevices (list_length=128)
```

Gets downstream device USB information for all Acroname hubs.

Parameters

list_length – The amount of memory to provide for the lower level C call.

Returns

Result object containing NO_ERROR and a tuple of DeviceNode's containing the detected downstream devices:: - **aErrParam**: Passed in values are not valid (NULL, size, etc). - **aErrMemory**: No more room in the list. - **aErrNotFound**: No Acroname devices were found.

Return type

Result

```
brainstem.discover.getIPv4Interfaces (list_length=30)
```

Populates a list with all of the available IPv4 Interfaces.

Parameters

list_length (*unsigned int*) – Size of list to allocate for.

Returns

A tuple of IPv4 interfaces.

Return type

tuple(unsigned int)

3.2.6 Entity

```
class brainstem.Entity_Entity.Entity (module, cmd, index)
```

Base class for BrainStem Entity.

Provides the default implementation for a functional entity within the BrainStem. This can include IO like GPIOs, Analogs etc. For a more detailed description of Entities see the [Terminology](#)¹¹³ section of the brainstem reference for more information.

```
call_UEI (option)
```

Call a set UEI on this entity.

Parameters

option (*byte*) – The command option.

Returns

An error result from the list of defined error codes in brainstem.result

```
property command
```

Return the entity command.

Type

int

drain_UEI (*option*)

Drain UEI packets matching option.

Parameters

option (*byte*) – The command option.

Returns

An error result from the list of defined error codes in brainstem.result

getStreamStatus (*buffer_length=1024*)

Gets all available stream values associated with the cmd and index of the called API.

Parameters

buffer_length (*unsigned int*) – Size of the buffer to allocate

Returns

An error result from the list of defined error codes in brainstem.result

get_UEI16 (*option*)

Get a UEI short value.

Parameters

option (*byte*) – The command option.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

get_UEI16_with_subindex (*option, subIndex*)

Call a get UEI short value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

get_UEI32 (*option*)

Get a UEI int value.

Parameters

option (*byte*) – The command option.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

get_UEI32_with_subindex (*option, subIndex*)

Call a get UEI int value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

get_UEI8 (*option*)

Get a UEI byte value.

Parameters

option (*byte*) – The command option.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

get_UEI8_with_subindex (*option*, *subIndex*)

Call a get UEI byte value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

get_UEIBytes (*option*, *buffer_length=65536*)

Get a UEI Bytes buffer on this entity.

Parameters

- **option** (*byte*) – The command option.
- **buffer_length** (*unsigned int*) – The subIndex of the entity.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

property index

Return the entity index

Type

int

loadEntityFromSavedValues ()

Load the Entity from memory.

Parameters

option (*byte*) – The command option.

Returns

An error result from the list of defined error codes in brainstem.result

property module

returns the associated module object.

Type

Module

registerOptionCallback (*option*, *enable*, *cb*, *pRef*)

Registers a callback function based on a specific option code. Option code applies to the cmd and index of the called API.

:param option The option code for the entities command and index. :type option: byte

:param enable Enable (True) or disable (False) streaming. :type enable: bool

:param cb Callback to be executed on the provided criteria. :type cb:
@ffi.callback("unsigned char(aPacket*, void*)")

:param pRef Handle to be passed to the provided callback. This handle must be kept alive by the caller. :type pRef: ffi handle

Returns

An error result from the list of defined error codes in brainstem.result

resetEntityToFactoryDefaults()

Resets the Entity to factory defaults

Parameters

option (*byte*) – The command option.

Returns

An error result from the list of defined error codes in brainstem.result

saveEntity()

Saves the Entity.

Parameters

option (*byte*) – The command option.

Returns

An error result from the list of defined error codes in brainstem.result

setStreamEnabled(enable)

Enables streaming for all possible option codes within the cmd and index the entity was created for.

Parameters

enable (*bool*) – Enable (True) or disable (False) streaming.

Returns

An error result from the list of defined error codes in brainstem.result

set_UEI16(option, value)

Call a set UEI with short value on this entity.

Parameters

- **option** (*byte*) – The command option.
- **value** (*short*) – The short parameter to send.

Returns

An error result from the list of defined error codes in brainstem.result

set_UEI16_with_subindex(option, subIndex, value)

Call a set UEI short value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.
- **value** (*short*) – The short parameter to send.

Returns

An error result from the list of defined error codes in brainstem.result

set_UEI32(option, value)

Call a set UEI with int value on this entity.

Parameters

- **option** (*byte*) – The command option.
- **value** (*int*) – The int parameter to send.

set_UEI32_with_subindex(option, subIndex, value)

Call a set UEI int value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.
- **value** (*int*) – The int parameter to send.

Returns

An error result from the list of defined error codes in `brainstem.result`

set_UEI8 (*option, value*)

Call a set UEI with byte value on this entity.

Parameters

- **option** (*byte*) – The command option.
- **value** (*byte*) – The byte parameter to send.

Returns

An error result from the list of defined error codes in `brainstem.result`

set_UEI8_with_subindex (*option, subIndex, value*)

Call a set UEI byte value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.
- **value** (*byte*) – The byte parameter to send.

Returns

An error result from the list of defined error codes in `brainstem.result`

set_UEIBytes (*option, buffer*)

Call a set UEI with buffer and length of buffer on this entity.

Parameters

- **option** (*byte*) – The command option.
- **buffer** (*bytearray()*) – The buffer to be sent

Returns

An error result from the list of defined error codes in `brainstem.result`

3.2.7 Link

A module that provides a Spec class for specifying a connection to a BrainStem module.

A Spec instance fully describes a connection to a brainstem module. In the case of USB based stems this is simply the serial number of the module. For TCPIP based stems this is an IP address and TCP port.

For more information about links and the Brainstem network see the [Acroname BrainStem Reference](https://acroname.com/reference/brainstem/terms.html)¹¹³

class `brainstem.link.Spec` (*transport, serial_number, module, model, **keywords*)

Spec class for specifying connection details

Instances of Spec represent the connection details for a brainstem link. The Spec class also contains constants representing the possible transport types for BrainStem modules.

Parameters

- **transport** (*int*) – One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*int*) – The module serial number.
- **module** – The module address on the Brainstem network.
- **model** – The device model number of the Brainstem module.

¹¹³ <https://acroname.com/reference/brainstem/terms.html>

¹¹⁴ <https://acroname.com/reference>

- ****keywords** – For TCPIP, SERIAL and AETHER connections. The possibilities are,
 - `ip_address`: (int/str) The IPV4 address for a TCPIP/AETHER connection type.
 - `ip_port`: (int/str) The port for a TCPIP/AETHER connection type.
 - `port`: (str) The serial port for a SERIAL connection type.
 - `baudrate`: (int/str) The baudrate for a SERIAL connection type.

AETHER = 4

AETHER transport type.

AETHER_SERIAL = 7

AETHER_SERIAL transport type.

AETHER_TCPIP = 6

AETHER_TCPIP transport type.

AETHER_USB = 5

AETHER_USB transport type.

INVALID = 0

INVALID Undefined transport type.

SERIAL = 3

SERIAL transport type.

TCPIP = 2

TCPIP transport type.

USB = 1

USB transport type.

static `cca_spec_to_python_spec` (*cca_spec*)

Internal: Translate cffi spec into python Spec

class `brainstem.link.Status`

Status variables represent the link status possibilities for Brainstem Links.

Status States:

- STOPPED (0)
- INITIALIZING (1)
- RUNNING (2)
- STOPPING (3)
- SYNCING (4)
- INVALID_LINK_STREAM (5)
- IO_ERROR (6)
- UNKNOWN_ERROR (7)

class `brainstem.link.StreamStatusEntry` (*key, value*)

property key

A unique key made up of module, cmd, option, index, subindex

Type

unsigned long long (64bit)

property value

The Value associated with the key

Type

unsigned int (32bit)

class `brainstem.link.aEtherConfig`

aEther configuration class for configuring AETHER connection types.

Note: If `localOnly == false` AND `networkInterface` is default (0 or `LOCALHOST_IP_ADDRESS`) it will be populated with the auto-selected interface upon successful connection.

enabled

True: Client-Server model is used; False: Direct module control is used.

fallback

True: If connections fails it will automatically search for network connections.

localOnly

True: Restricts access to localhost; False: Expose device to external network.

assignedPort

Server assigned port after successful connection.

networkInterface

Network interface to use for connections.

3.2.8 Module

A module that provides base classes for BrainStem Modules and Entities.

The Module and Entity classes are designed to be extended for specific types of BrainStem Modules and Entities. For more information about Brainstem Modules and Entities, please see the [Terminology¹¹⁵](#) section of the [Acroname BrainStem Reference¹¹⁶](#)

class `brainstem.module.Module` (*address, enable_auto_networking=True, model=0*)

The Module Entity provides a generic interface to a BrainStem hardware module. The Module Class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

property address

Module address of the device

Type

unsigned byte

property bAutoNetworking

Return the current networking mode.

¹¹⁵ <https://acroname.com/reference/brainstem/terms.html>

¹¹⁶ <https://acroname.com/reference>

Type

bool

classQuantity (*command*)

Queries the module to determine how many entities of the specified class are implemented by the module. Zero is a valid return value. For example, calling classQuantity with the command parameter of cmdANALOG would return the number of analog entities implemented by the module.

Parameters

command (*unsigned byte*) – One of the UEI commands (cmdXXX).

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

connect (*transport, serial_number*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

connectFromSpec (*spec*)

Connect to a BrainStem module with a Spec.

Parameters

spec (*Spec*) – The specifier for the connection.

Returns

An error result from the list of defined error codes in brainstem.result

connectThroughLinkModule (*module*)

Connects to a Brainstem module on a BrainStem network, through the module given as an argument. The module passed in must have an active valid connection.

Parameters

module (*Module*) – The brainstem module to connect through.

Returns

An error result from the list of defined error codes in brainstem.result

disconnect ()

Disconnect from the Brainstem module.

discoverAndConnect (*transport, serial_number=0*)

Discover and connect from the Module level.

A discover-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 or None as the serial number will create a link to the first link module found on the specified transport.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.

- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in `brainstem.result`

entityGroup (*command, index*)

Queries the module the group assigned to an entity and index. Entities groups are used to specify when certain hardware features are fundamentally related. E.g. certain hardware modules may have some digital pins associated with an adjustable voltage rail; these digitals would be in the same group as the rail. Zero is the default group.

Parameters

command (*unsigned byte*) – One of the UEI commands (cmdXXX).

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

getBuild ()

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

getConfig ()

Gets the links current aEther configuration

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

Return type

Result containing a `aEtherConfig`

getModuleAddress ()

Get the address of the module object.

This method changes the local address of the module, not of the device. It is possible to get the module address of the device via `system.getModuleSoftwareOffset()`.

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

getStatus ()

Returns the status of the BrainStem connection See `brainstem.link.Status` for the possible states.

hasUEI (*command, option, index, flags*)

Queries the module to determine if it implements a UEI. Each UEI has a command, option or variant, index and flag. The `hasUEI` method queries for a fully specified UEI. Returns `aErrNone` if the variation is supported and an appropriate error if not. This call is blocking for up to the `nMSTimeout` period.

Parameters

- **command** (*unsigned byte*) – One of the UEI commands (cmdXXX).
- **option** (*unsigned byte*) – The option or variant of the command.
- **index** (*unsigned byte*) – The entity index.

- **flags** (*unsigned byte*) – The flags (ueiOPTION_SET or ueiOPTION_GET).

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

property id

A unique identifier of the associated module

Type

unsigned int

isConnected()

Returns true if the Module has an active connection or false otherwise

property link

return the current link or None.

Type

Link

property model

Model number of the device

Type

unsigned byte

reconnect()

Reconnect a lost connection to a Brainstem module.

setConfig(config)

Sets the links aEther configuration. Note: Configuration must be set BEFORE connection.

Parameters

config (*aEtherConfig*) – (aEtherConfig object): aEther configuration to be set.

Returns

An error result from the list of defined error codes in brainstem.result

setModuleAddress(address)

Set the address of the module object.

This method changes the local address of the module, not of the device. It is possible to set the module address of the device via system.setModuleSoftwareOffset().

Parameters

address (*unsigned byte*) – The module address to switch to for this module instance.

Returns

An error result from the list of defined error codes in brainstem.result

setNetworkingMode(mode)

Changes the networking mode of the stem object. Auto mode is enabled by default which allows automatic adjustment of the module/stems networking configuration. Refer to BrainStem Networking at www.acroname.com/support

Parameters

mode (*bool*) – Mode to be set. True = Auto; False = Manual

Returns

An error result from the list of defined error codes in brainstem.result

subClassQuantity (*command, index*)

Queries the module to determine how many subclass entities of the specified class are implemented by the module for a given entity index. This is used for entities which may be 2-dimensional. E.g. cmdMUX subclasses are the number of channels supported by a particular mux type (index); as a specific example, a module may support 4 UART channels, so subClassQuantity(cmdMUX, aMUX_UART...) could return 4. Zero is a valid return value.

Parameters

command (*unsigned byte*) – One of the UEI commands (cmdXXX).

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

3.2.9 PDChannelLogger

```
class brainstem.pd_channel_logger.BS_PD_Packet (channel=0, seconds=0, uSeconds=0,  
                                                direction=0, sop=0, event=0, payload=[],  
                                                ccChannel=0, crc=0)
```

Python representation of BS_PD_Packet_t (C structure)

- channel (uint8_t): Channel/Index
- seconds (uint8_t): Seconds in device time since power on.
- uSeconds (uint32_t): Micro Seconds in device time since power on.
- **direction (enumeration): Direction of packet transmission relative to the device.**
 - Invalid = 0
 - Transmit = 1
 - Receive = 2
 - Unknown = 3
- **sop (enumeration): See bs_pd_packet.h for more details**
 - SOP = 0
 - SOP' = 1
 - SOP'' = 2
 - Unknown = 3
- **event (enumeration): See powerdeliveryLogEvent in aProtocolDefs.h**
 - pdEventNone = 0
 - pdEventPacket = 1
 - pdEventConnect = 2
 - pdEventDisconnect = 3
 - pdEventCableResetReceived = 4
 - pdEventCableResetSent = 5
 - pdEventHardResetReceived = 6
 - pdEventHardResetSent = 7

- pdEventMessageTransmitFailed = 8 // No GoodCRC received
- pdEventMessageTransmitDiscarded = 9 // Incoming message detected so tx discarded
- pdEventPDFunctionDisabled = 10 // PD Stack is giving up on PD Comms
- pdEventVBUSEnabled = 11
- pdEventVBUSDisabled = 12
- pdEventVCONNEnabled = 13
- pdEventVCONNDisabled = 14
- pdEventRp1A5 = 15 // Used for Src Atomic Message Sequences
- pdEventRp3A0 = 16 // Used for Src Atomic Message Sequences
- pdEventBistEnter = 17
- pdEventBistExit = 18
- pdEventInvalidPacket = 19
- pdEventBistSharedCapacityEnter = 20
- pdEventBistSharedCapacityExit = 21
- pdEventLast = 22 // Should always be last!!

- payload (list): Raw PD Packet data

class `brainstem.pd_channel_logger.PDChannelLogger` (*module, index, buffer_length=1024*)

Manages BrainStem Power Delivery logging packets.

Parameters

- **module** (*Module*) – : Reference to an existing BrainStem Module
- **index** (*unsigned byte*) – Index/channel logging should be enabled for.
- **buffer_length** (*unsigned short*) – Number of packets the class should queue before dropping.

property `buffer_length`

Gets the buffer length

Returns

Buffer length of the associated object.

Return type

unsigned int

getPacket ()

Attempts to takes a packet from the internal buffer.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

getPackets (*buffer_length=100*)

Attempts to take a multiple packets (up to a maximum) from the internal buffer.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

property index

Gets the Index/Channel

Returns

Index/channel of the associated object.

Return type

unsigned byte

property module

Gets the Module object.

Returns

The associated module object.

Return type

Module

setEnabled (enabled)

Enables Power Delivery logging.

Parameters

enable (*bool*) – True enables logging; False disables logging

return: An error result from the list of defined error codes in `brainstem.result`

3.2.10 Results

A module that provides a result class for returning results of UEI commands.

Results consist of an error attribute and a value attribute. If the error attribute is set to `NO_ERROR`, then the result value is the response to the UEI command that was sent.

For more information about return values for commands and UEI's see the [Acroname BrainStem Reference](https://acroname.com/reference)¹¹⁷

class `brainstem.result.Result (error, *args, **kwargs)`

Result class for returning results of commands

Instances of Result represent the response to a command. The Result class also contains constants for the error codes that Brainstem APIs may return, for example `Result.aErrNone`, `result.aErrorMemory`, etc. Refer to the C API Error Codes for a complete list.

property error

Return the error attribute

static `getErrorDescription (error, buffer_length=256)`

Get the description of an error code.

Parameters

error (*int or Result object*) – The error to decode.

Returns

The error code in human readable form.

Return type

string

¹¹⁷ <https://acroname.com/reference>

static `getErrorText (error)`

Get the string representation of an error code.

Parameters

error (*int or Result object*) – The error to decode.

Returns

The error code in human readable form.

Return type

string

items ()

Return the key, value mapping for each result value

keys ()

Return a list of the names of each value in the result

property value

Return the value attribute(s)

values ()

Return a list of the values of each value in the result

3.2.11 Version

Provides version access utilities.

`brainstem.version.getMajor ()`

Gets the major revision number for the software package.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

`brainstem.version.getMinor ()`

Gets the minor revision number for the software package.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

`brainstem.version.getPatch ()`

Gets the patch revision number for the software package.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

`brainstem.version.get_version_string (packed_version=None, buffer_length=256)`

Gets the version string from a packed version.

Parameters

- **packed_version** (*unsigned int*) – If version is provided, it is unpacked and presented as the version string. Most useful for printing the firmware version currently installed on a module.
- **buffer_length** (*unsigned short*) – The amount of C memory to allocate

Returns

The library version as a string

Return type

str

`brainstem.version.isAtLeast (major, minor, patch)`

Check that the current software version is at least major.minor.patch

Parameters

- **major** (*const unsigned byte*) – The major revision level.
- **minor** (*const unsigned byte*) – The minor revision level.
- **patch** (*const unsigned byte*) – The patch revision level.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

`brainstem.version.isAtleastCompare (major_lhs, minor_lhs, patch_lhs, major_rhs, minor_rhs, patch_rhs)`

Check that the supplied left hand side (lhs) version is at least (\geq) the right hand side (rhs).

Parameters

- **major_lhs** (*const unsigned byte*) – The lhs major revision level.
- **minor_lhs** (*const unsigned byte*) – The lhs minor revision level.
- **patch_lhs** (*const unsigned byte*) – The lhs patch revision level.
- **major_rhs** (*const unsigned byte*) – The rhs major revision level.
- **minor_rhs** (*const unsigned byte*) – The rhs minor revision level.
- **patch_rhs** (*const unsigned byte*) – The rhs patch revision level.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

`brainstem.version.isLegacyFormat (packed_version)`

Check if the given build version is of the legacy packing format

Parameters

packed_version (*const unsigned int*) – The packed version number returned from `version.pack()` or `system.getVersion()`

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

Return type

Result

`brainstem.version.pack (major, minor, patch)`

Packs the given version into a single integer

Parameters

- **major** (*const unsigned byte*) – The major revision level.
- **minor** (*const unsigned byte*) – The minor revision level.
- **patch** (*const unsigned byte*) – The patch revision level.

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

Return type

Result

`brainstem.version.parseMajor (packed_version)`

Parses the major revision level from the given build number.

Parameters

packed_version (*const unsigned int*) – The packed version number returned from `version.pack()` or `system.getVersion()`

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

Return type

Result

`brainstem.version.parseMinor (packed_version)`

Parses the minor revision level from the given build number.

Parameters

packed_version (*const unsigned int*) – The packed version number returned from `version.pack()` or `system.getVersion()`

Returns

Result object containing the requested value when the results error is set to `NO_ERROR(0)`

Return type

Result

`brainstem.version.parsePatch (packed_version)`

Parses the revision patch level from the given build number.

Parameters

packed_version (*const unsigned int*) – The packed version number returned from `version.pack()` or `system.getVersion()`

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

`brainstem.version.unpack_version(packed_version)`

Unpacks a packed version.

Parameters

packed_version (*unsigned int*) – The packed version number.

Returns

Returns the library version as a 3-tuple (major, minor, patch)

Return type

str

3.2.12 Analog

See the *Analog Entity* for generic information.

class `brainstem.entity.Analog(module, index)`

Interface to analog entities on BrainStem modules. Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

getBulkCaptureNumberOfSamples()

Get the current number of samples setting for this analog when bulk capturing.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] number of samples.

Return type

brainstem.result.Result

getBulkCaptureSampleRate()

Get the current sample rate setting for this analog when bulk capturing.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] upon success filled with current sample rate in samples per second (Hertz).

Return type

brainstem.result.Result

getBulkCaptureState()

Get the current bulk capture state for this analog.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] the state of bulk capture. - Idle: bulkCaptureIdle = 0 - Pending: bulkCapturePending = 1 - Finished: bulkCaptureFinished = 2 - Error: bulkCaptureError = 3

Return type

brainstem.result.Result

getConfiguration()

Get the analog configuration.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Current configuration of the analog entity.

Return type

brainstem.result.Result

getEnable()

Get the analog output enable status.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 0 if disabled 1 if enabled.

Return type

brainstem.result.Result

getRange()

Get the analog input range.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] 8 bit value corresponding to a discrete range option

Return type

brainstem.result.Result

getValue()

Get the raw ADC output value in bits.

Note: Not all modules are provide 16 useful bits; this value's least significant bits are zero-padded to 16 bits. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned short] 16 bit analog reading with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

Return type

brainstem.result.Result

getVoltage()

Get the scaled micro volt value with reference to ground.

Note: Not all modules provide 32 bits of accuracy. Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Return type

brainstem.result.Result

initiateBulkCapture()

Initiate a BulkCapture on this analog. Captured measurements are stored in the module's RAM store (RAM_STORE) slot 0. Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

Note: When the bulk capture is complete getBulkCaptureState() will return either bulkCaptureFinished or bulkCaptureError.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setBulkCaptureNumberOfSamples (value)

Set the number of samples to capture for this analog when bulk capturing.

Parameters**value** (*unsigned int*) – number of samples. - Minimum # of Samples: 0 - Maximum # of Samples: (BRAINSTEM_RAM_SLOT_SIZE / 2) = (3FFF / 2) = 1FFF = 8191**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setBulkCaptureSampleRate (value)

Set the sample rate for this analog when bulk capturing.

Parameters**value** (*unsigned int*) – sample rate in samples per second (Hertz). - Minimum rate: 7,000 Hz - Maximum rate: 200,000 Hz**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setConfiguration (configuration)

Set the analog configuration.

Parameters**configuration** (*unsigned char*) – `bitAnalogConfigurationOutput` configures the analog entity as an output.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setEnabled (enable)

Set the analog output enable state.

Parameters**enable** (*bool*) – set 1 to enable or 0 to disable.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setRange (range)

Set the analog input range.

Parameters**range** (*unsigned char*) – 8 bit value corresponding to a discrete range option**Returns**An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setValue (*value*)

Set the value of an analog output (DAC) in bits.

Parameters**value** (*unsigned short*) – 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

Note: Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

ReturnsAn error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setVoltage (*microvolts*)

Set the voltage level of an analog output (DAC) in microvolts.

Parameters**microvolts** (*int*) – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Note: Voltage range is dependent on the specific DAC channel range.

ReturnsAn error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

3.2.13 App

See the [App Entity](#) for generic information.**class** `brainstem.entity.App` (*module, index*)

Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

execute (*app_param*)

Execute the app reflex on the module. Doesn't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

Parameters**app_param** (*unsigned int*) – The app parameter handed to the reflex.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

executeAndReturn (*app_param*, *ms_timeout=1000*)

Execute the app reflex on the module. Waits for a return from the reflex execution for `msTimeout` milliseconds. This method will block for up to `msTimeout`.

Parameters

- **app_param** (*unsigned int*) – The app parameter handed to the reflex.
- **ms_timeout** (*unsigned int*) – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The return value filled in from the result of executing the reflex routine.

Return type

brainstem.result.Result

3.2.14 Clock

See the *Clock Entity* for generic information.

class `brainstem.entity.Clock` (*module*, *index*)

Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

getDay ()

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The two digit day portion of the real-time clock value.

Return type

brainstem.result.Result

getHour ()

Get the two digit hour value (0-23).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The two digit hour portion of the real-time clock value.

Return type

brainstem.result.Result

getMinute()

Get the two digit minute value (0-59).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The two digit minute portion of the real-time clock value.

Return type

brainstem.result.Result

getMonth()

Get the two digit month value (1-12).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The two digit month portion of the real-time clock value.

Return type

brainstem.result.Result

getSecond()

Get the two digit second value (0-59).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The two digit second portion of the real-time clock value.

Return type

brainstem.result.Result

getYear()

Get the four digit year value (0-4095).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned short] Get the year portion of the real-time clock value.

Return type

brainstem.result.Result

setDay (day)

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Parameters

day (*unsigned char*) – The two digit day portion of the real-time clock value.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setHour (hour)

Set the two digit hour value (0-23).

Parameters

hour (*unsigned char*) – The two digit hour portion of the real-time clock value.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setMinute (minute)

Set the two digit minute value (0-59).

Parameters

minute (*unsigned char*) – The two digit minute portion of the real-time clock value.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setMonth (month)

Set the two digit month value (1-12).

Parameters

month (*unsigned char*) – The two digit month portion of the real-time clock value.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setSecond (second)

Set the two digit second value (0-59).

Parameters

second (*unsigned char*) – The two digit second portion of the real-time clock value.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setYear (*year*)

Set the four digit year value (0-4095).

Parameters**year** (*unsigned short*) – Set the year portion of the real-time clock value.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

3.2.15 Digital

See the [Digital Entity](#) for generic information.**class** `brainstem.entity.Digital` (*module, index*)

Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

getConfiguration ()

Get the digital configuration.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Current configuration of the digital entity.

Return type[brainstem.result.Result](#)**getLinkChannel** ()

Get the link channel (entity index) for linking digital entities.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current link channel (entity index) value.

Return type[brainstem.result.Result](#)**getState** ()

Get the state.

Note: If in high Z state an error will be returned.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[bool] The current state of the digital entity. 0 is logic low, 1 is logic high.

Return type*brainstem.result.Result***getStateAll()**

Gets the logical state of all available digitals in a bit mapped representation. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.

Return type*brainstem.result.Result***getValue()**

Get the expected value of the digital pin.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The expected value of the digital pin. 0 is logic low, 1 is logic high.

Return type*brainstem.result.Result***setConfiguration(configuration)**

Set the digital configuration to one of the available 5 states.

Parameters**configuration** (*unsigned char*) -**The configuration to be applied**

- Digital Input: digitalConfigurationInput = 0
- Digital Output: digitalConfigurationOutput = 1
- RCServo Input: digitalConfigurationRCServoInput = 2
- RCServo Output: digitalConfigurationRCServoOutput = 3
- High Z State: digitalConfigurationHiZ = 4

- Digital Input: digitalConfigurationInputPullUp = 0
- Digital Input: digitalConfigurationInputNoPull = 4
- Digital Input: digitalConfigurationInputPullDown = 5

Note: Some configurations are only supported on specific pins.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setLinkChannel (*link_channel*)

Set the link channel (entity index) for linking digital entities. Two digital entities will link when one is configured as LinkInput, one as LinkOutput, and both have each others linkChannel indexes.

Parameters

link_channel (*unsigned char*) – The link channel (entity index) value. Entities with matching linkChannel values can be linked.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setState (*state*)

Set the logical state.

Parameters

state (*bool*) – The state to be set. 0 is logic low, 1 is logic high.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setStateAll (*state*)

Sets the logical state of all available digitals based on the bit mapping. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

Parameters

state (*unsigned int*) – The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setValue (*value*)

Set the expected value of the digital pin.

Parameters

value (*unsigned char*) – The expected value of the digital pin. 0 is logic low, 1 is logic high.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.16 Equalizer

See the [Equalizer Entity](#) for generic information.

class `brainstem.entity.Equalizer (module, index)`

Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

getReceiverConfig (channel)

Gets the receiver configuration for a given channel.

Parameters

channel (*unsigned char*) – The equalizer receiver channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Configuration of the receiver.

Return type

[brainstem.result.Result](#)

getTransmitterConfig ()

Gets the transmitter configuration

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Configuration of the Transmitter.

Return type

[brainstem.result.Result](#)

setReceiverConfig (channel, config)

Sets the receiver configuration for a given channel.

Parameters

- **channel** (*unsigned char*) – The equalizer receiver channel.
- **config** (*unsigned char*) – Configuration to be applied to the receiver.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setTransmitterConfig (*config*)

Sets the transmitter configuration

Parameters

config (*unsigned char*) – Configuration to be applied to the transmitter.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.17 Ethernet

See the [Ethernet Entity](#) for generic information.

class `brainstem.entity.Ethernet` (*module, index*)

IP configuration. MAC info. BrainD port.

getEnabled ()

Gets the current enable value of the Ethernet interface.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Fully enabled network connectivity; 0 = Ethernet MAC is disabled.

Return type

[brainstem.result.Result](#)

getHostname (*buffer_length=65536*)

Get hostname that's requested when this device sends a DHCP request.

Parameters

buffer_length (*unsigned int*) – N, for N bytes.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[string] alias to an array of `uint8_t[N]`

Return type

[brainstem.result.Result](#)

getIPv4Address (*buffer_length=65536*)

Get the effective IP address of this device.

Parameters

buffer_length (*unsigned int*) – size of buffer. Should be 4.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[4] for returned output

Return type

brainstem.result.Result

getIPv4DNSAddress (*buffer_length=65536*)

Get effective IPv4 DNS addresses, for the current NetworkConfiguration

Parameters

buffer_length (*unsigned int*) – Maximum length of array, in bytes.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[N][4]

Return type

brainstem.result.Result

getIPv4Gateway (*buffer_length=65536*)

Get the effective IP gateway of this device.

Parameters

buffer_length (*unsigned int*) – size of buffer. Should be 4.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[4] for returned output

Return type

brainstem.result.Result

getIPv4Netmask (*buffer_length=65536*)

Get the effective IP netmask of this device.

Parameters

buffer_length (*unsigned int*) – size of buffer. Should be 4.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[4] for returned output

Return type*brainstem.result.Result***getInterfacePort (service)**

Get the port of a TCPIP service on the device.

Parameters

service (*unsigned char*) – The index of the service to get the port for.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned short] The port of the TCPIP server.

Return type*brainstem.result.Result***getMACAddress (buffer_length=65536)**

Get the MAC address of the Ethernet interface.

Parameters

buffer_length (*unsigned int*) – length of buffer that's writeable, should be > 6.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[6]

Return type*brainstem.result.Result***getNetworkConfiguration ()**

Get the method in which IP Address is assigned to this device

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Method used. Current methods

- NONE = 0
- STATIC = 1
- DHCP = 2

Return type*brainstem.result.Result*

getStaticIPv4Address (*buffer_length=65536*)

Get the expected IPv4 address of this device, when networkConfiguration == STATIC

Parameters

buffer_length (*unsigned int*) – size of buffer. Should be 4.

Note: The functional IPv4 address of The Module will differ if NetworkConfiguration != STATIC.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[4] for returned output

Return type

brainstem.result.Result

getStaticIPv4DNSAddress (*buffer_length=65536*)

Get IPv4 DNS addresses (plural), when NetworkConfiguration == STATIC

Parameters

buffer_length (*unsigned int*) – Maximum length of array, in bytes.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[N][4]

Return type

brainstem.result.Result

getStaticIPv4Gateway (*buffer_length=65536*)

Get the expected IPv4 gateway of this device, when networkConfiguration == STATIC

Parameters

buffer_length (*unsigned int*) – size of buffer. Should be 4.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[4] for returned output

Return type

brainstem.result.Result

getStaticIPv4Netmask (*buffer_length=65536*)

Get the expected IPv4 netmask of this device, when networkConfiguration == STATIC

Parameters

buffer_length (*unsigned int*) – size of buffer. Should be 4.

Note: The functional IPv4 netmask of The Module will differ if NetworkConfiguration != STATIC.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] alias to an array of uint8_t[4] for returned output

Return type

brainstem.result.Result

setEnabled (*enabled*)

Sets the Ethernet's interface to enabled/disabled.

Parameters

enabled (*bool*) – 1 = enabled; 0 = disabled

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setHostname (*buffer*)

Set hostname that's requested when this device sends a DHCP request.

Parameters

buffer (*string*) – alias to an array of uint8_t[N]

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setInterfacePort (*service, port*)

Set the port of a TCPIP service on the device.

Parameters

- **service** (*unsigned char*) – The index of the service to set the port for.
- **port** (*unsigned short*) – The port to be used for the TCPIP server.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setNetworkConfiguration (*address_style*)

Get the method in which IP Address is assigned to this device

Parameters

address_style (*unsigned char*) – Method to use. See getNetworkConfiguration for addressStyle enumerations.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setStaticIPv4Address (*buffer*)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC.

Parameters

buffer (*list(unsigned char)*) – alias to an array of uint8_t[4] with an IP address

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setStaticIPv4DNSAddress (*buffer*)

Set IPv4 DNS Addresses (plural), if NetworkConfiguration == STATIC

Parameters

buffer (*list(unsigned char)*) – alias to an array of uint8_t[N][4]

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setStaticIPv4Gateway (*buffer*)

Set the desired IPv4 gateway of this device, if NetworkConfiguration == STATIC

Parameters

buffer (*list(unsigned char)*) – alias to an array of uint8_t[4] with an IP address

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setStaticIPv4Netmask (*buffer*)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC

Parameters

buffer (*list(unsigned char)*) – alias to an array of uint8_t[4] with an IP address

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

3.2.18 HDBaseT

See the [HDBaseT Entity](#) for generic information.

class `brainstem.entity.HDBaseT (module, index)`

This entity is only available on certain modules, and provides information on HDBaseT extenders.

getCableLength ()

Gets the perceived cable length

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Cable length in micro-meters

Return type

[brainstem.result.Result](#)

getEncodingState ()

Gets the current encoding state.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Signal modulation encoding type.

Return type

[brainstem.result.Result](#)

getFirmwareVersion ()

Gets the firmware version of the HDBaseT device

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] A bit packet representation of the firmware version Major: Bits 24-31; Minor: Bits 16-23; Patch: Bits 8-15; Build: Bits 0-7

Return type

[brainstem.result.Result](#)

getLinkRole ()

Gets the current link role In the case of “Auto” the getState API will provide the current role.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Link role

Return type*brainstem.result.Result***getLinkUtilization()**

Gets the current link utilization

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Utilization in milli-percent

Return type*brainstem.result.Result***getMSEA()**

Gets the Mean Squared Error (MSE) for channel A

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The current MSE for channel A in micro-dB

Return type*brainstem.result.Result***getMSEB()**

Gets the Mean Squared Error (MSE) for channel B

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The current MSE for channel B in micro-dB

Return type*brainstem.result.Result***getRetransmissionRate()**

Gets the number of successful messages between retransmission

Returns**Object containing error code and returned value on success.**

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Instantaneous number of successful messages between retransmission. To be interpreted as: $1 / \text{retransmissionRate}$ for rate interpretation. If the value is 0, there have been no retransmissions, otherwise higher is better..

Return type

brainstem.result.Result

getSerialNumber (*buffer_length=65536*)

Gets the serial number of the HDBaseT device (6 bytes)

Parameters

buffer_length (*unsigned int*) - Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getState ()

Gets the current state of the HDBaseT link

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bit packeted representation of the state.

Return type

brainstem.result.Result

getUSB2DeviceTree (*buffer_length=65536*)

Gets the USB2 tree at the HDBaseT device.

Parameters

buffer_length (*unsigned int*) - Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getUSB3DeviceTree (*buffer_length=65536*)

Gets the USB3 tree at the HDBaseT device.

Parameters

buffer_length (*unsigned int*) – Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

setLinkRole (*role*)

Sets the active link role

Parameters

role (*unsigned char*) – The role to be set.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

3.2.19 I2C

See the *I2C Entity* for generic information.

class *brainstem.entity.I2C* (*module, index*)

Interface the I2C buses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entities bus.

getSpeed ()

Get I2C bus speed.

This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values:

1 - 100Khz 2 - 400Khz 3 - 1MHz

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The speed setting value.

Return type

brainstem.result.Result

read (*address*, *read_length*)

Read from a device on this I2C bus.

Parameters

- **address** (*unsigned char*) – The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **read_length** (*unsigned char*) – The length of the data to read in bytes.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAINSTEM_MAXPACKETBYTES - 5

Return type

brainstem.result.Result

setPullup (*enable*)

Set bus pull-up state. This call only works with stems that have software controlled pull-ups. Check the datasheet for more information. This parameter is saved when system.save is called.

Parameters

enable (*bool*) – true enables pull-ups false disables them.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setSpeed (*speed*)

Set I2C bus speed.

This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values:

1 - 100Khz 2 - 400Khz 3 - 1MHz

Parameters

speed (*unsigned char*) – The speed setting value.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

write (*address*, *buffer*)

Write to a device on this I2C bus.

Parameters

- **address** (*unsigned char*) – The I2C address (7bit <XXXX-XXX0>) of the device to write.

- **buffer** (*unsigned char*) – The data to send to the device This array should be no larger than `ABRAINSTEM_MAXPACKETBYTES - 5`

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.20 Mux

See the [Mux Entity](#) for generic information.

class `brainstem.entity.Mux (module, index)`

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs. One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs; some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

getChannel ()

Get the current selected mux channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Indicates which channel is selected.

Return type

[brainstem.result.Result](#)

getChannelVoltage (channel)

Get the voltage of the indicated mux channel.

Parameters

channel (*unsigned char*) – The channel in which voltage was requested.

Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Return type*brainstem.result.Result***getConfiguration()**

Get the configuration of the mux.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] integer representing the mux configuration either default, or split-mode.

Return type*brainstem.result.Result***getEnable()**

Get the mux enable/disable status

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[bool] true: mux is enabled, false: the mux is disabled.

Return type*brainstem.result.Result***getSplitMode()**

Get the current split mode mux configuration.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Return type*brainstem.result.Result***setChannel(channel)**

Set the current mux channel.

Parameters**channel** (*unsigned char*) – mux channel to select.**Returns**An error result from the list of defined error codes in *brainstem.result.Result***Return type**

unsigned byte

setConfiguration (config)

Set the configuration of the mux.

Parameters

config (*int*) – integer representing the mux configuration either muxConfig_default, or muxConfig_splitMode.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setEnabled (enable)

Enable the mux.

Parameters

enable (*bool*) – true: enables the mux for the selected channel.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setSplitMode (split_mode)

Sets the mux's split mode configuration.

Parameters

split_mode (*int*) – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

3.2.21 PoE

See the [PoE Entity](#) for generic information.

class brainstem.entity.**PoE** (*module, index*)

This entity is only available on certain modules, and provides a Power over Ethernet control ability.

getPairAccumulatedPower (pair)

Gets the accumulated power for a given pair.

Parameters

pair (*unsigned char*) –

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] Variable to be filled with the total accumulated POE power in milli-watts (mW).

Return type

brainstem.result.Result

getPairCapacitance (*pair*)

Gets the Capacitance for a given pair

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The capacitance in nanofarads (1 == 1e-9F).

Return type

brainstem.result.Result

getPairCurrent (*pair*)

Gets the Current for a given pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The current in microamps (1 == 1e-6V).

Return type

brainstem.result.Result

getPairDetectionStatus (*pair*)

Gets detected status of the POE connection for a given pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current detected status of the pairs.

Return type

brainstem.result.Result

getPairDiscoveredClass (*pair*)

Gets the discovered class for a given pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The negotiated POE class by the device (PSE/PD).

Return type

brainstem.result.Result

getPairEnabled (*pair*)

Gets the current enable value of the indicated POE pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Enabled; 0 = Disabled;

Return type

brainstem.result.Result

getPairPower (*pair*)

Get the instantaneous power consumption for a given pair The equivalent of Voltage x Current

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] Variable to be filled with the pairs power in milli-watts (mW).

Return type

brainstem.result.Result

getPairRequestedClass (*pair*)

Gets the requested class for a given pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The requested POE class by the device (PD).

Return type

brainstem.result.Result

getPairResistance (*pair*)

Gets the Resistance for a given pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The resistance in milliohms (1 == 1e-3Z).

Return type

brainstem.result.Result

getPairSourcingClass (pair)

Gets the sourcing class for a given pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The POE class being offered by the device (PSE).

Return type

brainstem.result.Result

getPairVoltage (pair)

Gets the Voltage for a given pair.

Parameters

pair (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The voltage in microvolts (1 == 1e-6V).

Return type

brainstem.result.Result

getPowerMode ()

Gets the power mode of the device

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The power mode (PD, PSE, Auto, Off).

Return type

brainstem.result.Result

getPowerState()

Gets the power state of the device

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The power state (PD, PSE, Off).

Return type

brainstem.result.Result

getTotalAccumulatedPower()

Gets the total Accumulated Power

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] Variable to be filled with the total accumulated POE power in milli-watts (mW).

Return type

brainstem.result.Result

getTotalPower()

Gets the total instantaneous power consumption The equivalent of Pair1(Voltage x Current) + Pair2(Voltage x Current)

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] Variable to be filled with the total POE power in milli-watts (mW).

Return type

brainstem.result.Result

setPairAccumulatedPower(pair, power)

Sets the accumulated power for a given pair.

Parameters

- **pair** (*unsigned char*) -

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

- **power** (*int*) – The power accumulator value to be set in milli-watts (mW).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPairEnabled (*pair, enable*)

Enables or disables the indicated POE pair.

Parameters

- **pair** (*unsigned char*) –

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

- **enable** (*bool*) – 1 = Enable port; 0 = Disable port.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPairSourcingClass (*pair, value*)

Sets the sourcing class for a given pair.

Parameters

- **pair** (*unsigned char*) –

Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4

- **value** (*unsigned char*) – The POE class being offered by the device (PSE).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerMode (*value*)

Sets the power mode of the device

Parameters

value (*unsigned char*) – The power mode (PD, PSE, Auto, Off).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setTotalAccumulatedPower (*power*)

Sets the total accumulated power

Parameters

power (*int*) – The power accumulator value to be set in milli-watts (mW).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.22 Pointer

See the [Pointer Entity](#) for generic information.

class `brainstem.entity.Pointer` (*module, index*)

Allows access to the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the BrainStem scratchpad from the host, and provides a mechanism for allowing the host application and BrainStem relexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via `setOffset`. A read of a character short or int can be made from that cursor position.

In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

getChar ()

Get a char (1 byte) value from the pointer at this object's index, where elements are 1 byte long.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The value of a single character (1 byte) stored in the pointer.

Return type

[brainstem.result.Result](#)

getInt ()

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The value of a single int (4 byte) stored in the pointer.

Return type

[brainstem.result.Result](#)

getMode ()

Get the mode of the pointer

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Return type

brainstem.result.Result

getOffset ()

Get the offset of the pointer

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned short] The value of the offset.

Return type

brainstem.result.Result

getShort ()

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned short] The value of a single short (2 byte) stored in the pointer.

Return type

brainstem.result.Result

getTransferStore ()

Get the handle to the store.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The handle of the store.

Return type

brainstem.result.Result

initiateTransferFromStore (*transfer_length*)

Transfer data from the store.

Parameters

transfer_length (*unsigned char*) – The length of the data transfer.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

initiateTransferToStore (*transfer_length*)

Transfer data to the store.

Parameters

transfer_length (*unsigned char*) – The length of the data transfer.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setChar (*value*)

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

Parameters

value (*unsigned char*) – The single char (1 byte) value to be stored in the pointer.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setInt (*value*)

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

Parameters

value (*unsigned int*) – The single int (4 byte) value to be stored in the pointer.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setMode (*mode*)

Set the mode of the pointer

Parameters

mode (*unsigned char*) – The mode: `aPOINTER_MODE_STATIC` or `aPOINTER_MODE_AUTO_INCREMENT`.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setOffset (*offset*)

Set the offset of the pointer

Parameters

offset (*unsigned short*) – The value of the offset.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setShort (*value*)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

Parameters

value (*unsigned short*) – The single short (2 byte) value to be set in the pointer.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setTransferStore (*handle*)

Set the handle to the store.

Parameters

handle (*unsigned char*) – The handle of the store.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.23 Port

See the [Port Entity](#) for generic information.

class `brainstem.entity.Port` (*module, index*)

The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

getAllocatedPower ()

Gets the currently allocated power This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] Variable to be filled with the allocated power in milli-watts (mW).

Return type*brainstem.result.Result***getAvailablePower()**

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the available power in milli-watts (mW).

Return type*brainstem.result.Result***getCC1AccumulatedPower()**

Gets the CC1 Accumulated Power

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The accumulated power on Vconn in milliwatt-hours.

Return type*brainstem.result.Result***getCC1Current()**

Get the current through the CC1 for a port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel current in micro-amps (1 == 1e-6A).

Return type*brainstem.result.Result***getCC1Enabled()**

Gets the current enable value of the CC1 lines. Sub-component of getCCEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = CC1 enabled; 0 = CC1 disabled.

Return type*brainstem.result.Result***getCC1State()**

Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned short] Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:

- None = 0 = portCC1State_None
- Invalid = 1 = portCC1State_Invalid
- Rp (default) = 2 = portCC1State_RpDefault
- Rp (1.5A) = 3 = portCC1State_Rp1p5
- Rp (3A) = 4 = portCC1State_Rp3p0
- Rd = 5 = portCC1State_Rd
- Ra = 6 = portCC1State_Ra
- Managed by controller = 7 = portCC1State_Managed
- Unknown = 8 = portCC1State_Unknown

Return type*brainstem.result.Result***getCC1Voltage()**

Get the voltage of CC1 for a port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type*brainstem.result.Result***getCC2AccumulatedPower()**

Gets the CC2 Accumulated Power

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The accumulated power on Vconn in milliwatt-hours.

Return type

brainstem.result.Result

getCC2Current ()

Get the current through the CC2 for a port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel current in micro-amps (1 == 1e-6A).

Return type

brainstem.result.Result

getCC2Enabled ()

Gets the current enable value of the CC2 lines. Sub-component of getCCEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = CC2 enabled; 0 = CC2 disabled.

Return type

brainstem.result.Result

getCC2State ()

Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned short] Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:

- None = 0 = portCC2State_None
- Invalid = 1 = portCC2State_Invalid
- Rp (default) = 2 = portCC2State_RpDefault
- Rp (1.5A) = 3 = portCC2State_Rp1p5
- Rp (3A) = 4 = portCC2State_Rp3p0
- Rd = 5 = portCC2State_Rd

- Ra = 6 = portCC2State_Ra
- Managed by controller = 7 = portCC2State_Managed
- Unknown = 8 = portCC2State_Unknown

Return type*brainstem.result.Result***getCC2Voltage()**

Get the voltage of CC2 for a port.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type*brainstem.result.Result***getCCCurrentLimit()**

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated representation of the CC Current limit.

- 0 = None
- 1 = Default (500/900mA)
- 2 = 1.5A
- 3 = 3.0A

Return type*brainstem.result.Result***getCCEnabled()**

Gets the current enable value of the CC lines. Sub-component (CC) of getEnabled.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = CC enabled; 0 = CC disabled.

Return type*brainstem.result.Result*

getCurrentLimit()

Gets the current limit of the port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the limit in microAmps (uA).

Return type

brainstem.result.Result

getCurrentLimitMode()

Gets the current limit mode. The mode determines how the port will react to an over current condition.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Return type

brainstem.result.Result

getDataEnabled()

Gets the current enable value of the data lines. Sub-component (Data) of getEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Data enabled; 0 = Data disabled.

Return type

brainstem.result.Result

getDataHS1Enabled()

Gets the current enable value of the High Speed A side (HSA) data lines. Sub-component of getDataHSEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Data enabled; 0 = Data disabled.

Return type*brainstem.result.Result***getDataHS2Enabled()**

Gets the current enable value of the High Speed B side (HSB) data lines. Sub-component of getDataHSEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Data enabled; 0 = Data disabled.

Return type*brainstem.result.Result***getDataHSEnabled()**

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Data enabled; 0 = Data disabled.

Return type*brainstem.result.Result***getDataHSRoutingBehavior()**

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Return type*brainstem.result.Result***getDataRole()**

Gets the Port Data Role.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The data role to be set. See datasheet for details.

Return type

brainstem.result.Result

getDataSS1Enabled()

Gets the current enable value of the Super Speed A side (SSA) data lines. Sub-component of getDataSSEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Data enabled; 0 = Data disabled.

Return type

brainstem.result.Result

getDataSS2Enabled()

Gets the current enable value of the Super Speed B side (SSB) data lines. Sub-component of getDataSSEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Data enabled; 0 = Data disabled.

Return type

brainstem.result.Result

getDataSSEnabled()

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Data enabled; 0 = Data disabled.

Return type

brainstem.result.Result

getDataSSRoutingBehavior()

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Return type*brainstem.result.Result***getDataSpeed ()**

Gets the speed of the enumerated device.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Bit mapped value representing the devices speed. See “Devices” reference for details.

Return type*brainstem.result.Result***getEnabled ()**

Gets the current enable value of the port.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Fully enabled port; 0 = One or more disabled components.

Return type*brainstem.result.Result***getErrors ()**

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bit mapped field representing the current errors of the ports

Return type*brainstem.result.Result*

getHSBoost ()

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.

Return type

brainstem.result.Result

getMode ()

Gets current mode of the port

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bit mapped value representing the ports mode. See “Devices” reference for details.

Return type

brainstem.result.Result

getName (buffer_length=65536)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

buffer_length (*unsigned int*) - Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[string] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getPowerEnabled ()

Gets the current enable value of the power lines. Sub-component (Power) of getEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Power enabled; 0 = Power disabled.

Return type

brainstem.result.Result

getPowerLimit()

Gets the user defined power limit for the port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the power limit in milli-watts (mW).

Return type

brainstem.result.Result

getPowerLimitMode()

Gets the power limit mode. The mode determines how the port will react to an over power condition.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.

Return type

brainstem.result.Result

getPowerMode()

Gets the Port Power Mode: Convenience Function of get/setPortMode

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current power mode.

Return type

brainstem.result.Result

getSBU1Voltage()

Get the voltage of SBU1 for a port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type

brainstem.result.Result

getSBU2Voltage ()

Get the voltage of SBU2 for a port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type

brainstem.result.Result

getState ()

A bit mapped representation of the current state of the port. Reflects what the port IS which may differ from what was requested.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the current state.

Return type

brainstem.result.Result

getVbusAccumulatedPower ()

Gets the Vbus Accumulated Power

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The accumulated power on Vbus in milliwatt-hours.

Return type

brainstem.result.Result

getVbusCurrent ()

Gets the Vbus Current

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The current in microamps ($1 == 1e-6A$) currently present on Vbus.

Return type

brainstem.result.Result

getVbusVoltage()

Gets the Vbus Voltage

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The voltage in microvolts ($1 == 1e-6V$) currently present on Vbus.

Return type

brainstem.result.Result

getVconn1Enabled()

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Vconn1 enabled; 0 = Vconn1 disabled.

Return type

brainstem.result.Result

getVconn2Enabled()

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Vconn2 enabled; 0 = Vconn2 disabled.

Return type

brainstem.result.Result

getVconnAccumulatedPower()

Gets the Vconn Accumulated Power

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The accumulated power on Vconn in milliwatt-hours.

Return type

brainstem.result.Result

getVconnCurrent ()

Gets the Vconn Current

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The current in microamps (1 == 1e-6A) currently present on Vconn.

Return type

brainstem.result.Result

getVconnEnabled ()

Gets the current enable value of the Vconn lines. Sub-component (Vconn) of getEnabled.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] 1 = Vconn enabled; 0 = Vconn disabled.

Return type

brainstem.result.Result

getVconnVoltage ()

Gets the Vconn Voltage

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The voltage in microvolts (1 == 1e-6V) currently present on Vconn.

Return type

brainstem.result.Result

getVoltageSetpoint ()

Gets the current voltage setpoint value for the port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] the voltage setpoint of the port in uV.

Return type

brainstem.result.Result

resetVbusAccumulatedPower ()

Resets the Vbus Accumulated Power to zero.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

resetVconnAccumulatedPower ()

Resets the Vconn Accumulated Power to zero.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setCC1AccumulatedPower (milliwatthours)

Sets the CC1 Accumulated Power

Parameters

milliwatthours (*int*) – The accumulated power on Vconn to be set.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setCC1Enabled (enable)

Enables or disables the CC1 lines. Sub-component of setCCEnabled.

Parameters

enable (*bool*) – 1 = Enable CC1 lines; 0 = Disable CC1 lines.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setCC2AccumulatedPower (milliwatthours)

Sets the CC2 Accumulated Power

Parameters

milliwatthours (*int*) – The accumulated power on Vconn to be set.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setCC2Enabled (*enable*)

Enables or disables the CC2 lines. Sub-component of setCCEnabled.

Parameters

enable (*bool*) – 1 = Enable CC2 lines; 0 = Disable CC2 lines.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setCCCurrentLimit (*value*)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

Parameters

value (*unsigned char*) –

Variable to be filled with an enumerated representation of the CC Current limit.

- 0 = None
- 1 = Default (500/900mA)
- 2 = 1.5A
- 3 = 3.0A

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setCCEnabled (*enable*)

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

Parameters

enable (*bool*) – 1 = Enable CC lines; 0 = Disable CC lines.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setCurrentLimit (*limit*)

Sets the current limit of the port.

Parameters

limit (*unsigned int*) – Current limit to be applied in microAmps (uA).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setCurrentLimitMode (*mode*)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters

mode (*unsigned char*) – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataEnabled (enable)

Enables or disables the data lines. Sub-component (Data) of `setEnabled`.

Parameters

enable (*bool*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataHS1Enabled (enable)

Enables or disables the High Speed A side (HSA) data lines. Sub-component of `setDataHSEnabled`.

Parameters

enable (*bool*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataHS2Enabled (enable)

Enables or disables the High Speed B side (HSB) data lines. Sub-component of `setDataHSEnabled`.

Parameters

enable (*bool*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataHSEnabled (enable)

Enables or disables the High Speed (HS) data lines. Sub-component of `setDataEnabled`.

Parameters

enable (*bool*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataHSRoutingBehavior (mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode (*unsigned char*) – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataSS1Enabled (*enable*)

Enables or disables the Super Speed A side (SSA) data lines. Sub-component of `setDataEnabled`.

Parameters

enable (*bool*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataSS2Enabled (*enable*)

Enables or disables the Super Speed B side (SSB) data lines. Sub-component of `setDataSSEnabled`.

Parameters

enable (*bool*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataSSEnabled (*enable*)

Enables or disables the Super Speed (SS) data lines. Sub-component of `setDataEnabled`.

Parameters

enable (*bool*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataSSRoutingBehavior (*mode*)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode (*unsigned char*) – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setEnabled (enable)

Enables or disables the entire port.

Parameters

enable (*bool*) – 1 = Fully enable port; 0 = Fully disable port.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setHSBoost (boost)

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Parameters

boost (*unsigned char*) – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setMode (mode)

Sets the mode of the port

Parameters

mode (*unsigned int*) – Port mode to be set. See “Devices” documentation for details.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setName (buffer)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

buffer (*string*) – Pointer to the start of a c style buffer to be transferred.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerEnabled (enable)

Enables or Disables the power lines. Sub-component (Power) of `setEnabled`.

Parameters

enable (*bool*) – 1 = Enable power; 0 = Disable disable.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerLimit (*limit*)

Sets a user defined power limit for the port.

Parameters

limit (*unsigned int*) – Power limit to be applied in milli-watts (mW).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerLimitMode (*mode*)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

mode (*unsigned char*) – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerMode (*power_mode*)

Sets the Port Power Mode: Convenience Function of `get/setPortMode`

Parameters

power_mode (*unsigned char*) – The power mode to be set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVbusAccumulatedPower (*milliwatthours*)

Sets the Vbus Accumulated Power

Parameters

milliwatthours (*int*) – The accumulated power on Vbus in milliwatt-hours to be set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVconn1Enabled (*enable*)

Enables or disables the Vconn1 lines. Sub-component of `setVconnEnabled`.

Parameters

enable (*bool*) – 1 = Enable Vconn1 lines; 0 = Disable Vconn1 lines.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVconn2Enabled (*enable*)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

Parameters

enable (*bool*) – 1 = Enable Vconn2 lines; 0 = Disable Vconn2 lines.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVconnAccumulatedPower (*milliwatthours*)

Sets the Vconn Accumulated Power

Parameters

milliwatthours (*int*) – The accumulated power on Vconn to be set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVconnEnabled (*enable*)

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

Parameters

enable (*bool*) – 1 = Enable Vconn lines; 0 = Disable Vconn lines.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVoltageSetpoint (*value*)

Sets the current voltage setpoint value for the port.

Parameters

value (*unsigned int*) – the voltage setpoint of the port in uV.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.24 PowerDelivery

See the [PowerDelivery Entity](#) for generic information.

class `brainstem.entity.PowerDelivery` (*module, index*)

Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface. This Entity will allow you to directly access the vast landscape of PD.

getAttachTimeElapsed (*buffer_length=16384*)

Gets the length of time that the port has been in the attached state. Returned as a list of two unsigned integers, first seconds, then microseconds.

Parameters

buffer_length (*unsigned int*) – Length of the buffer to be filed

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned int)] Pointer to list of unsigned integers to fill with attach time elapsed

Return type

brainstem.result.Result

getCableCurrentMax ()

Gets the maximum current capability report by the e-mark of the attached cable.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filled with an enumerated representation of current.

- Unknown/Unattached (0)
- 3 Amps (1)
- 5 Amps (2)

Return type

brainstem.result.Result

getCableOrientation ()

Gets the current orientation being used for PD communication

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable filled with an enumeration of the orientation.

- Unconnected (0)
- CC1 (1)
- CC2 (2)

Return type

brainstem.result.Result

getCableSpeedMax()

Gets the maximum data rate capability reported by the e-mark of the attached cable.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filled with an enumerated representation of data speed.

- Unknown/Unattached (0)
- USB 2.0 (1)
- USB 3.2 gen 1 (2)
- USB 3.2 / USB 4 gen 2 (3)
- USB 4 gen 3 (4)

Return type

brainstem.result.Result

getCableType()

Gets the cable type reported by the e-mark of the attached cable.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filled with an enumerated representation of the cable type.

- Invalid, no e-mark and not Vconn powered (0)
- Passive cable with e-mark (1)
- Active cable (2)

Return type

brainstem.result.Result

getCableVoltageMax()

Gets the maximum voltage capability reported by the e-mark of the attached cable.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filled with an enumerated representation of voltage.

- Unknown/Unattached (0)
- 20 Volts DC (1)
- 30 Volts DC (2)
- 40 Volts DC (3)
- 50 Volts DC (4)

Return type*brainstem.result.Result***getConnectionState ()**

Gets the current state of the connection in the form of an enumeration.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Pointer to be filled with the current connection state.

Return type*brainstem.result.Result***getDataRoleCapabilities ()**

Gets the data roles that may be advertised by the local partner.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filled with the data role

- None = 0 = pdDataRoleCapabilities_None
- DFP = 1 = pdDataRoleCapabilities_DFP
- UFP = 2 = pdDataRoleCapabilities_UFP
- DFP/UFP = 3 = pdDataRoleCapabilities_DualRole (Dual Role Port)

Return type*brainstem.result.Result***getFastRoleSwapCurrent ()**

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

An enumerated value referring to current swap value.

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

Return type*brainstem.result.Result***getFlagMode (flag)**

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Parameters**flag** (*unsigned char*) – Flag/Advertisement to be modified**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filled with the current mode.

- Disabled (0)
- Enabled (1)
- Auto (2) default

Return type*brainstem.result.Result***getLinkState ()**

Gets the current state of the connection in the form of a bitmask.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Pointer to be filled with the current connection state bits.

Return type*brainstem.result.Result***getNumberOfPowerDataObjects (partner, power_role)**

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

Parameters

- **partner** (*unsigned char*) -

Indicates which side of the PD connection is in question.

- Local = 0 = powerdeliveryPartnerLocal
- Remote = 1 = powerdeliveryPartnerRemote

- **power_role** (*unsigned char*) -

Indicates which power role of PD connection is in question.

- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with the number of PDOs.

Return type

brainstem.result.Result

getOverride()

Gets the current enabled overrides

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bit mapped representation of the current override configuration.

Return type

brainstem.result.Result

getPeakCurrentConfiguration()

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

An enumerated value referring to the current configuration.

- Allowable values are 0 - 4

Return type

brainstem.result.Result

getPowerDataObject (*partner, power_role, rule_index*)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

Parameters

- **partner** (*unsigned char*) -
Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **power_role** (*unsigned char*) -
Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **rule_index** (*unsigned char*) - The index of the PDO in question. Valid index are 1-7.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the requested power rule.

Return type

brainstem.result.Result

getPowerDataObjectEnabled (*power_role, rule_index*)

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

Parameters

- **power_role** (*unsigned char*) -
Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **rule_index** (*unsigned char*) - The index of the PDO in question. Valid index are 1-7.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] Variable to be filled with enabled state.

Return type*brainstem.result.Result***getPowerDataObjectEnabledList** (*power_role*)

Gets all Power Data Object enables for a given power role. Equivalent of calling PowerDeliveryClass::getPowerDataObjectEnabled() for all indexes.

Parameters**power_role** (*unsigned char*) -

Indicates which power role of PD connection is in question.

- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with a mapped representation of the enabled PDOs for a given power role. Values align with a given rule index (bits 1-7, bit 0 is invalid)

Return type*brainstem.result.Result***getPowerDataObjectList** (*buffer_length=16384*)

Gets all Power Data Objects (PDOs). Equivalent to calling PowerDeliveryClass::getPowerDataObject() on all partners, power roles, and index's.

Parameters**buffer_length** (*unsigned int*) - Length of the buffer to be filled**Returns**

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned int)] pointer to the start of a c style buffer to be filled The order of which is:

- Rules 1-7 Local Source
- Rules 1-7 Local Sink
- Rules 1-7 Partner Source
- Rules 1-7 Partner Sink.

Return type*brainstem.result.Result***getPowerRole** ()

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filed with the power role

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Return type

brainstem.result.Result

getPowerRoleCapabilities()

Gets the power roles that may be advertised by the local partner. (CC Strapping).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filed with the power role

- None = 0 = pdPowerRoleCapabilities_None
- Source = 1 = pdPowerRoleCapabilities_Source
- Sink = 2 = pdPowerRoleCapabilities_Sink
- Source/Sink = 3 = pdPowerRoleCapabilities_DualRole (Dual Role Port)

Return type

brainstem.result.Result

getPowerRolePreferred()

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Return type*brainstem.result.Result***getProtocolRevisionMax()**

Gets the configured maximum USB Power Delivery protocol revision used as a negotiation ceiling (pdProtocolRevision2 = 2, pdProtocolRevision3 = 3). This is unrelated to hardware or product revision.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Filled with pdProtocolRevision2 (2) or pdProtocolRevision3 (3).

Return type*brainstem.result.Result***getRequestDataObject(partner)**

Gets the current Request Data Object (RDO) for a given partner. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

Parameters

partner (*unsigned char*) -

Indicates which side of the PD connection is in question.

- Local = 0 = powerdeliveryPartnerLocal
- Remote = 1 = powerdeliveryPartnerRemote

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the current RDO. Zero indicates the RDO is not active.

Return type*brainstem.result.Result***static packDataObjectAttributes(partner, power_role, rule_index)**

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

Parameters

- **partner** (*unsigned char*) -

Indicates which side of the PD connection.

- Local = 0 = powerdeliveryPartnerLocal
- Remote = 1 = powerdeliveryPartnerRemote

- **power_role** (*unsigned char*) -

Indicates which power role of PD connection.

- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

- **rule_index** (*unsigned char*) – Data object index.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with packed values.

Return type

brainstem.result.Result

request (*request*)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

Parameters

request (*unsigned char*) –

Request to be issued to the remote partner

- pdRequestHardReset (0)
- pdRequestSoftReset (1)
- pdRequestDataReset (2)
- pdRequestPowerRoleSwap (3)
- pdRequestPowerFastRoleSwap (4)
- pdRequestDataRoleSwap (5)
- pdRequestVconnSwap (6)
- pdRequestSinkGoToMinimum (7)
- pdRequestRemoteSourcePowerDataObjects (8)
- pdRequestRemoteSinkPowerDataObjects (9)
- pdRequestRemoteSourceExtendedCapabilities (10)
- pdRequestRemoteSinkExtendedCapabilities (11)
- pdRequestStatus (12)
- pdRequestPPSSStatus (13)
- pdRequestBatteryCapabilities (14)
- pdRequestBatteryStatus (15)
- pdRequestManufacturerInfoSop (16)
- pdRequestManufacturerInfoSopp (17)
- pdRequestManufacturerInfoSopp (18)
- pdRequestDiscoverIdentitySop (19)
- pdRequestDiscoverIdentitySopp (20)

- `pdRequestDiscoverIdentitySoppp` (21)
- `pdRequestRevision` (22)
- `pdRequestSourceInfo` (23)
- `pdRequestCountryCode` (24)
- `pdRequestCountryInfo` (25)
- `pdRequestRemoteSourceEPRCapabilities` (26)
- `pdRequestRemoteSinkEPRCapabilities` (27)

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

`requestStatus` ()

Gets the status of the last request command sent.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the status

Return type

`brainstem.result.Result`

`resetPowerDataObjectToDefault` (*`power_role`*, *`rule_index`*)

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

Parameters

- **`power_role`** (*`unsigned char`*) -

Indicates which power role of PD connection is in question.

- Source = 1 = `powerdeliveryPowerRoleSource`
- Sink = 2 = `powerdeliveryPowerRoleSink`

- **`rule_index`** (*`unsigned char`*) - The index of the PDO in question. Valid index are 1-7.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

`setFastRoleSwapCurrent` (*`swap_current`*)

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

Parameters

- **`swap_current`** (*`unsigned char`*) -

An enumerated value referring to value to be set.

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setFlagMode (*flag, mode*)

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Parameters

- **flag** (*unsigned char*) – Flag/Advertisement to be modified
- **mode** (*unsigned char*) –

Value to be applied.

- Disabled (0)
- Enabled (1)
- Auto (2) default

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setOverride (*overrides*)

Sets the current enabled overrides

Parameters

overrides (*unsigned int*) – Overrides to be set in a bit mapped representation.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPeakCurrentConfiguration (*configuration*)

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Parameters

configuration (*unsigned char*) –

An enumerated value referring to the configuration to be set

- Allowable values are 0 - 4

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerDataObject (*power_role*, *rule_index*, *pdo*)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

Parameters

- **power_role** (*unsigned char*) -
Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **rule_index** (*unsigned char*) - The index of the PDO in question. Valid index are 1-7.
- **pdo** (*unsigned int*) - Power Data Object to be set.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setPowerDataObjectEnabled (*power_role*, *rule_index*, *enabled*)

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

Parameters

- **power_role** (*unsigned char*) -
Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **rule_index** (*unsigned char*) - The index of the PDO in question. Valid index are 1-7.
- **enabled** (*bool*) - The state to be set.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setPowerRole (*power_role*)

Set the current power role to be advertised by the Local partner. (CC Strapping).

Parameters**power_role** (*unsigned char*) -**Value to be applied.**

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource

- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerRolePreferred (*power_role*)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

Parameters

power_role (*unsigned char*) -

Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setProtocolRevisionMax (*max_protocol_revision*)

Sets the maximum USB Power Delivery protocol revision for negotiation. Only `pdProtocolRevision2` (2) and `pdProtocolRevision3` (3) are valid API values.

Parameters

max_protocol_revision (*unsigned char*) - `pdProtocolRevision2` (2) or `pdProtocolRevision3` (3).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setRequestDataObject (*rdo*)

Sets the current Request Data Object (RDO) for a given partner. Only the local partner can be changed. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

Parameters

rdo (*unsigned int*) - Request Data Object to be set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

static unpackDataObjectAttributes (*attributes*)

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

Parameters

attributes (*unsigned char*) - Variable to be filled with packed values.

Returns

Object containing error code and returned values on success.

error

[unsigned byte] An error result from the list of defined error codes

partner

[unsigned char]

Indicates which side of the PD connection.

- Local = 0 = powerdeliveryPartnerLocal
- Remote = 1 = powerdeliveryPartnerRemote

power_role

[unsigned char]

Indicates which power role of PD connection.

- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

rule_index

[unsigned char] Data object index.

Return type

brainstem.result.Result

3.2.25 RCServo

See the *RCServo Entity* for generic information.

class `brainstem.entity.RCServo (module, index)`

Interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

getEnable()

Get the enable status of the servo channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] The current enable status of the servo entity. 0 is disabled, 1 is enabled.

Return type

brainstem.result.Result

getPosition()

Get the position of the servo channel

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current position of the servo channel. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Return type

brainstem.result.Result

getReverse ()

Get the reverse status of the servo channel

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] The current reverse status of the servo entity. 0 = not reversed, 1 = reversed.

Return type

brainstem.result.Result

setEnabled (enable)

Enable the servo channel

Parameters

enable (*bool*) – The state to be set. 0 is disabled, 1 is enabled.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setPosition (position)

Set the position of the servo channel

Parameters

position (*unsigned char*) – The position to be set. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setReverse (reverse)

Set the output to be reversed on the servo channel

Parameters

reverse (*bool*) – Reverses the value set by “setPosition”. For example, if the position is set to 64 (1ms pulse) the output will now be 192 (2ms pulse), however “getPostion” will return the set value of 64. 0 = not reversed, 1 = reversed.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type
unsigned byte

3.2.26 Rail

See the [Rail Entity](#) for generic information.

class `brainstem.entity.Rail (module, index)`

Provides power rail functionality on certain modules. The RailClass can be used to control power to downstream devices. It has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

clearFaults ()

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type
unsigned byte

getCurrent ()

Get the rail current.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The current in micro-amps (1 == 1e-6A).

Return type
[brainstem.result.Result](#)

getCurrentLimit ()

Get the rail current limit setting. (Check product datasheet to see if this feature is available)

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The current in micro-amps (1 == 1e-6A).

Return type
[brainstem.result.Result](#)

getCurrentSetpoint ()

Get the rail setpoint current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The current in micro-amps ($1 == 1e-6A$) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the `setCurrent` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Return type

brainstem.result.Result

getEnable()

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage

Return type

brainstem.result.Result

getKelvinSensingEnable()

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] Kelvin sensing is enabled or disabled.

Return type

brainstem.result.Result

getKelvinSensingState()

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] Kelvin sensing is enabled or disabled.

Return type

brainstem.result.Result

getOperationalMode()

Determine the current operational mode of the system. Refer to the module datasheet for definition of the rail operational mode capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current operational mode setting.

Return type

brainstem.result.Result

getOperationalState()

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The current operational state, hardware configuration, faults, and operating mode.

Return type

brainstem.result.Result

getPower()

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The power in micro-watts ($1 == 1\text{e-}6\text{W}$) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to determine if this is a measured or stored value.

Return type

brainstem.result.Result

getPowerLimit()

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The power in micro-watts (1 == 1e-6W).

Return type

brainstem.result.Result

getPowerSetpoint ()

Get the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The power in micro-watts (1 == 1e-6W) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to determine if this is a measured or stored value.

Return type

brainstem.result.Result

getResistance ()

Get the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The resistance in micro-ohms (1 == 1e-6Ohms) currently drawn by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setResistance interface. Refer to the module datasheet to determine if this is a measured or stored value.

Return type

brainstem.result.Result

getResistanceSetpoint ()

Get the rail setpoint resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The resistance in micro-ohms (1 == 1e-6Ohms) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setResistance interface. Refer to the module datasheet to determine if this is a measured or stored value.

Return type*brainstem.result.Result***getTemperature ()**

Get the rail temperature.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The measured temperature associated with the rail in micro-Celsius (1 == 1e-6°C). The temperature may be associated with the module's internal rail circuitry or an externally connected temperature sensors. Refer to the module datasheet for definition of the temperature measurement location and specific capabilities.

Return type*brainstem.result.Result***getVoltage ()**

Get the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The voltage in micro-volts (1 == 1e-6V) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Return type*brainstem.result.Result***getVoltageMaxLimit ()**

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The voltage in micro-volts (1 == 1e-6V).

Return type*brainstem.result.Result***getVoltageMinLimit ()**

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The voltage in micro-volts (1 == 1e-6V).

Return type

brainstem.result.Result

getVoltageSetpoint ()

Get the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The voltage in micro-volts (1 == 1e-6V) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Return type

brainstem.result.Result

setCurrentLimit (microamps)

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

Parameters

microamps (*int*) – The current in micro-amps (1 == 1e-6A).

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setCurrentSetpoint (microamps)

Set the rail supply current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Parameters

microamps (*int*) – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setEnabled (enable)

Set the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Parameters

enable (*bool*) – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setKelvinSensingEnable (*enable*)

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

enable (*bool*) – enable or disable kelvin sensing.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setOperationalMode (*mode*)

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

Parameters

mode (*unsigned char*) – The operational mode to employ.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerLimit (*microwatts*)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microwatts (*int*) – The power in micro-watts (1 == 1e-6W).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerSetpoint (*microwatts*)

Set the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

microwatts (*int*) – The power in micro-watts (1 == 1e-6W) to be supplied by the rail.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setResistanceSetpoint (*microohms*)

Set the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

microohms (*int*) – The resistance in micro-ohms (1 == 1e-6Ohms) to be drawn by the rail.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVoltageMaxLimit (*microvolts*)

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts (*int*) – The voltage in micro-volts (1 == 1e-6V).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVoltageMinLimit (*microvolts*)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts (*int*) – The voltage in micro-volts (1 == 1e-6V).

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setVoltageSetpoint (*microvolts*)

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters

microvolts (*int*) – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.27 Relay

See the [Relay Entity](#) for generic information.

class `brainstem.entity.Relay` (*module, index*)

Interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

getEnable ()

Get the state.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] False or 0 = Disabled, True or 1 = Enabled

Return type

brainstem.result.Result

getVoltage ()

Get the scaled micro volt value with reference to ground.

Note: Not all modules provide 32 bits of accuracy. Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] 32 bit signed integer (in micro Volts) based on the boards ground and reference voltages.

Return type

brainstem.result.Result

setEnabled (enable)

Set the enable/disable state.

Parameters

enable (*bool*) – False or 0 = Disabled, True or 1 = Enabled

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

3.2.28 Signal

See the *Signal Entity* for generic information.

class *brainstem.entity.Signal* (*module, index*)

Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

getEnable()

Get the Enable/Disable of the signal.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] True to enable, false to disable

Return type

brainstem.result.Result

getInvert()

Get the invert status the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] True to invert, false for normal mode.

Return type

brainstem.result.Result

getT2Time()

Get the signal active period or T2 in nanoseconds.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Return type

brainstem.result.Result

getT3Time()

Get the signal period or T3 in nanoseconds.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Return type*brainstem.result.Result***setEnabled** (*enable*)

Enable/Disable the signal output.

Parameters**enable** (*bool*) – True to enable, false to disable**Returns**An error result from the list of defined error codes in *brainstem.result.Result***Return type**

unsigned byte

setInvert (*invert*)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters**invert** (*bool*) – True to invert, false for normal mode.**Returns**An error result from the list of defined error codes in *brainstem.result.Result***Return type**

unsigned byte

setT2Time (*t2_nsec*)

Set the signal active period or T2 in nanoseconds.

Parameters**t2_nsec** (*unsigned int*) – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.**Returns**An error result from the list of defined error codes in *brainstem.result.Result***Return type**

unsigned byte

setT3Time (*t3_nsec*)

Set the signal period or T3 in nanoseconds.

Parameters**t3_nsec** (*unsigned int*) – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.**Returns**An error result from the list of defined error codes in *brainstem.result.Result***Return type**

unsigned byte

3.2.29 Store

See the [Store Entity](#) for generic information.

class `brainstem.entity.Store (module, index)`

The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

getSlotCapacity (slot)

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold.

Parameters

slot (*unsigned char*) – The slot number.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The slot capacity.

Return type

[brainstem.result.Result](#)

getSlotLocked (slot)

Gets the current lock state of the slot Allows for write protection on a slot.

Parameters

slot (*unsigned char*) – The slot number

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] Variable to be filed with the locked state.

Return type

[brainstem.result.Result](#)

getSlotSize (slot)

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes.

Parameters

slot (*unsigned char*) – The slot number.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The slot size.

Return type

brainstem.result.Result

getSlotState (slot)

Get slot state.

Parameters

slot (*unsigned char*) – The slot number.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] true: enabled, false: disabled.

Return type

brainstem.result.Result

loadSlot (slot, buffer)

Load the slot.

Parameters

- **slot** (*unsigned char*) – The slot number.
- **buffer** (*unsigned char*) – The data.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setSlotLocked (slot, lock)

Sets the locked state of the slot Allows for write protection on a slot.

Parameters

- **slot** (*unsigned char*) – The slot number
- **lock** (*bool*) – state to be set.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

slotDisable (slot)

Disable slot.

Parameters

slot (*unsigned char*) – The slot number.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

slotEnable (*slot*)

Enable slot.

Parameters**slot** (*unsigned char*) – The slot number.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

unloadSlot (*slot, buffer_length=65536*)

Unload the slot data.

Parameters

- **slot** (*unsigned char*) – The slot number.
- **buffer_length** (*unsigned int*) – The length of buffer buffer in bytes. This is the maximum number of bytes that should be unloaded.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned char)] Byte array that the unloaded data will be placed into.

Return type*brainstem.result.Result*

3.2.30 System

See the *System Entity* for generic information.**class** `brainstem.entity.System` (*module, index*)

The System class provides access to the core settings, configuration and system information of the Brain-Stem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc.

getBootSlot ()

Get the store slot which is mapped when the module boots.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The slot number in aSTORE_INTERNAL that is mapped after the module boots.

Return type*brainstem.result.Result***getBuild()**

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with build.

Return type*brainstem.result.Result***getErrors()**

Gets any system level errors. Calling this function will clear the current errors. If the error persists it will be set again.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bit mapped field representing the devices errors

Return type*brainstem.result.Result***getHBInterval()**

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current heartbeat delay.

Return type*brainstem.result.Result***getHardwareVersion()**

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The module's hardware version information.

Return type

brainstem.result.Result

getInputCurrent ()

Get the module's input current.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The module's input current reported in microamps.

Return type

brainstem.result.Result

getInputPowerBehavior ()

Gets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated value representing behavior.

Return type

brainstem.result.Result

getInputPowerBehaviorConfig (buffer_length=16384)

Gets the input power behavior configuration. Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

buffer_length (*unsigned int*) – Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned int)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getInputPowerSource ()

Provides the source of the current power source in use.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with enumerated representation of the source.

Return type

brainstem.result.Result

getInputVoltage ()

Get the module's input voltage.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The module's input voltage reported in microvolts.

Return type

brainstem.result.Result

getLED ()

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] true: LED on, false: LED off.

Return type

brainstem.result.Result

getLEDMaxBrightness ()

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum).

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Brightness value relative to 255

Return type

brainstem.result.Result

getLinkInterface ()

Gets the link interface configuration. This refers to which interface is being used for control by the device.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Variable to be filled with an enumerated value representing interface.

- 0 = Auto= systemLinkAuto
- 1 = Control Port = systemLinkUSBControl
- 2 = Hub Upstream Port = systemLinkUSBHub

Return type

brainstem.result.Result

getMaximumTemperature ()

Get the module's maximum temperature ever recorded in micro-C (uC). This value will persists through a power cycle.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The module's maximum system temperature in micro-C

Return type

brainstem.result.Result

getMinimumTemperature ()

Get the module's minimum temperature ever recorded in micro-C (uC). This value will persists through a power cycle.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The module's minimum system temperature in micro-C

Return type

brainstem.result.Result

getModel ()

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The module's model enumeration.

Return type

brainstem.result.Result

getModule ()

Get the current address the module uses on the BrainStem network.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The address the module is using on the BrainStem network.

Return type

brainstem.result.Result

getModuleBaseAddress ()

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The address the module is using on the BrainStem network.

Return type

brainstem.result.Result

getModuleHardwareOffset ()

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The module address offset.

Return type

brainstem.result.Result

getModuleSoftwareOffset ()

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The address for the module. Value must be even from 0-254.

Return type

brainstem.result.Result

getName (buffer_length=65536)

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

Parameters

buffer_length (*unsigned int*) – Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[string] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getPowerLimit ()

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The available power in milli-Watts (mW, 1 t)

Return type

brainstem.result.Result

getPowerLimitMax ()

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the power limit in milli-Watts (mW)

Return type

brainstem.result.Result

getPowerLimitState ()

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit().

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Variable to be filled with the state.

Return type

brainstem.result.Result

getProtocolFeatures ()

Gets the firmware protocol features

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Value representing the firmware protocol features

Return type

brainstem.result.Result

getRouter ()

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The address.

Return type

brainstem.result.Result

getRouterAddressSetting ()

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The address for the module. Value must be even from 0-254.

Return type*brainstem.result.Result***getSerialNumber()**

Get the module's serial number. The serial number is a unique 32 bit integer which is usually communicated in hexadecimal format.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The module's serial number.

Return type*brainstem.result.Result***getTemperature()**

Get the module's current temperature in micro-C

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The module's system temperature in micro-C

Return type*brainstem.result.Result***getUnregulatedCurrent()**

Gets the current passing through the unregulated port.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] Variable to be filled with the current in micro-Amps (uA).

Return type*brainstem.result.Result*

getUnregulatedVoltage()

Gets the voltage present at the unregulated port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] Variable to be filled with the voltage in micro-Volts (uV).

Return type

brainstem.result.Result

getUptime()

Get the module's accumulated uptime in minutes

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The module's accumulated uptime in minutes.

Return type

brainstem.result.Result

getVersion()

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The build version date code.

Return type

brainstem.result.Result

logEvents()

Saves system log events to a slot defined by the module (usually ram slot 0).

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

reset()

Reset the system. A return value of aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

resetDeviceToFactoryDefaults ()

Resets the device to it factory default configuration.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

routeToMe (enable)

Enables/Disables the route to me function. This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

Parameters

enable (*bool*) – Enable or disable of the route to me function 1 = enable.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

save ()

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setBootSlot (slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

Parameters

slot (*unsigned char*) – The slot number in `aSTORE_INTERNAL` to be marked as a boot slot.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setHBInterval (interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

Parameters

interval (*unsigned char*) – The desired heartbeat delay.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setInputPowerBehavior (*behavior*)

Sets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Parameters

behavior (*unsigned char*) – An enumerated representation of behavior to be set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setInputPowerBehaviorConfig (*buffer*)

Sets the input power behavior configuration. Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

buffer (*list (unsigned int)*) – Pointer to the start of a c style buffer to be transferred.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setLED (*led_on*)

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

Parameters

led_on (*bool*) – true: turn the LED on, false: turn LED off.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setLEDMaxBrightness (*brightness*)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Parameters

brightness (*unsigned char*) – Brightness value relative to 255

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setLinkInterface (*link_interface*)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters**link_interface** (*unsigned char*) -**An enumerated representation of interface to be set.**

- 0 = Auto= systemLinkAuto
- 1 = Control Port = systemLinkUSBControl
- 2 = Hub Upstream Port = systemLinkUSBHub

ReturnsAn error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setModuleSoftwareOffset (*address*)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters**address** (*unsigned char*) - The address for the module. Value must be even from 0-254.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setName (*buffer*)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

Parameters**buffer** (*string*) - Pointer to the start of a c style buffer to be transferred.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setPowerLimitMax (*power*)

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters**power** (*unsigned int*) - Limit in milli-Watts (mW) to be set.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setRouter (*address*)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

Parameters**address** (*unsigned char*) – The router address to be used.**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

3.2.31 Temperature

See the [Temperature Entity](#) for generic information.

class `brainstem.entity.Temperature` (*module, index*)

This entity is only available on certain modules, and provides a temperature reading in microcelsius.

getValue ()

Get the modules temperature in micro-C

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The temperature in micro-Celsius (1 == 1e-6C).

Return type[brainstem.result.Result](#)**getValueMax** ()

Get the module’s maximum temperature in micro-C since the last power cycle.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The module’s maximum temperature in micro-C

Return type[brainstem.result.Result](#)**getValueMin** ()

Get the module’s minimum temperature in micro-C since the last power cycle.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The module's minimum temperature in micro-C

Return type*brainstem.result.Result*

3.2.32 Timer

See the *Timer Entity* for generic information.**class** `brainstem.entity.Timer (module, index)`

The Timer Class provides access to a simple scheduler. The timer can set to fire only once, or to repeat at a certain interval. Additionally, a timer entity can execute custom Reflex routines upon firing.

getExpiration()

Get the currently set expiration time in microseconds. This is not a "live" timer. That is, it shows the expiration time originally set with `setExpiration`; it does not "tick down" to show the time remaining before expiration.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The timer expiration duration in microseconds.

Return type*brainstem.result.Result***getMode()**

Get the mode of the timer which is either single or repeat mode.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The mode of the time. `aTIMER_MODE_REPEAT` or `aTIMER_MODE_SINGLE`.

Return type*brainstem.result.Result***setExpiration (usec_duration)**

Set the expiration time for the timer entity. When the timer expires, it will fire the associated `timer[index]()` reflex.

Parameters

usec_duration (*unsigned int*) – The duration before timer expiration in microseconds.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setMode (*mode*)

Set the mode of the timer which is either single or repeat mode.

Parameters

mode (*unsigned char*) – The mode of the timer. `aTIMER_MODE_REPEAT` or `aTIMER_MODE_SINGLE`.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.33 UART

See the [UART Entity](#) for generic information.

class `brainstem.entity.UART` (*module, index*)

A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port. The UART Class allows the enabling and disabling of the UART data lines.

getAvailableProtocols ()

Returns a bitmask containing a list of protocols that this UART entity is capable of selecting, and has an available protocol resource to assign.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bitmask containing list of protocols that are available to select. The value of the `uartProtocol` is mapped to the bit index (e.g. `uartProtocol_Undefined` is bit 0, `uartProtocol_ExtronResponder_Value` is bit 1, etc.)

Return type

brainstem.result.Result

getBaudRate ()

Get the UART baud rate. If zero, automatic baud rate selection is used.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Pointer variable to be filled with baud rate.

Return type

brainstem.result.Result

getCapableProtocols ()

Returns a bitmask containing a list of protocols that this UART entity is allowed to select. This does not guarantee that selecting a protocol with “setProtocol” will have an available resource.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bitmask containing list of protocols that may be selected. The value of the uartProtocol is mapped to the bit index (e.g. uartProtocol_Undefined is bit 0, uartProtocol_ExtronResponder_Value is bit 1, etc.)

Return type

brainstem.result.Result

getDataBits ()

Get the number of bits per character

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Pointer to where result is placed.

Return type

brainstem.result.Result

getEnable ()

Get the enabled state of the uart.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] true: enabled, false: disabled.

Return type

brainstem.result.Result

getFlowControl ()

Set the UART flow control configuration

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Pointer to bitmask where result is placed. Possible bits:

- uartFlowControl_RTS_CTS_Bit
- uartFlowControl_DSR_DTR_Bit
- uartFlowControl_XON_XOFF_Bit

Return type

brainstem.result.Result

getLinkChannel ()

Gets the index of the UART Entity that this entity is linked to.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Pointer to where result is placed.

Return type

brainstem.result.Result

getParity ()

Get the UART parity.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Pointer variable to be filled with value. Possible values:

- uartParity_None_Value
- uartParity_Odd_Value
- uartParity_Even_Value
- uartParity_Mark_Value
- uartParity_Space_Value

Return type

brainstem.result.Result

getProtocol ()

Get the UART protocol.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Pointer to where result is placed.

Return type

brainstem.result.Result

getStopBits()

Set the UART stop bit configuration

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Pointer to where result is placed. Possible values:

- uartStopBits_1_Value
- uartStopBits_1p5_Value
- uartStopBits_2_Value

Return type

brainstem.result.Result

setBaudRate(rate)

Set the UART baud rate. If zero, automatic baud rate selection is used.

Parameters

rate (*unsigned int*) – baud rate.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setDataBits(data_bits)

Set the number of bits per character

Parameters

data_bits (*unsigned char*) – Data Bits of UART Channel.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setEnabled(enabled)

Enable the UART channel.

Parameters

enabled (*bool*) – true: enabled, false: disabled.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setFlowControl (*flow_control*)

Set the UART flow control configuration

Parameters**flow_control** (*unsigned char*) -**Flow Control of UART Channel as a bitmask. Allowed bits:**

- uartFlowControl_RTS_CTS_Bit
- uartFlowControl_DSR_DTR_Bit
- uartFlowControl_XON_XOFF_Bit

ReturnsAn error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setLinkChannel (*channel*)

Set the index of another UART Entity that should be linked to this UART.

If set to the index of this entity, the channel will not be linked. If set to the index of another UART entity, data will be sent between the two UART entities with no additional processing.

Parameters**channel** (*unsigned char*) - Index of the UART Entity to link**Returns**An error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setParity (*parity*)

Set the UART parity.

Parameters**parity** (*unsigned char*) -**Parity of UART Channel. Allowed options:**

- uartParity_None_Value
- uartParity_Odd_Value
- uartParity_Even_Value
- uartParity_Mark_Value
- uartParity_Space_Value

ReturnsAn error result from the list of defined error codes in `brainstem.result.Result`**Return type**

unsigned byte

setProtocol (*protocol*)

Set the UART protocol.

Parameters

protocol (*unsigned char*) – An enumeration of serial protocols.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setStopBits (*stop_bits*)

Set the UART stop bit configuration

Parameters

stop_bits (*unsigned char*) –

Stop Bits of UART Channel. Allowed options:

- `uartStopBits_1_Value`
- `uartStopBits_1p5_Value`
- `uartStopBits_2_Value`

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.34 USB

See the [USB Entity](#) for generic information.

class `brainstem.entity.USB` (*module, index*)

The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

clearPortErrorStatus (*channel*)

Clear the error status for the given port.

Parameters

channel (*unsigned char*) – The port to clear error status for.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

getAltModeConfig (*channel*)

Get USB Alt Mode Configuration.

Parameters

channel (*unsigned char*) – The USB sub channel

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The USB configuration for the given channel.

Return type

brainstem.result.Result

getCC1Current (*channel*)

Get the current through the CC1 for a port.

Parameters

channel1 (*unsigned char*) – The USB sub channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel current in micro-amps (1 == 1e-6A).

Return type

brainstem.result.Result

getCC1Enable (*channel*)

Get Enable/Disable on the CC1 line.

Parameters

channel1 (*unsigned char*) – USB channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool]

State to be filled

- Disabled: 0
- Enabled: 1

Return type

brainstem.result.Result

getCC1Voltage (*channel*)

Get the voltage of CC1 for a port.

Parameters

channel1 (*unsigned char*) – The USB sub channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type*brainstem.result.Result***getCC2Current** (*channel*)

Get the current through the CC2 for a port.

Parameters**channel1** (*unsigned char*) – The USB sub channel.**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel current in micro-amps (1 == 1e-6A).

Return type*brainstem.result.Result***getCC2Enable** (*channel*)

Get Enable/Disable on the CC2 line.

Parameters**channel1** (*unsigned char*) – USB channel.**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[bool]

State to be filled

- Disabled: 0
- Enabled: 1

Return type*brainstem.result.Result***getCC2Voltage** (*channel*)

Get the voltage of CC2 for a port.

Parameters**channel1** (*unsigned char*) – The USB sub channel.**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type*brainstem.result.Result*

getCableFlip (*channel*)

Get Cable flip setting.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] The enable/disable status of cable flip.

Return type

brainstem.result.Result

getConnectMode (*channel*)

Gets the connect mode of the switch.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current connect mode

Return type

brainstem.result.Result

getDownstreamBoostMode ()

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current Downstream boost setting 0, 1, 2, or 3.

Return type

brainstem.result.Result

getDownstreamDataSpeed (*channel*)

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

Parameters

channel (*unsigned char*) – USB downstream channel to check.

Returns

Object containing error code and returned value on success.**error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char]

Filled with the current port data speed

- N/A: usbDownstreamDataSpeed_na = 0
- Hi Speed: usbDownstreamDataSpeed_hs = 1
- SuperSpeed: usbDownstreamDataSpeed_ss = 2

Return type*brainstem.result.Result***getEnumerationDelay ()**

Get the inter-port enumeration delay in milliseconds.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Millisecond delay in 100mS increments (100, 200, 300 etc.)

Return type*brainstem.result.Result***getHubMode ()**

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

Returns**Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The USB hub mode.

Return type*brainstem.result.Result***getPortCurrent (channel)**

Get the current through the power line for a port.

Parameters**channel** (*unsigned char*) – The USB sub channel.**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel current in micro-amps (1 == 1e-6A).

Return type

brainstem.result.Result

getPortCurrentLimit (*channel*)

Get the current limit for the port.

Parameters

channel1 (*unsigned char*) - USB downstream channel to limit.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The current limit setting.

Return type

brainstem.result.Result

getPortError (*channel*)

Get the current error for the Port.

Parameters

channel1 (*unsigned char*) - USB downstream channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Return type

brainstem.result.Result

getPortMode (*channel*)

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of usbPortMode.

Parameters

channel1 (*unsigned char*) - USB downstream channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Return type*brainstem.result.Result***getPortState** (*channel*)

Get the current State for the Port.

Parameters**channel** (*unsigned char*) – USB downstream channel.**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Return type*brainstem.result.Result***getPortVoltage** (*channel*)

Get the voltage on the power line for a port.

Parameters**channel** (*unsigned char*) – The USB sub channel.**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in microvolts (1 == 1e-6V).

Return type*brainstem.result.Result***getSBU1Voltage** (*channel*)

Get the voltage of SBU1 for a port.

Parameters**channel** (*unsigned char*) – The USB sub channel.**Returns****Object containing error code and returned value on success.****error**

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type*brainstem.result.Result***getSBU2Voltage** (*channel*)

Get the voltage of SBU2 for a port.

Parameters

channel1 (*unsigned char*) – The USB sub channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[int] The USB channel voltage in micro-volts (1 == 1e-6V).

Return type

brainstem.result.Result

getSBUEnable (*channel*)

Get the Enable/Disable status of the SBU

Parameters

channel1 (*unsigned char*) – The USB sub channel.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[bool] The enable/disable status of the SBU

Return type

brainstem.result.Result

getUpstreamBoostMode ()

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current Upstream boost setting 0, 1, 2, or 3.

Return type

brainstem.result.Result

getUpstreamMode ()

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The Upstream port mode.

Return type*brainstem.result.Result***getUpstreamState ()**

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The Upstream port state.

Return type*brainstem.result.Result***setAltModeConfig (channel, configuration)**

Set USB Alt Mode Configuration.

Parameters

- **channel** (*unsigned char*) – The USB sub channel
- **configuration** (*unsigned int*) – The USB configuration to be set for the given channel.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setCC1Enable (channel, enable)

Set Enable/Disable on the CC1 line.

Parameters

- **channel** (*unsigned char*) – USB channel.
- **enable** (*bool*) –

State to be set

- Disabled: 0
- Enabled: 1

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setCC2Enable (channel, enable)

Set Enable/Disable on the CC2 line.

Parameters

- **channel** (*unsigned char*) – USB channel.

- **enable** (*bool*) –

State to be filled

- Disabled: 0
- Enabled: 1

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setCableFlip (*channel*, *enable*)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters

- **channel** (*unsigned char*) – The USB sub channel.
- **enable** (*bool*) –

The state to be set The state to be set

- Disabled: 0
- Enabled: 1

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setConnectMode (*channel*, *mode*)

Sets the connect mode of the switch.

Parameters

- **channel** (*unsigned char*) – The USB sub channel.
- **mode** (*unsigned char*) –

The connect mode

- `usbManualConnect` = 0
- `usbAutoConnect` = 1

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataDisable (*channel*)

Disable only the data lines for a port without changing the state of the power line.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDataEnable (*channel*)

Enable the only the data lines for a port without changing the state of the power line.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setDownstreamBoostMode (*setting*)

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Parameters

setting (*unsigned char*) – Downstream boost setting 0, 1, 2, or 3.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setEnumerationDelay (*ms_delay*)

Set the inter-port enumeration delay in milliseconds.

Parameters

ms_delay (*unsigned int*) – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setHiSpeedDataDisable (*channel*)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setHiSpeedDataEnable (*channel*)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setHubMode (*mode*)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

Parameters

mode (*unsigned int*) – The USB hub mode.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPortCurrentLimit (*channel, microamps*)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a `stem.system.save()` call.

Parameters

- **channel** (*unsigned char*) – USB downstream channel to limit.
- **microamps** (*unsigned int*) – The current limit setting.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPortDisable (*channel*)

Disable both power and data lines for a port.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPortEnable (*channel*)

Enable both power and data lines for a port.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPortMode (*channel, mode*)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of `usbPortMode`.

Parameters

- **channel** (*unsigned char*) – USB downstream channel to set the mode on.
- **mode** (*unsigned int*) – The port mode setting as packed bit field.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerDisable (*channel*)

Disable only the power line for a port without changing the state of the data lines.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setPowerEnable (*channel*)

Enable only the power line for a port without changing the state of the data lines.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setSBUEnable (*channel, enable*)

Enable/Disable only the SBU1/2 based on the configuration of the `usbPortMode` settings.

Parameters

- **channel** (*unsigned char*) – The USB sub channel.
- **enable** (*bool*) – The state to be set - Disabled: 0 - Enabled: 1

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setSuperSpeedDataDisable (*channel*)

Disable only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setSuperSpeedDataEnable (*channel*)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters

channel (*unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setUpstreamBoostMode (*setting*)

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Parameters

setting (*unsigned char*) – Upstream boost setting 0, 1, 2, or 3.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setUpstreamMode (*mode*)

Set the upstream switch mode for the USB upstream ports. Values are `usbUpstreamModeAuto`, `usbUpstreamModePort0`, `usbUpstreamModePort1`, and `usbUpstreamModeNone`.

Parameters

mode (*unsigned char*) – The Upstream port mode.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

3.2.35 USBSystem

See the [USBSys^{tem} Entity](#) for generic information.

class `brainstem.entity.USBSystem` (*module, index*)

The USBSystem class provides high level control of the lower level Port Class.

getDataHSMaxData^{rate} ()

Gets the USB HighSpeed Max data^{rate}

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Current maximum datarate for the USB HighSpeed signals.

Return type

brainstem.result.Result

getDataRoleBehavior()

Gets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Return type

brainstem.result.Result

getDataRoleBehaviorConfig(buffer_length=16384)

Gets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine priority host priority.

Parameters

buffer_length (*unsigned int*) – Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned int)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getDataRoleList()

Gets the data role of all ports with a single call Equivalent to calling PortClass::getDataRole() on each individual port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] A bit packed representation of the data role for all ports.

Return type

brainstem.result.Result

getDataSSMaxDatarate()

Gets the USB SuperSpeed Max datarate

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Current maximum datarate for the USB SuperSpeed signals.

Return type

brainstem.result.Result

getEnabledList ()

Gets the current enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bit packed representation of the enabled status for all ports.

Return type

brainstem.result.Result

getEnumerationDelay ()

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] the current inter-port delay in milliseconds.

Return type

brainstem.result.Result

getModeList (buffer_length=16384)

Gets the current mode of all ports with a single call. Equivalent to calling PortClass::getMode() on each port.

Parameters

buffer_length (*unsigned int*) – Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned int)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getOverride()

Gets the current enabled overrides

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned int] Bit mapped representation of the current override configuration.

Return type

brainstem.result.Result

getPowerBehavior()

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system, i.e. What happens when requested power greater than available power.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Return type

brainstem.result.Result

getPowerBehaviorConfig(buffer_length=16384)

Gets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

buffer_length (*unsigned int*) - Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned int)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getSelectorMode()

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] Variable to be filled with the selector mode

Return type

brainstem.result.Result

getStateList (*buffer_length=16384*)

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port.

Parameters

buffer_length (*unsigned int*) - Length of the buffer to be filled

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[list(unsigned int)] pointer to the start of a c style buffer to be filled

Return type

brainstem.result.Result

getUpstream ()

Gets the upstream port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current upstream port.

Return type

brainstem.result.Result

getUpstreamHS ()

Gets the USB HighSpeed upstream port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current upstream port.

Return type

brainstem.result.Result

getUpstreamSS ()

Gets the USB SuperSpeed upstream port.

Returns

Object containing error code and returned value on success.

error

[unsigned byte] An error result from the list of defined error codes

value

[unsigned char] The current upstream port.

Return type

brainstem.result.Result

setDataHSMaxDatarate (datarate)

Sets the USB HighSpeed Max datarate

Parameters

datarate (*unsigned int*) – Maximum datarate for the USB HighSpeed signals.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setDataRoleBehavior (behavior)

Sets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters

behavior (*unsigned char*) – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setDataRoleBehaviorConfig (buffer)

Sets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine host priority.

Parameters

buffer (*list (unsigned int)*) – Pointer to the start of a c style buffer to be transferred.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setDataSSMaxDatarate (datarate)

Sets the USB SuperSpeed Max datarate

Parameters

datarate (*unsigned int*) – Maximum datarate for the USB SuperSpeed signals.

Returns

An error result from the list of defined error codes in *brainstem.result.Result*

Return type

unsigned byte

setEnabledList (*enabled_list*)

Sets the enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

Parameters

enabled_list (*unsigned int*) – Bit packed representation of the enabled status for all ports to be applied.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setEnumerationDelay (*ms_delay*)

Sets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Parameters

ms_delay (*unsigned int*) – The delay in milliseconds to be applied between port enables

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setModeList (*buffer*)

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port

Parameters

buffer (*list (unsigned int)*) – Pointer to the start of a c style buffer to be transferred.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setOverride (*overrides*)

Sets the current enabled overrides

Parameters

overrides (*unsigned int*) – Overrides to be set in a bit mapped representation.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setPowerBehavior (*behavior*)

Sets the behavior of how available power is managed, i.e. What happens when requested power is greater than available power.

Parameters

behavior (*unsigned char*) – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in brainstem.result.Result

Return type

unsigned byte

setPowerBehaviorConfig (*buffer*)

Sets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

buffer (*list* (*unsigned int*)) – Pointer to the start of a c style buffer to be transferred.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setSelectorMode (*mode*)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters

mode (*unsigned char*) – Mode to be set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setUpstream (*port*)

Sets the upstream port.

Parameters

port (*unsigned char*) – The upstream port to set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setUpstreamHS (*port*)

Sets the USB HighSpeed upstream port.

Parameters

port (*unsigned char*) – The upstream port to set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type

unsigned byte

setUpstreamSS (*port*)

Sets the USB SuperSpeed upstream port.

Parameters

port (*unsigned char*) – The upstream port to set.

Returns

An error result from the list of defined error codes in `brainstem.result.Result`

Return type
unsigned byte

3.3 C++ API Reference

3.3.1 Acroname Modules

USBHub3c

Class

class **aUSBHub3c** : public Acroname::BrainStem::Module

Concrete Module implementation of a USBHub3c Allows a user to connect to and control an attached hub.

Public Types

enum **PORT_ID**

Port ID

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_4**

enumerator **kPORT_ID_5**

enumerator **kPORT_ID_CONTROL**

enumerator **kPORT_ID_POWER_C**

typedef enum *aUSBHub3c::PORT_ID* **PORT_ID_t**

Port ID

Public Members

HubClass **hub**

Hub Class

Acroname::BrainStem::*PowerDeliveryClass* **pd**[aUSBHUB3C_NUM_PD_PORTS]

Power Delivery Class

Acroname::BrainStem::*RailClass* **rail**[aUSBHUB3C_NUM_RAILS]

Rail Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB3C_NUM_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TemperatureClass* **temperature**[aUSBHUB3C_NUM_TEMPERATURES]

Temperature Class

Acroname::BrainStem::*I2CClass* **i2c**[aUSBHUB3C_NUM_I2C]

I2C Class

Acroname::BrainStem::*USBClass* **usb**

USB Class

Acroname::BrainStem::*UARTClass* **uart**[aUSBHUB3C_NUM_UART]

UART Class

class **HubClass** : public Acroname::BrainStem::*USBSysClass*

Hub class implementation for use with USBHub3c.

Defines

aUSBHUB3C_MODULE 6

USBHub3c module number

aUSBHUB3C_NUM_STORES 2

Number of Store instances available

aUSBHUB3C_NUM_INTERNAL_SLOTS 12

Store: Number of internal slots instances available

aUSBHUB3C_NUM_RAM_SLOTS 1

Store: Number of RAM slot instances available

aUSBHUB3C_STORE_INTERNAL_INDEX 0

Store: Array index for internal store

aUSBHUB3C_STORE_RAM_INDEX 1

Store: Array index for RAM store

aUSBHUB3C_STORE_EEPROM_INDEX 2

Store: Array index for EEPROM store

aUSBHUB3C_NUM_TEMPERATURES 3

Number of Temperature instances available

aUSBHUB3C_NUM_USB 1

Number of USB instances available

aUSBHUB3C_NUM_USB_PORTS 8

Number of USB ports available

aUSBHUB3C_NUM_PORTS 8

Number of Ports available

aUSBHUB3C_NUM_PD_PORTS 8

Number of PD compatible ports available

aUSBHUB3C_NUM_PD_RULES_PER_PORT 7

Number of PD Rules per port available

aUSBHUB3C_NUM_RAILS 7

Number of Rail instances available

aUSBHUB3C_NUM_I2C 1

Number of I2C instances available

aUSBHUB3C_NUM_UART 2

Number of UART instances available

USBHub3p

Class

class **aUSBHub3p** : public Acroname::BrainStem::Module

Concrete Module implementation of a [aUSBHub3p](#) Allows a user to connect to and control an attached hub.

Public Types

enum **PORT_ID**

Port ID 3p

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_4**

enumerator **kPORT_ID_5**

enumerator **kPORT_ID_6**

enumerator **kPORT_ID_7**

enumerator **kPORT_ID_DWNA**

enumerator **kPORT_ID_UP0**

enumerator **kPORT_ID_UP1**

enumerator **kPORT_ID_CONTROL**

typedef enum *aUSBHub3p::PORT_ID* **PORT_ID_t**

Port ID 3p

Public Members

HubClass **hub**

Hub Class

Acroname::BrainStem::*AppClass* **app**[aUSBHUB3P_NUM_APPS]

App Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBHUB3P_NUM_POINTERS]

Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB3P_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass* **temperature**
Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBHUB3P_NUM_TIMERS]
Timer Class

Acroname::BrainStem::*USBClass* **usb**
USB Class

class **HubClass** : public Acroname::BrainStem::*USBSystemClass*
Hub class implementation for use with USBHub3p.

Defines

aUSBHUB3P_MODULE 6
USBHub3p module number

aUSBHUB3P_NUM_APPS 4
Number of App instances available

aUSBHUB3P_NUM_POINTERS 4
Number of Pointer instances available

aUSBHUB3P_NUM_STORES 2
Number of Store instances available

aUSBHUB3P_NUM_INTERNAL_SLOTS 12
Store: Number of internal slots instances available

aUSBHUB3P_NUM_RAM_SLOTS 1
Store: Number of RAM slot instances available

aUSBHUB3P_NUM_TIMERS 8
Number of Timer instances available

aUSBHUB3P_NUM_USB 1
Number of USB instances available

aUSBHUB3P_NUM_USB_PORTS 8
Number of USB ports available

aUSBHUB3P_NUM_PORTS 12
Number of Prts available

Port State Defines

aUSBHUB3P_USB_VBUS_ENABLED 0
USB VBUS current state

aUSBHUB3P_USB2_DATA_ENABLED 1
USB2 data current state

aUSBHUB3P_USB3_DATA_ENABLED 3
USB3 data current state

aUSBHUB3P_USB_SPEED_USB2 11
USB2 speed current state

aUSBHUB3P_USB_SPEED_USB3 12
USB3 speed current state

aUSBHUB3P_USB_ERROR_FLAG 19
Error indicator for this port
(see 'Port Errors' below)

aUSBHUB3P_USB2_BOOST_ENABLED 20
USB2 boost current state

aUSBHUB3P_DEVICE_ATTACHED 23
Device attached indicator for this port

Port State Error Defines

aUSBHUB3P_ERROR_VBUS_OVERCURRENT 0
VBUS overcurrent error

aUSBHUB3P_ERROR_VBUS_BACKDRIVE 1
VBUS backdrive (backpower) error

aUSBHUB3P_ERROR_HUB_POWER 2

Hub power error

aUSBHUB3P_ERROR_OVER_TEMPERATURE 3

Over temperature error

aUSBHUB3P_ERROR_DISCHARGE_ERR 4

For compat with USBHub2x4

aUSBHUB3P_ERROR_SHORT_CIRCUIT 5

Short circuit detected

USBHub2x4

Class

class **aUSBHub2x4** : public Acroname::BrainStem::Module

Concrete Module implementation of a USBHub2x4 Allows a user to connect to and control an attached hub.

Public Types

enum **PORT_ID**

Port ID 2x4

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_UP0**

enumerator **kPORT_ID_UP1**

typedef enum *aUSBHub2x4::PORT_ID* **PORT_ID_t**

Port ID 2x4

Public Members

HubClass **hub**

Hub Class

Acroname::BrainStem::*AppClass* **app**[aUSBHUB2X4_NUM_APPS]

App Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBHUB2X4_NUM_POINTERS]

Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB2X4_NUM_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TemperatureClass* **temperature**

Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBHUB2X4_NUM_TIMERS]

Timer Class

Acroname::BrainStem::*USBClass* **usb**

USB Class

class **HubClass** : public Acroname::BrainStem::*USBSystemClass*

Hub class implementation for use with USBHub2x4.

Defines

aUSBHUB2X4_MODULE 6

USBHub2x4 module number

aUSBHUB2X4_NUM_APPS 4

Number of App instances available

aUSBHUB2X4_NUM_POINTERS 4

Number of Pointer instances available

aUSBHUB2X4_NUM_STORES 2

Number of Store instances available

aUSBHUB2X4_NUM_INTERNAL_SLOTS 12

Store: Number of internal slots instances available

aUSBHUB2X4_NUM_RAM_SLOTS 1

Store: Number of RAM slot instances available

aUSBHUB2X4_NUM_TIMERS 8

Number of Timer instances available

aUSBHUB2X4_NUM_USB 1

Number of USB instances available

aUSBHUB2x4_NUM_USB_PORTS 4

Number of USB ports available

aUSBHUB2x4_NUM_PORTS 6

Number of Ports available

Port State Defines

aUSBHUB2X4_USB_VBUS_ENABLED 0

USB VBUS current state

aUSBHUB2X4_USB2_DATA_ENABLED 1

USB2 data current state

aUSBHUB2X4_USB_ERROR_FLAG 19

Error indicator for this port

(see 'Port Errors' below)

aUSBHUB2X4_USB2_BOOST_ENABLED 20

USB2 boost current state

aUSBHUB2X4_DEVICE_ATTACHED 23

Device attached indicator for this port

aUSBHUB2X4_CONSTANT_CURRENT 24

Constant current mode indicator

Port State Error Defines

aUSBHUB2X4_ERROR_VBUS_OVERCURRENT 0

VBUS overcurrent error

aUSBHUB2X4_ERROR_OVER_TEMPERATURE 3

Over temperature error

aUSBHub2X4_ERROR_DISCHARGE 4

Discharge error

USBCSwitchPro

Class

class **aUSBCSwitchPro** : public Acroname::BrainStem::Module

Concrete Module implementation of a USBCSwitchPro Allows a user to connect to and control an attached switch.

Public Types

enum **PORT_ID**

Port ID

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_COMMON**

enumerator **kPORT_ID_CONTROL**

typedef enum *aUSBCSwitchPro::PORT_ID* **PORT_ID_t**

Port ID

Public Members

Acroname::BrainStem::*PowerDeliveryClass* **pd**[aUSBCSWITCHPRO_NUM_PD_PORTS]
Power Delivery Class

Acroname::BrainStem::*RailClass* **rail**[aUSBCSWITCHPRO_NUM_RAILS]
Rail Class

Acroname::BrainStem::*StoreClass* **store**[aUSBCSWITCHPRO_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass*
temperature[aUSBCSWITCHPRO_NUM_TEMPERATURES]
Temperature Class

Acroname::BrainStem::*I2CClass* **i2c**[aUSBCSWITCHPRO_NUM_I2C]
I2C Class

Acroname::BrainStem::*USBClass* **usb**
USB Class

Acroname::BrainStem::*UARTClass* **uart**[aUSBCSWITCHPRO_NUM_UART]
UART Class

Acroname::BrainStem::*MuxClass* **mux**
Mux Class

Acroname::BrainStem::*DigitalClass* **digital**[aUSBCSWITCHPRO_NUM_DIGITALS]
Digital Class

Defines

aUSBCSWITCHPRO_MODULE 16
USBCSwitchPro module number

aUSBCSWITCHPRO_NUM_STORES 3
Number of Store instances available

aUSBCSWITCHPRO_NUM_INTERNAL_SLOTS 12
Store: Number of internal slots instances available

aUSBCSWITCHPRO_NUM_RAM_SLOTS 1

Store: Number of RAM slot instances available

aUSBCSWITCHPRO_NUM_EEPROM_SLOTS 8

Store: Number of EEPROM slot instances available

aUSBCSWITCHPRO_STORE_INTERNAL_INDEX 0

Store: Array index for internal store

aUSBCSWITCHPRO_STORE_RAM_INDEX 1

Store: Array index for RAM store

aUSBCSWITCHPRO_STORE_EEPROM_INDEX 2

Store: Array index for EEPROM store

aUSBCSWITCHPRO_NUM_TEMPERATURES 5

Number of Temperature instances available

aUSBCSWITCHPRO_NUM_USB 1

Number of USB instances available

aUSBCSWITCHPRO_NUM_USB_PORTS 6

Number of USB ports available

aUSBCSWITCHPRO_NUM_PORTS 6

Number of Ports available

aUSBCSWITCHPRO_NUM_PD_PORTS 6

Number of PD compatible ports available

aUSBCSWITCHPRO_NUM_PD_RULES_PER_PORT 7

Number of PD Rules per port available

aUSBCSWITCHPRO_NUM_RAILS 1

Number of Rail instances available

aUSBCSWITCHPRO_NUM_I2C 1

Number of I2C instances available

aUSBCSWITCHPRO_NUM_UART 2

Number of UART instances available

aUSBCSWITCHPRO_NUM_MUX 1

Number of Mux instances available

aUSBCSWITCHPRO_NUM_MUX_CHANNELS 4

Number of Mux channels available

aUSBCSWITCHPRO_NUM_DIGITALS 1

Number of Digital instances available

Port State Defines

Warning: doxygengroup: Cannot find group “aUSBCSwitchPro_Port_State_Defines” in doxygen xml output for project “BrainStem” from directory: doxml/xml

Port State Error Defines

Warning: doxygengroup: Cannot find group “aUSBCSwitchPro_Port_Orientation_Defines” in doxygen xml output for project “BrainStem” from directory: doxml/xml

USBCSwitch

Class

class **aUSBCSwitch** : public Acroname::BrainStem::Module

Concrete Module implementation of a USBCSwitch Allows a user to connect to and control an attached switch.

Public Types

enum **EQUALIZER_3P0_TRANSMITTER_CONFIGS**

Equalizer 3P0 transmitter configs

Values:

enumerator **MUX_1db_COM_0db_900mV**

enumerator **MUX_0db_COM_1db_900mV**

enumerator **MUX_1db_COM_1db_900mV**

enumerator **MUX_0db_COM_0db_900mV**

enumerator **MUX_0db_COM_0db_1100mV**

enumerator **MUX_1db_COM_0db_1100mV**

enumerator **MUX_0db_COM_1db_1100mV**

enumerator **MUX_2db_COM_2db_1100mV**

enumerator **MUX_0db_COM_0db_1300mV**

enum **EQUALIZER_3P0_RECEIVER_CONFIGS**

Equalizer 3P0 receiver configs

Values:

enumerator **LEVEL_1_3P0**

enumerator **LEVEL_2_3P0**

enumerator **LEVEL_3_3P0**

enumerator **LEVEL_4_3P0**

enumerator **LEVEL_5_3P0**

enumerator **LEVEL_6_3P0**

enumerator **LEVEL_7_3P0**

enumerator **LEVEL_8_3P0**

enumerator **LEVEL_9_3P0**

enumerator **LEVEL_10_3P0**

enumerator **LEVEL_11_3P0**

enumerator **LEVEL_12_3P0**

enumerator **LEVEL_13_3P0**

enumerator **LEVEL_14_3P0**

enumerator **LEVEL_15_3P0**

enumerator **LEVEL_16_3P0**

enum **EQUALIZER_2P0_TRANSMITTER_CONFIGS**

Equalizer 2P0 transmitter configs

Values:

enumerator **TRANSMITTER_2P0_40mV**

enumerator **TRANSMITTER_2P0_60mV**

enumerator **TRANSMITTER_2P0_80mV**

enumerator **TRANSMITTER_2P0_0mV**

enum **EQUALIZER_2P0_RECEIVER_CONFIGS**

Equalizer 3P0 receiver configs

Values:

enumerator **LEVEL_1_2P0**

enumerator **LEVEL_2_2P0**

enum **EQUALIZER_CHANNELS**

Equalizer channels

Values:

enumerator **BOTH**

enumerator **MUX**

enumerator **COMMON**

enum **daughtercard_type**

Daughter Cards

Values:

enumerator **NO_DAUGHTERCARD**

enumerator **PASSIVE_DAUGHTERCARD**

enumerator **REDRIVER_DAUGHTERCARD**

enumerator **UNKNOWN_DAUGHTERCARD**

Public Members

Acroname::BrainStem::*MuxClass* **mux**
Mux Class

Acroname::BrainStem::*StoreClass* **store**[aUSBCSWITCH_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class Timer Class

Acroname::BrainStem::*USBCClass* **usb**
USB Class

Acroname::BrainStem::*EqualizerClass* **equalizer**[aUSBCSWITCH_NUM_EQ]
Equalizer Class

Defines

aUSBCSWITCH_MODULE 6
USBCSwitch module number

aUSBCSWITCH_NUM_STORES 2
Number of Store instances available

aUSBCSWITCH_NUM_INTERNAL_SLOTS 12
Store: Number of internal slots instances available

aUSBCSWITCH_NUM_RAM_SLOTS 1
Store: Number of RAM slot instances available

aUSBCSWITCH_NUM_USB 1
Number of USB instances available

aUSBCSWITCH_NUM_MUX 1
Number of Mux instances available

aUSBCSWITCH_NUM_EQ 2
Number of Equalizer instances available

aUSBCSWITCH_NUM_MUX_CHANNELS 4
Number of Mux channels available

Port State Defines

usbPortStateVBUS 0

USB VBUS current state

usbPortStateUSB2A 1

USB2 side A current state

usbPortStateUSB2B 2

USB2 side B current state

usbPortStatesBU 3

SBU current state

usbPortStateSS1 4

SS1 current state

usbPortStateSS2 5

SS2 A current state

usbPortStateCC1 6

CC1 current state

usbPortStateCC2 7

CC2 A current state

usbPortStateCCFlip 14

CC flip status

usbPortStateSSFlip 15

SS flip status

usbPortStatesBUFlip 16

SBU flip status

usbPortStateUSB2Flip 17

USB2 flip status

usbPortStateConnectionEstablished 23

Connection established state

usbPortStateCC1Inject 26

CC1 inject current state

usbPortStateCC2Inject 27

CC2 inject current state

usbPortStateCC1Detect 28

CC1 detect current state

usbPortStateCC2Detect 29

CC2 detect current state

usbPortStateCC1LogicState 30

CC1 logic current state

usbPortStateCC2LogicState 31

CC2 logic current state

get_usbPortStateDaughterCard (var) ((var & (3 << 18)) >> 18)

Daughter card status

Port State Error Defines

Warning: doxygengroup: Cannot find group “aUSBCSwitch_Port_Orientation_Defines” in doxygen xml output for project “BrainStem” from directory: doxml/xml

MTM-DAQ-2

Class

class **aMTMDAQ2** : public Acroname::BrainStem::Module

Concrete Module implementation of an MTM-DAQ-2 Allows a user to connect to and control an attached module.

Public Members

Acroname::BrainStem::AnalogClass **analog**[aMTMDAQ2_NUM_ANALOGS]

Analog Class

Acroname::BrainStem::AppClass **app**[aMTMDAQ2_NUM_APPS]

App Class

Acroname::BrainStem::DigitalClass **digital**[aMTMDAQ2_NUM_DIGITALS]

Digital Class

Acroname::BrainStem::*I2CClass* **i2c**[aMTMDAQ2_NUM_I2C]
I2C Class

Acroname::BrainStem::*PointerClass* **pointer**[aMTMDAQ2_NUM_POINTERS]
Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aMTMDAQ2_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TimerClass* **timer**[aMTMDAQ2_NUM_TIMERS]
Timer Class

Public Static Functions

static inline const std::list<uint8_t> &**getSingleEndedInputRanges** (void)
Get list of analog ranges for single-ended inputs.

Return values

std::list – analog ranges

static inline const std::list<uint8_t> &**getDifferentialInputRanges** (void)
Get list of analog ranges for differential inputs.

Return values

std::list – analog ranges

static inline const std::list<uint8_t> &**getOutputRanges** (void)
Get list of analog range outputs.

Return values

std::list – analog ranges

Defines

aMTMDAQ2_MODULE_BASE_ADDRESS 10
MTM-DAQ-2 module base address

aMTMDAQ2_NUM_ANALOGS 18
Number of Analog instances available

aMTMDAQ2_NUM_ANALOG_INPUTS 16
Analog: Number of Inputs available

aMTMDAQ2_NUM_ANALOG_OUTPUTS 2
Analog: Number of Outputs available

aMTMDAQ2_NUM_APPS 4

Number of App instances available

aMTMDAQ2_BULK_CAPTURE_MAX_HZ 500000

Bulk Capture Max Hertz

aMTMDAQ2_BULK_CAPTURE_MIN_HZ 1

Bulk Capture Min Hertz

aMTMDAQ2_NUM_DIGITALS 2

Number of Digital instances available

aMTMDAQ2_NUM_I2C 1

Number of I2C instances available

aMTMDAQ2_NUM_POINTERS 4

Number of Pointer instances available

aMTMDAQ2_NUM_STORES 2

Number of Store instances available

aMTMDAQ2_NUM_INTERNAL_SLOTS 12

Store: Number of internal slots instances available

aMTMDAQ2_NUM_RAM_SLOTS 1

Store: Number of RAM slot instances available

aMTMDAQ2_NUM_TIMERS 8

Number of Timer instances available

MTM-EtherStem

Class

class **aMTMEtherStem** : public *aMTMStemModule*

Concrete Module implementation of an MTM-EtherStem Allows a user to connect to and control an attached module.

Defines

aMTM_ETHERSTEM_MODULE_BASE_ADDRESS *aMTM_STEM_MODULE_BASE_ADDRESS*

MTM-EtherStem module base address

aMTM_ETHERSTEM_NUM_STORES *aMTM_STEM_NUM_STORES*

Number of Store instances available

aMTM_ETHERSTEM_NUM_STORES *aMTM_STEM_NUM_STORES*

Number of Store instances available

aMTM_ETHERSTEM_NUM_INTERNAL_SLOTS *aMTM_STEM_NUM_INTERNAL_SLOTS*

Store: Number of internal slots instances available

aMTM_ETHERSTEM_NUM_INTERNAL_SLOTS *aMTM_STEM_NUM_INTERNAL_SLOTS*

Store: Number of internal slots instances available

aMTM_ETHERSTEM_NUM_RAM_SLOTS *aMTM_STEM_NUM_RAM_SLOTS*

Store: Number of RAM slot instances available

aMTM_ETHERSTEM_NUM_RAM_SLOTS *aMTM_STEM_NUM_RAM_SLOTS*

Store: Number of RAM slot instances available

aMTM_ETHERSTEM_NUM_SD_SLOTS *aMTM_STEM_NUM_SD_SLOTS*

Store: Number of SD slot instances available

aMTM_ETHERSTEM_NUM_SD_SLOTS *aMTM_STEM_NUM_SD_SLOTS*

Store: Number of SD slot instances available

aMTM_ETHERSTEM_NUM_A2D *aMTM_STEM_NUM_A2D*

Number of Analog instances available

aMTM_ETHERSTEM_NUM_APPS *aMTM_STEM_NUM_APPS*

Number of App instances available

aMTM_ETHERSTEM_BULK_CAPTURE_MAX_HZ *aMTM_STEM_BULK_CAPTURE_MAX_HZ*

Bulk Capture Max Hertz

aMTM_ETHERSTEM_BULK_CAPTURE_MIN_HZ *aMTM_STEM_BULK_CAPTURE_MIN_HZ*

Bulk Capture Min Hertz

aMTM_ETHERSTEM_NUM_CLOCK *aMTM_STEM_NUM_CLOCK*

Number of Clock instances available

aMTM_ETHERSTEM_NUM_DIG *aMTM_STEM_NUM_DIG*

Number of Digital instances available

aMTM_ETHERSTEM_NUM_I2C *aMTM_STEM_NUM_I2C*

Number of I2C instances available

aMTM_ETHERSTEM_NUM_POINTERS *aMTM_STEM_NUM_POINTERS*

Number of Pointer instances available

aMTM_ETHERSTEM_NUM_SERVOS *aMTM_STEM_NUM_SERVOS*

Number of RC Servo instances available

aMTM_ETHERSTEM_NUM_SIGNALS *aMTM_STEM_NUM_SIGNALS*

Number of Signal instances available

aMTM_ETHERSTEM_NUM_OUTPUT_SIGNALS *aMTM_STEM_NUM_OUTPUT_SIGNALS*

Signal: Number of output signal instances available

aMTM_ETHERSTEM_NUM_INPUT_SIGNALS *aMTM_STEM_NUM_INPUT_SIGNALS*

Signal: Number of input signal instances available

aMTM_ETHERSTEM_NUM_TIMERS *aMTM_STEM_NUM_TIMERS*

Number of Timer instances available

MTM-IO-Serial

Class

class **aMTMIOSerial** : public Acroname::BrainStem::Module

Concrete Module implementation of an MTM-IO-Serial Allows a user to connect to and control an attached module.

Public Types

enum **PORT_ID**

Port ID

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_UP0**

typedef enum *aMTMIOSerial::PORT_ID* **PORT_ID_t**
Port ID

Public Members

HubClass **hub**

Hub Class

Acroname::BrainStem::*AppClass* **app**[aMTMIOSERIAL_NUM_APPS]

App Class

Acroname::BrainStem::*DigitalClass* **digital**[aMTMIOSERIAL_NUM_DIGITALS]

Digital Class

Acroname::BrainStem::*I2CClass* **i2c**[aMTMIOSERIAL_NUM_I2C]

I2C Class

Acroname::BrainStem::*UARTClass* **uart**[aMTMIOSERIAL_NUM_UART]

UART Class

Acroname::BrainStem::*PointerClass* **pointer**[aMTMIOSERIAL_NUM_POINTERS]

Pointer Class

Acroname::BrainStem::*RailClass* **rail**[aMTMIOSERIAL_NUM_RAILS]

Rail Class

Acroname::BrainStem::*RCServoClass* **servo**[aMTM_STEM_NUM_SERVOS]

RC Servo Class

Acroname::BrainStem::*SignalClass* **signal**[aMTMIOSERIAL_NUM_SIGNALS]

Signal Class

Acroname::BrainStem::*StoreClass* **store**[aMTMIOSERIAL_NUM_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TemperatureClass* **temperature**

Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aMTMIO SERIAL_NUM_TIMERS]
Timer Class

Acroname::BrainStem::*USBClass* **usb**
USB Class

class **HubClass** : public Acroname::BrainStem::*USBSystemClass*
Hub class implementation for use with MTMIOSerial.

Defines

aMTMIO SERIAL_MODULE_BASE_ADDRESS 8
MTM-IO-Serial module number

aMTMIO SERIAL_NUM_APPS 4
Number of App instances available

aMTMIO SERIAL_NUM_DIGITALS 8
Number of Digital instances available

aMTMIO SERIAL_NUM_I2C 1
Number of I2C instances available

aMTMIO SERIAL_NUM_POINTERS 4
Number of Pointer instances available

aMTMIO SERIAL_NUM_RAILS 3
Number of Rail instances available

aMTMIO SERIAL_5VRAIL 0
Rail: 5v Rail specifier

aMTMIO SERIAL_ADJRAIL1 1
Rail: Adjustable Rail 0 specifier

aMTMIO SERIAL_ADJRAIL2 2
Rail: Adjustable Rail 1 specifier

aMTMIO SERIAL_MAX_MICROVOLTAGE 5000000
Rail: Max voltage in microvolts

aMTMIO SERIAL_MIN_MICROVOLTAGE 1800000
Rail: Min voltage in microvolts

aMTMIO SERIAL_NUM_SERVOS 8

Number of RC Servo instances available

aMTMIO SERIAL_NUM_SIGNALS 5

Number of Signal instances available

aMTMIO SERIAL_NUM_OUTPUT_SIGNALS 4

Signal: Number of output signal instances available

aMTMIO SERIAL_NUM_INPUT_SIGNALS 5

Signal: Number of input signal instances available

aMTMIO SERIAL_NUM_STORES 2

Number of Store instances available

aMTMIO SERIAL_NUM_INTERNAL_SLOTS 12

Store: Number of internal slots instances available

aMTMIO SERIAL_NUM_RAM_SLOTS 1

Store: Number of RAM slot instances available

aMTMIO SERIAL_NUM_TIMERS 8

Number of Timer instances available

aMTMIO SERIAL_NUM_UART 4

Number of UART instances available

aMTMIO SERIAL_NUM_USB 1

Number of USB instances available

aMTMIO SERIAL_NUM_USB_PORTS 4

Number of USB ports available

aMTMIO SERIAL_NUM_PORTS 5

Number of Ports available

aMTMIO SERIAL_USB_NUM_CHANNELS 4

Number of channels available

aUSB_UPSTREAM_CONFIG_AUTO 0

Upstream Mode specifier: Auto (Default)

aUSB_UPSTREAM_CONFIG_ONBOARD 1

Upstream Mode specifier: Onboard

aUSB_UPSTREAM_CONFIG_EDGE 2

Upstream Mode specifier: Edge Connector

aUSB_UPSTREAM_ONBOARD 0

Upstream State specifier: Onboard

aUSB_UPSTREAM_EDGE 1

Upstream State specifier: Edge Connector

Port State Defines

aMTMIO SERIAL_USB_VBUS_ENABLED 0

USB VBUS current state

aMTMIO SERIAL_USB2_DATA_ENABLED 1

USB2 data current state

aMTMIO SERIAL_USB_ERROR_FLAG 19

Error indicator for this channel

(see 'Port Errors' below)

aMTMIO SERIAL_USB2_BOOST_ENABLED 20

USB2 boost current state

Port State Error Defines

aMTMIO SERIAL_ERROR_VBUS_OVERCURRENT 0

VBUS overcurrent error

MTM-Load-1

Class

class **aMTMLoad1** : public Acroname::BrainStem::Module

Concrete Module implementation of an MTM-Load-1 Allows a user to connect to and control an attached module.

Public Members

Acroname::BrainStem::*AppClass* **app**[aMTMLOAD1_NUM_APPS]
App Class

Acroname::BrainStem::*DigitalClass* **digital**[aMTMLOAD1_NUM_DIGITALS]
Digital Class

Acroname::BrainStem::*I2CClass* **i2c**[aMTMLOAD1_NUM_I2C]
I2C Class

Acroname::BrainStem::*PointerClass* **pointer**[aMTMLOAD1_NUM_POINTERS]
Pointer Class

Acroname::BrainStem::*RailClass* **rail**[aMTMLOAD1_NUM_RAILS]
Rail Class

Acroname::BrainStem::*StoreClass* **store**[aMTMLOAD1_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass* **temperature**
Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aMTMLOAD1_NUM_TIMERS]
Timer Class

Defines

aMTMLOAD1_MODULE_BASE_ADDRESS 14
MTM-Load-1 module base address

aMTMLOAD1_NUM_APPS 4
Number of App instances available

aMTMLOAD1_NUM_DIGITALS 4
Number of Digital instances available

aMTMLOAD1_NUM_I2C 1
Number of I2C instances available

aMTMLOAD1_NUM_POINTERS 4
Number of Pointer instances available

aMTMLOAD1_NUM_RAILS 1

Rail: Number of Rail instances available

aMTMLOAD1_RAIL0 0

Rail: Define for Rail 0

aMTMLOAD1_MAX_MICROVOLTAGE 32000000

Rail: Max voltage in microvolts

aMTMLOAD1_MIN_MICROVOLTAGE 0

Rail: Min voltage in microvolts

aMTMLOAD1_MAX_MICROAMPS 11000000

Rail: Max current in microamps

aMTMLOAD1_MIN_MICROAMPS 0

Rail: Min current in microamps

aMTMLOAD1_MAX_MILLIWATTS 150000

Rail: Max power in milliwatts

aMTMLOAD1_MIN_MILLIWATTS 0

Rail: Min power in milliwatts

aMTMLOAD1_MAX_MICROOHMS 1000000000

Rail: Max resistance in microohms

aMTMLOAD1_MIN_MICROOHMS 0

Rail: Min resistance in microohms

aMTMLOAD1_MAX_VOLTAGE_LIMIT_MICROVOLTS 35000000

Rail: Max voltage limit in microvolts

aMTMLOAD1_MIN_VOLTAGE_LIMIT_MICROVOLTS -700000

Rail: Min voltage limit in microvolts

aMTMLOAD1_MAX_CURRENT_LIMIT_MICROAMPS 12000000

Rail: Max current limit in microamps

aMTMLOAD1_MIN_CURRENT_LIMIT_MICROAMPS -1000000

Rail: Min current limit in microamps

aMTMLOAD1_MAX_POWER_LIMIT_MILLIWATTS 150000

Rail: Max power limit in milliwatts

aMTMLOAD1_MIN_POWER_LIMIT_MILLIWATTS 0

Rail: Min power limit in milliwatts

aMTMLOAD1_NUM_STORES 2

Number of Store instances available

aMTMLOAD1_NUM_INTERNAL_SLOTS 12

Store: Number of internal slots instances available

aMTMLOAD1_NUM_RAM_SLOTS 1

Store: Number of RAM slot instances available

aMTMLOAD1_NUM_TEMPERATURES 1

Number of Temperature instances available

aMTMLOAD1_NUM_TIMERS 8

Number of Timer instances available

MTM-PM-1

Class

class **aMTMPM1** : public Acroname::BrainStem::Module

Concrete Module implementation of an MTM-PM-1 Allows a user to connect to and control an attached module.

Public Members

Acroname::BrainStem::AppClass **app**[aMTMPM1_NUM_APPS]

App Class

Acroname::BrainStem::DigitalClass **digital**[aMTMPM1_NUM_DIGITALS]

Digital Class

Acroname::BrainStem::I2CClass **i2c**[aMTMPM1_NUM_I2C]

I2C Class

Acroname::BrainStem::PointerClass **pointer**[aMTMPM1_NUM_POINTERS]

Pointer Class

Acroname::BrainStem::RailClass **rail**[aMTMPM1_NUM_RAILS]

Rail Class

Acroname::BrainStem::*StoreClass* **store**[aMTMPM1_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass* **temperature**
Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aMTMPM1_NUM_TIMERS]
Timer Class

Defines

aMTMPM1_MODULE_BASE_ADDRESS 6
MTM-PM-1 module base address

aMTMPM1_NUM_APPS 4
Number of App instances available

aMTMPM1_NUM_DIGITALS 2
Number of Digital instances available

aMTMPM1_NUM_I2C 1
Number of I2C instances available

aMTMPM1_NUM_POINTERS 4
Number of Pointer instances available

aMTMPM1_NUM_RAILS 2
Number of Rail instances available

aMTMPM1_RAIL0 0
Rail: Define for Rail 0

aMTMPM1_RAIL1 1
Rail: Define for Rail 1

aMTMPM1_MAX_MICROVOLTAGE 5000000
Rail: Max voltage in microvolts

aMTMPM1_MIN_MICROVOLTAGE 1800000
Rail: Min voltage in microvolts

aMTMPM1_MAX_CURRENT_LIMIT_MICROAMPS 3000000

Rail: Max current in microamps

aMTMPM1_MIN_CURRENT_LIMIT_MICROAMPS 0

Rail: Min current in microamps

aMTMPM1_NUM_STORES 2

Number of Store instances available

aMTMPM1_NUM_INTERNAL_SLOTS 12

Store: Number of internal slots instances available

aMTMPM1_NUM_RAM_SLOTS 1

Store: Number of RAM slot instances available

aMTMPM1_NUM_TEMPERATURES 1

Number of Temperature instances available

aMTMPM1_NUM_TIMERS 8

Number of Timer instances available

MTM-Relay

Class

class **aMTMRelay** : public Acroname::BrainStem::Module

Concrete Module implementation of an MTM-Relay Allows a user to connect to and control an attached module.

Public Members

Acroname::BrainStem::AppClass **app**[aMTMRELAY_NUM_APPS]

App Class

Acroname::BrainStem::DigitalClass **digital**[aMTMRELAY_NUM_DIGITALS]

Digital Class

Acroname::BrainStem::I2CClass **i2c**[aMTMRELAY_NUM_I2C]

I2C Class

Acroname::BrainStem::PointerClass **pointer**[aMTMRELAY_NUM_POINTERS]

Pointer Class

Acroname::BrainStem::*RelayClass* **relay**[aMTMRELAY_NUM_RELAYS]
Relay Class

Acroname::BrainStem::*StoreClass* **store**[aMTMRELAY_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TimerClass* **timer**[aMTMRELAY_NUM_TIMERS]
Timer Class

Defines

aMTMRELAY_MODULE_BASE_ADDRESS 12
MTM-RELAY module base address

aMTMRELAY_NUM_APPS 4
Number of App instances available

aMTMRELAY_NUM_DIGITALS 4
Number of Digital instances available

aMTMRELAY_NUM_I2C 1
Number of I2C instances available

aMTMRELAY_NUM_POINTERS 4
Number of Pointer instances available

aMTMRELAY_NUM_RELAYS 4
Number of Rail instances available

aMTMRELAY_NUM_STORES 2
Number of Store instances available

aMTMRELAY_NUM_INTERNAL_SLOTS 12
Store: Number of internal slots instances available

aMTMRELAY_NUM_RAM_SLOTS 1
Store: Number of RAM slot instances available

aMTMRELAY_NUM_TIMERS 8
Number of Timer instances available

MTM-USBStem

Class

class **aMTMUSBStem** : public *aMTMStemModule*

Concrete Module implementation of an MTM-USBStem Allows a user to connect to and control an attached module.

Defines

aMTM_USBSTEM_MODULE_BASE_ADDRESS *aMTM_STEM_MODULE_BASE_ADDRESS*

MTM-USBStem module base address

aMTM_USBSTEM_NUM_A2D *aMTM_STEM_NUM_A2D*

Number of Analog instances available

aMTM_USBSTEM_NUM_APPS *aMTM_STEM_NUM_APPS*

Number of App instances available

aMTM_USBSTEM_BULK_CAPTURE_MAX_HZ *aMTM_STEM_BULK_CAPTURE_MAX_HZ*

Bulk Capture Max Hertz

aMTM_USBSTEM_BULK_CAPTURE_MIN_HZ *aMTM_STEM_BULK_CAPTURE_MIN_HZ*

Bulk Capture Min Hertz

aMTM_USBSTEM_NUM_CLOCK *aMTM_STEM_NUM_CLOCK*

Number of Clock instances available

aMTM_USBSTEM_NUM_DIG *aMTM_STEM_NUM_DIG*

Number of Digital instances available

aMTM_USBSTEM_NUM_I2C *aMTM_STEM_NUM_I2C*

Number of I2C instances available

aMTM_USBSTEM_NUM_POINTERS *aMTM_STEM_NUM_POINTERS*

Number of Pointer instances available

aMTM_USBSTEM_NUM_SERVOS *aMTM_STEM_NUM_SERVOS*

Number of RC Servo instances available

aMTM_USBSTEM_NUM_SIGNALS *aMTM_STEM_NUM_SIGNALS*

Number of Signal instances available

aMTM_USBSTEM_NUM_OUTPUT_SIGNALS *aMTM_STEM_NUM_OUTPUT_SIGNALS*

Signal: Number of output signal instances available

aMTM_USBSTEM_NUM_INPUT_SIGNALS *aMTM_STEM_NUM_INPUT_SIGNALS*

Signal: Number of input signal instances available

aMTM_USBSTEM_NUM_STORES *aMTM_STEM_NUM_STORES*

Number of Store instances available

aMTM_USBSTEM_NUM_INTERNAL_SLOTS *aMTM_STEM_NUM_INTERNAL_SLOTS*

Store: Number of internal slots instances available

aMTM_USBSTEM_NUM_RAM_SLOTS *aMTM_STEM_NUM_RAM_SLOTS*

Store: Number of RAM slot instances available

aMTM_USBSTEM_NUM_SD_SLOTS *aMTM_STEM_NUM_SD_SLOTS*

Store: Number of SD slot instances available

aMTM_USBSTEM_NUM_TIMERS *aMTM_STEM_NUM_TIMERS*

Number of Timer instances available

MTM-Stem

Class

class **aMTMStemModule** : public Acroname::BrainStem::Module

Instantiation of base class MTM-Stem-Module.

Subclassed by *aMTMEtherStem*, *aMTMUSBStem*

Public Members

Acroname::BrainStem::AnalogClass **analog**[aMTM_STEM_NUM_A2D]

Analog Class

Acroname::BrainStem::AppClass **app**[aMTM_STEM_NUM_APPS]

App Class

Acroname::BrainStem::ClockClass **clock**

Clock Class

Acroname::BrainStem::DigitalClass **digital**[aMTM_STEM_NUM_DIG]

Digital Class

Acroname::BrainStem::I2CClass **i2c**[aMTM_STEM_NUM_I2C]

I2C Class

Acroname::BrainStem::*PointerClass* **pointer**[aMTM_STEM_NUM_POINTERS]
 Pointer Class

Acroname::BrainStem::*RCServoClass* **servo**[aMTM_STEM_NUM_SERVOS]
 RC Servo Class

Acroname::BrainStem::*SignalClass* **signal**[aMTM_STEM_NUM_SIGNALS]
 Signal Class

Acroname::BrainStem::*StoreClass* **store**[aMTM_STEM_NUM_STORES]
 Store Class

Acroname::BrainStem::*SystemClass* **system**
 System Class

Acroname::BrainStem::*TimerClass* **timer**[aMTM_STEM_NUM_TIMERS]
 Timer Class

Defines

aMTM_STEM_MODULE_BASE_ADDRESS 4
 MTM-Stem module base address

aMTM_STEM_NUM_A2D 4
 Number of Analog instances available

aMTM_STEM_NUM_APPS 4
 Number of App instances available

aMTM_STEM_BULK_CAPTURE_MAX_HZ *analog_Hz_Maximum*
 Bulk Capture Max Hertz: 200000

aMTM_STEM_BULK_CAPTURE_MIN_HZ *analog_Hz_Minimum*
 Bulk Capture Min Hertz: 7000

aMTM_STEM_NUM_CLOCK 1
 Number of Clock instances available

aMTM_STEM_NUM_DIG 15
 Number of Digital instances available

aMTM_STEM_NUM_I2C 2
 Number of I2C instances available

aMTM_STEM_NUM_POINTERS 4

Number of Pointer instances available

aMTM_STEM_NUM_SERVOS 8

Number of RC Servo instances available

aMTM_STEM_NUM_SIGNALS 5

Number of Signal instances available

aMTM_STEM_NUM_OUTPUT_SIGNALS 4

Signal mber of output signal instances available

aMTM_STEM_NUM_INPUT_SIGNALS 5

Signal mber of input signal instances available

aMTM_STEM_NUM_STORES 3

Number of Store instances available

aMTM_STEM_NUM_INTERNAL_SLOTS 12

Store mber of internal slots instances available

aMTM_STEM_NUM_RAM_SLOTS 1

Store mber of RAM slot instances available

aMTM_STEM_NUM_SD_SLOTS 255

Store mber of SD slot instances available

aMTM_STEM_NUM_TIMERS 8

Number of Timer instances available

3.3.2 Entity Class

class **EntityClass**

EntityClass: The *EntityClass* is the base class for interacting with BrainStem UEI entities. All BrainStem UEI classes inherit from *EntityClass*. Advanced users may use *EntityClass* to extend BrainStem functionality specific to their needs.

Subclassed	by	<i>Acroname::BrainStem::AnalogClass,</i>	<i>Acron-</i>	
		<i>ame::BrainStem::AppClass,</i>	<i>ame::BrainStem::ClockClass,</i>	<i>Acron-</i>
		<i>ame::BrainStem::DigitalClass,</i>	<i>Acroname::BrainStem::EqualizerClass,</i>	
		<i>Acroname::BrainStem::Ethernet::Utilities::EthernetClassProperties,</i>	<i>Acron-</i>	
		<i>ame::BrainStem::EthernetClass,</i>	<i>Acroname::BrainStem::FactoryClass,</i>	<i>Acron-</i>
		<i>ame::BrainStem::HDBaseTClass,</i>	<i>Acroname::BrainStem::I2CClass,</i>	<i>Acron-</i>
		<i>ame::BrainStem::MuxClass,</i>	<i>Acroname::BrainStem::PoEClass,</i>	<i>Acron-</i>
		<i>ame::BrainStem::PointerClass,</i>	<i>Acroname::BrainStem::PortClass,</i>	<i>Acron-</i>
		<i>ame::BrainStem::PowerDeliveryClass,</i>	<i>Acroname::BrainStem::RailClass,</i>	<i>Acron-</i>
		<i>ame::BrainStem::RCServoClass,</i>	<i>Acroname::BrainStem::RelayClass,</i>	<i>Acron-</i>

<i>ame::BrainStem::SignalClass,</i>	<i>Acroname::BrainStem::StoreClass,</i>	<i>Acron-</i>
<i>ame::BrainStem::SystemClass,</i>	<i>Acroname::BrainStem::TemperatureClass,</i>	<i>Acron-</i>
<i>ame::BrainStem::TimerClass,</i>	<i>Acroname::BrainStem::UARTClass,</i>	<i>Acron-</i>
<i>ame::BrainStem::USBClass,</i>	<i>Acroname::BrainStem::USBSystemClass</i>	

Public Functions

EntityClass (void)

Constructor.

virtual **~EntityClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t command, const uint8_t index)
init.

Initialize the entity class.

Parameters

- **pModule** – The BrainStem module object.
- **command** – The command of the UEI.
- **index** – The index of the UEI entity.

aErr **callUEI** (const uint8_t option)

A callUEI is a setUEI that has no data length.

Parameters

option – An option for the UEI.

Returns

Returns *common entity* return values

aErr **setUEI8** (const uint8_t option, const uint8_t byteValue)

Set a byte value.

Parameters

- **option** – The option for the UEI.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI8** (const uint8_t option, const uint8_t param, const uint8_t byteValue)

Set a byte value with a subindex.

Parameters

- **option** – The option for the UEI.
- **param** – of the option.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **getUEI8** (const uint8_t option, uint8_t *byteValue)

Get a byte value.

Parameters

- **option** – The option for the UEI.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **getUEI8** (const uint8_t option, const uint8_t param, uint8_t *byteValue)

Get a byte value with a parameter.

Parameters

- **option** – The option for the UEI.
- **param** – The parameter.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI16** (const uint8_t option, const uint16_t shortValue)

Set a 2-byte value.

Parameters

- **option** – The option for the UEI.
- **shortValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI16** (const uint8_t option, const uint8_t param, const uint16_t shortValue)

Set a 2-byte value with a parameter.

Parameters

- **option** – The option for the UEI.
- **param** – The parameter.
- **shortValue** – The value.

Returns

Returns *common entity* return values

aErr **getUEI16** (const uint8_t option, uint16_t *shortValue)

Get a 2-byte value.

Parameters

- **option** – The option for the UEI.
- **shortValue** – The value.

Returns

Returns *common entity* return values

aErr **getUEI16** (const uint8_t option, const uint8_t param, uint16_t *shortValue)

Get a 2-byte value with a parameter.

Parameters

- **option** – The option for the UEI.
- **param** – The parameter.
- **shortValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI32** (const uint8_t option, const uint32_t intValue)

Set a 4-byte value.

Parameters

- **option** – The option for the UEI.
- **intValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI32** (const uint8_t option, const uint8_t subIndex, const uint32_t intValue)

Set a 4-byte value, with a subindex parameter.

Parameters

- **option** – The option for the UEI.

- **subIndex** – The subindex to set.
- **intValue** – The value.

Returns

Returns *common entity* return values

aErr **getUEI32** (const uint8_t option, uint32_t *intValue)

Get a 4-byte value.

Parameters

- **option** – The option for the UEI.
- **intValue** – The 4 byte value

Returns

Returns *common entity* return values

aErr **getUEI32** (const uint8_t option, const uint8_t param, uint32_t *intValue)

Get a 4-byte value with parameter.

Parameters

- **option** – The option for the UEI.
- **param** – The parameter.
- **intValue** – The 4 byte value

Returns

Returns *common entity* return values

aErr **setUEIBytes** (const uint8_t option, const uint8_t *bufPtr, const size_t bufLen)

Set a multi-byte value.

Parameters

- **option** – The option for the UEI.
- **bufPtr** – The pointer to a data buffer
- **bufLen** – The length of the data buffer

Returns

Returns *common entity* return values

aErr **getUEIBytes** (const uint8_t option, uint8_t *buf, const size_t bufLength, size_t *unloadedLength)

Unloads UEI Bytes data as byte data

Parameters

- **option** – The option for the UEI.
- **buf** – Start of where data should be stored..
- **bufLength** – Size of the buffer
- **unloadedLength** – Amount of data unloaded (in bytes)

Returns

Returns *common entity* return values

aErr **getUEIBytesCheck** (size_t *unloadedLength, const size_t valueSize)

Parameters

- **unloadedLength** – Amount of data unloaded (in bytes)
- **valueSize** – The base type size in this array

Returns

Returns *common entity* return values

uint8_t **getIndex** (void) const

Get the UEI entity index.

Returns

The 1 byte index of the UEI entity.

aErr **drainUEI** (const uint8_t option)

Drain all packets matching this UEI from the packet fifo.

This functionality is useful in rare cases where packet synchronization is lost and a valid return packet is not accessible.

aErr **setStreamEnabled** (uint8_t enabled)

Enables streaming for all possible option codes within the cmd and index the entity was created for.

Parameters

enabled – The state to be applied. 0 = Disabled; 1 = enabled

Returns

Returns *common entity* return values

aErr **registerOptionCallback** (const uint8_t option, const bool enable,
Link::streamCallback_t cb, void *pRef)

Registers a callback function based on a specific option code. Option code applies to the cmd and index of the called API.

Parameters

- **option** – option to filter by (supports Wildcards)
- **enable** – True - installs/updates callback and ref; False - uninstalls callback
- **cb** – Callback to be executed when a new packet matching the criteria is received.
- **pRef** – Pointer to user reference for use inside the callback function.

Returns

aErrNotFound - Item not found (uninstalling only)

Returns

aErrNone - success

aErr **getStreamStatus** (*Link::StreamStatusEntry_t* *buffer, const size_t bufferLength,
size_t *unloadedSize)

Gets all available stream values associated with the cmd and index of the called API. Keys can be decoded with *Link::getStreamKeyElement*.

Parameters

- **buffer** – Buffer of user allocated memory to be filled with stream data
- **bufferLength** – Number of elements the buffer can hold.
- **unloadedSize** – Number of elements that were placed in the buffer

Returns

aErrParam if status or unloadedSize is null

Returns

aErrResource - if the link is not valid

Returns

aErrNone - success

aErr **resetEntityToFactoryDefaults** (void)

Resets the Entity to factory defaults

Returns

Returns *common entity* return values

aErr **loadEntityFromStore** (void)

Load the Entity from the internal store

Returns

Returns *common entity* return values

aErr **saveEntityToStore** (void)

Saves the Entity to the internal store

Returns

Returns *common entity* return values

Public Static Functions

static uint8_t **getUEIBytesSequence** (const *aPacket* *packet)

Parameters

packet – UEI packet to be checked/filtered.

Returns

The sequence number of the byte.

static bool **getUEIBytesContinue** (const *aPacket* *packet)

Parameters

packet – UEI packet to be checked/filtered.

Returns

True - Continue bit is set (more packets to come); False - Continue bit is not set (first or last packet).

static uint8_t **sUEIBytesFilter** (const *aPacket* *packet, const void *ref)

Filter function for UEI Bytes calls. Exposed for unit-testing purposes only.

Parameters

- **packet** – UEI packet to be checked/filtered.
- **ref** – Opaque reference handle

class impl

group EntityReturnValues

Common *EntityClass* Return Values.

- *aErrNone* - Action completed successfully.
- *aErrTimeout* - Request timed out without response.
- *aErrConnection* - No active link.

3.3.3 Analog Class

See the *Analog Entity* for generic information.

class **AnalogClass** : public Acroname::BrainStem::EntityClass

AnalogClass: Interface to analog entities on BrainStem modules. Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

Public Functions

AnalogClass (void)

Constructor.

virtual **~AnalogClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Analog Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Analog entity to be addressed.

aErr **getValue** (uint16_t *value)

Get the raw ADC output value in bits.

Note: Not all modules provide 16 useful bits; this value's least significant bits are zero-padded to 16 bits. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Parameters

value – 16 bit analog reading with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

Returns

Returns *common entity* return values

aErr **getVoltage** (int32_t *microvolts)

Get the scaled micro volt value with reference to ground.

Note: Not all modules provide 32 bits of accuracy. Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

microvolts – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns *common entity* return values

aErr **getRange** (uint8_t *range)

Get the analog input range.

Parameters

range – 8 bit value corresponding to a discrete range option

Returns

Returns *common entity* return values

aErr **getEnable** (uint8_t *enable)

Get the analog output enable status.

Parameters

enable – 0 if disabled 1 if enabled.

Returns

Returns *common entity* return values

aErr **setValue** (const uint16_t value)

Set the value of an analog output (DAC) in bits.

Note: Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Parameters

value – 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

Returns

Returns *common entity* return values

aErr **setVoltage** (const int32_t microvolts)

Set the voltage level of an analog output (DAC) in microvolts.

Note: Voltage range is dependent on the specific DAC channel range.

Parameters

microvolts – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns *common entity* return values

aErr **setRange** (const uint8_t range)

Set the analog input range.

Parameters

range – 8 bit value corresponding to a discrete range option

Returns

Returns *common entity* return values

aErr **setEnabled** (const uint8_t enable)

Set the analog output enable state.

Parameters

enable – set 1 to enable or 0 to disable.

Returns

Returns *common entity* return values

***aErr* setConfiguration** (const uint8_t configuration)

Set the analog configuration.

Parameters

configuration – bitAnalogConfigurationOutput configures the analog entity as an output.

Returns

Returns *common entity* return values

***aErr* getConfiguration** (uint8_t *configuration)

Get the analog configuration.

Parameters

configuration – Current configuration of the analog entity.

Returns

Returns *common entity* return values

***aErr* setBulkCaptureSampleRate** (const uint32_t value)

Set the sample rate for this analog when bulk capturing.

Parameters

value – sample rate in samples per second (Hertz).

- Minimum rate: 7,000 Hz
- Maximum rate: 200,000 Hz

Returns

Returns *common entity* return values

***aErr* getBulkCaptureSampleRate** (uint32_t *value)

Get the current sample rate setting for this analog when bulk capturing.

Parameters

value – upon success filled with current sample rate in samples per second (Hertz).

Returns

Returns *common entity* return values

***aErr* setBulkCaptureNumberOfSamples** (const uint32_t value)

Set the number of samples to capture for this analog when bulk capturing.

Parameters

value – number of samples.

- Minimum # of Samples: 0
- Maximum # of Samples: (BRAINSTEM_RAM_SLOT_SIZE / 2) = (3FFF / 2) = 1FFF = 8191

Returns

Returns *common entity* return values

***aErr* getBulkCaptureNumberOfSamples** (uint32_t *value)

Get the current number of samples setting for this analog when bulk capturing.

Parameters

value – number of samples.

Returns

Returns *common entity* return values

aErr **initiateBulkCapture** (void)

Initiate a BulkCapture on this analog. Captured measurements are stored in the module's RAM store (RAM_STORE) slot 0. Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

Note: When the bulk capture is complete *getBulkCaptureState()* will return either bulkCaptureFinished or bulkCaptureError.

Returns

Returns *common entity* return values

aErr **getBulkCaptureState** (uint8_t *state)

Get the current bulk capture state for this analog.

Parameters

state – the state of bulk capture.

- Idle: bulkCaptureIdle = 0
- Pending: bulkCapturePending = 1
- Finished: bulkCaptureFinished = 2
- Error: bulkCaptureError = 3

Returns

Returns *common entity* return values

3.3.4 App Class

See the *App Entity* for generic information.

class **AppClass** : public Acroname::BrainStem::EntityClass

AppClass: Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

Public Functions

AppClass (void)

Constructor.

virtual **~AppClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the App Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the App entity to be addressed.

aErr **execute** (const uint32_t appParam)

Execute the app reflex on the module. Doesn't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

Parameters

appParam – The app parameter handed to the reflex.

Return values

- **aErrNone** – success.
- **aErrTimeout** – The request timed out waiting to start execution.
- **aErrConnection** – No active link connection.
- **aErrNotFound** – the app reflex was not found or not enabled on the module.

aErr **execute** (const uint32_t appParam, uint32_t *returnVal, const uint32_t msTimeout = 1000)

Execute the app reflex on the module. Waits for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

Parameters

- **appParam** – The app parameter handed to the reflex.
- **returnVal** – The return value filled in from the result of executing the reflex routine.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

Return values

- **aErrNone** – - success.
- **aErrTimeout** – - The request timed out waiting for a response.
- **aErrConnection** – - No active link connection.
- **aErrNotFound** – - the app reflex was not found or not enabled on the module.

3.3.5 Clock Class

See the *Clock Entity* for generic information.

class **ClockClass** : public Acroname::BrainStem::EntityClass

ClockClass: Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

Note: Clock time must be reset if power to the BrainStem module is lost.

Public Functions

ClockClass (void)

Constructor.

virtual **~ClockClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Clock Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Clock entity to be addressed.

aErr **getYear** (uint16_t *year)

Get the four digit year value (0-4095).

Parameters

year – Get the year portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setYear** (const uint16_t year)

Set the four digit year value (0-4095).

Parameters

year – Set the year portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getMonth** (uint8_t *month)

Get the two digit month value (1-12).

Parameters

month – The two digit month portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setMonth** (const uint8_t month)

Set the two digit month value (1-12).

Parameters

month – The two digit month portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getDay** (uint8_t *day)

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Parameters

day – The two digit day portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setDay** (const uint8_t day)

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Parameters

day – The two digit day portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getHour** (uint8_t *hour)

Get the two digit hour value (0-23).

Parameters

hour – The two digit hour portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setHour** (const uint8_t hour)

Set the two digit hour value (0-23).

Parameters

hour – The two digit hour portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getMinute** (uint8_t *minute)

Get the two digit minute value (0-59).

Parameters

minute – The two digit minute portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setMinute** (const uint8_t minute)

Set the two digit minute value (0-59).

Parameters

minute – The two digit minute portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getSecond** (uint8_t *second)

Get the two digit second value (0-59).

Parameters

second – The two digit second portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setSecond** (const uint8_t second)

Set the two digit second value (0-59).

Parameters

second – The two digit second portion of the real-time clock value.

Returns

Returns *common entity* return values

3.3.6 Digital Class

See the *Digital Entity* for generic information.

class **DigitalClass** : public Acroname::BrainStem::EntityClass

DigitalClass: Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

Public Functions

DigitalClass (void)

Constructor.

virtual ~**DigitalClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Digital Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Digital entity to be addressed.

aErr **setConfiguration** (const uint8_t configuration)

Set the digital configuration to one of the available 5 states.

Note: Some configurations are only supported on specific pins.

Parameters

configuration – The configuration to be applied

- Digital Input: digitalConfigurationInput = 0
- Digital Output: digitalConfigurationOutput = 1
- RCServo Input: digitalConfigurationRCServoInput = 2
- RCServo Output: digitalConfigurationRCServoOutput = 3
- High Z State: digitalConfigurationHiZ = 4
- Digital Input: digitalConfigurationInputPullUp = 0
- Digital Input: digitalConfigurationInputNoPull = 4
- Digital Input: digitalConfigurationInputPullDown = 5

Returns

Returns *common entity* return values

aErr **getConfiguration** (uint8_t *configuration)

Get the digital configuration.

Parameters

configuration – Current configuration of the digital entity.

Returns

Returns *common entity* return values

aErr **setState** (const uint8_t state)

Set the logical state.

Parameters

state – The state to be set. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **getState** (uint8_t *state)

Get the state.

Note: If in high Z state an error will be returned.

Parameters

state – The current state of the digital entity. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **setStateAll** (const uint32_t state)

Sets the logical state of all available digitals based on the bit mapping. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

Parameters

state – The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc.

Returns

Returns *common entity* return values

aErr **getStateAll** (uint32_t *state)

Gets the logical state of all available digitals in a bit mapped representation. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

Parameters

state – The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **setValue** (const uint8_t value)

Set the expected value of the digital pin.

Parameters

value – The expected value of the digital pin. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **getValue** (uint8_t *value)

Get the expected value of the digital pin.

Parameters

value – The expected value of the digital pin. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **setLinkChannel** (const uint8_t linkChannel)

Set the link channel (entity index) for linking digital entities. Two digital entities will link when one is configured as LinkInput, one as LinkOutput, and both have each others linkChannel indexes.

Parameters

linkChannel – The link channel (entity index) value. Entities with matching linkChannel values can be linked.

Returns

Returns *common entity* return values

aErr **getLinkChannel** (uint8_t *linkChannel)

Get the link channel (entity index) for linking digital entities.

Parameters

linkChannel – The current link channel (entity index) value.

Returns

Returns *common entity* return values

3.3.7 Equalizer Class

See the *Equalizer Entity* for generic information.

class **EqualizerClass** : public Acroname::BrainStem::EntityClass

EqualizerClass: Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

Public Functions

EqualizerClass (void)

Constructor.

virtual **~EqualizerClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Equalizer Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Equalizer entity to be addressed.

aErr **setReceiverConfig** (const uint8_t channel, const uint8_t config)

Sets the receiver configuration for a given channel.

Parameters

- **channel** – The equalizer receiver channel.
- **config** – Configuration to be applied to the receiver.

Returns

Returns *common entity* return values

aErr **getReceiverConfig** (const uint8_t channel, uint8_t *config)

Gets the receiver configuration for a given channel.

Parameters

- **channel** – The equalizer receiver channel.
- **config** – Configuration of the receiver.

Returns

Returns *common entity* return values

aErr **setTransmitterConfig** (const uint8_t config)

Sets the transmitter configuration

Parameters

config – Configuration to be applied to the transmitter.

Returns

Returns *common entity* return values

aErr **getTransmitterConfig** (uint8_t *config)

Gets the transmitter configuration

Parameters

config – Configuration of the Transmitter.

Returns

Returns *common entity* return values

3.3.8 Ethernet Class

See the *Ethernet Entity* for generic information.

class **EthernetClass** : public Acroname::BrainStem::EntityClass

EthernetClass: IP configuration. MAC info. BrainD port.

Public Functions

EthernetClass (void)

Constructor.

virtual **~EthernetClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Ethernet Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Ethernet entity to be addressed.

***aErr* setEnabled** (const uint8_t enabled)

Sets the Ethernet's interface to enabled/disabled.

Parameters

enabled – 1 = enabled; 0 = disabled

Returns

Returns *common entity* return values

***aErr* getEnabled** (uint8_t *enabled)

Gets the current enable value of the Ethernet interface.

Parameters

enabled – 1 = Fully enabled network connectivity; 0 = Ethernet MAC is disabled.

Returns

Returns *common entity* return values

***aErr* getNetworkConfiguration** (uint8_t *addressStyle)

Get the method in which IP Address is assigned to this device

Parameters

addressStyle – Method used. Current methods

- NONE = 0
- STATIC = 1
- DHCP = 2

Returns

Returns *common entity* return values

***aErr* setNetworkConfiguration** (const uint8_t addressStyle)

Get the method in which IP Address is assigned to this device

Parameters

addressStyle – Method to use. See getNetworkConfiguration for addressStyle enumerations.

Returns

Returns *common entity* return values

***aErr* getStaticIPv4Address** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Get the expected IPv4 address of this device, when networkConfiguration == STATIC

Note: The functional IPv4 address of The *Module* will differ if NetworkConfiguration != STATIC.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4Address** (const uint8_t *buffer, const size_t bufferLength)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC.

Parameters

- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4.

Returns

Returns *common entity* return values

aErr **getStaticIPv4Netmask** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Get the expected IPv4 netmask of this device, when networkConfiguration == STATIC

Note: The functional IPv4 netmask of The *Module* will differ if NetworkConfiguration != STATIC.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4Netmask** (const uint8_t *buffer, const size_t bufferLength)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4.

Returns

Returns *common entity* return values

aErr **getStaticIPv4Gateway** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Get the expected IPv4 gateway of this device, when networkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4Gateway** (const uint8_t *buffer, const size_t bufferLength)

Set the desired IPv4 gateway of this device, if NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4. setStaticIPv4Gateway([192, 168, 1, 1], 4) would equate with address “192.168.1.1”

Returns

Returns *common entity* return values

aErr **getIPv4Address** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Get the effective IP address of this device.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **getIPv4Netmask** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Get the effective IP netmask of this device.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **getIPv4Gateway** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Get the effective IP gateway of this device.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4DNSAddress** (const uint8_t *buffer, const size_t bufferLength)

Set IPv4 DNS Addresses (plural), if NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Total array length in bytes. Must be a multiple of 4.

Returns

Returns *common entity* return values

aErr **getStaticIPv4DNSAddress** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Get IPv4 DNS addresses (plural), when NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Maximum length of array, in bytes.

- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **getIPv4DNSAddress** (uint8_t *buffer, const size_t bufferSize, size_t *unloadedLength)

Get effective IPv4 DNS addresses, for the current NetworkConfiguration

Parameters

- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Maximum length of array, in bytes.
- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **setHostname** (const uint8_t *buffer, const size_t bufferSize)

Set hostname that's requested when this device sends a DHCP request.

Parameters

- **buffer** – alias to an array of uint8_t[N]
- **bufferLength** – N, for N bytes.

Returns

Returns *common entity* return values

aErr **getHostname** (uint8_t *buffer, const size_t bufferSize, size_t *unloadedLength)

Get hostname that's requested when this device sends a DHCP request.

Parameters

- **buffer** – alias to an array of uint8_t[N]
- **bufferLength** – N, for N bytes.
- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **getMACAddress** (uint8_t *buffer, const size_t bufferSize, size_t *unloadedLength)

Get the MAC address of the Ethernet interface.

Parameters

- **buffer** – alias to an array of uint8_t[6]
- **bufferLength** – length of buffer that's writeable, should be > 6.
- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **setInterfacePort** (const uint8_t service, const uint16_t port)

Set the port of a TCPIP service on the device.

Parameters

- **service** – The index of the service to set the port for.
- **port** – The port to be used for the TCPIP server.

Returns

Returns *common entity* return values

aErr **getInterfacePort** (const uint8_t service, uint16_t *port)

Get the port of a TCPIP service on the device.

Parameters

- **service** – The index of the service to get the port for.
- **port** – The port of the TCPIP server.

Returns

Returns *common entity* return values

3.3.9 HDBaseT Class

See the *HDBaseT Entity* for generic information.

class **HDBaseTClass** : public Acroname::BrainStem::EntityClass

HDBaseTClass: This entity is only available on certain modules, and provides information on HDBaseT extenders.

Public Functions

HDBaseTClass (void)

Constructor.

virtual **~HDBaseTClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the HDBaseT Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the HDBaseT entity to be addressed.

aErr **getSerialNumber** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the serial number of the HDBaseT device (6 bytes)

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getFirmwareVersion** (uint32_t *firmwareVersion)

Gets the firmware version of the HDBaseT device

Parameters

firmwareVersion – A bit packet representation of the firmware version Major: Bits 24-31; Minor: Bits 16-23; Patch: Bits 8-15; Build: Bits 0-7

Returns

Returns *common entity* return values

aErr **getState** (uint32_t *state)

Gets the current state of the HDBaseT link

Parameters

state – Bit packeted representation of the state.

Returns

Returns *common entity* return values

aErr **getCableLength** (uint32_t *cableLength)

Gets the perceived cable length

Parameters

cableLength – Cable length in micro-meters

Returns

Returns *common entity* return values

aErr **getMSEA** (int32_t *mseA)

Gets the Mean Squared Error (MSE) for channel A

Parameters

mseA – The current MSE for channel A in micro-dB

Returns

Returns *common entity* return values

aErr **getMSEB** (int32_t *mseB)

Gets the Mean Squared Error (MSE) for channel B

Parameters

mseB – The current MSE for channel B in micro-dB

Returns

Returns *common entity* return values

aErr **getRetransmissionRate** (uint32_t *retransmissionRate)

Gets the number of successful messages between retransmission

Parameters

retransmissionRate – Instantaneous number of successful messages between retransmission. To be interpreted as: 1 / retransmissionRate for rate interpretation. If the value is 0, there have been no retransmissions, otherwise higher is better..

Returns

Returns *common entity* return values

aErr **getLinkUtilization** (uint32_t *linkUtilization)

Gets the current link utilization

Parameters

linkUtilization – Utilization in milli-percent

Returns

Returns *common entity* return values

aErr **getEncodingState** (uint8_t *encodingState)

Gets the current encoding state.

Parameters

encodingState – Signal modulation encoding type.

Returns

Returns *common entity* return values

aErr **getUSB2DeviceTree** (uint8_t *buffer, const size_t bufferSize, size_t *unloadedLength)

Gets the USB2 tree at the HDBaseT device.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getUSB3DeviceTree** (uint8_t *buffer, const size_t bufferSize, size_t *unloadedLength)

Gets the USB3 tree at the HDBaseT device.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getLinkRole** (uint8_t *role)

Gets the current link role In the case of “Auto” the getState API will provide the current role.

Parameters

role – *Link* role

Returns

Returns *common entity* return values

aErr **setLinkRole** (const uint8_t role)

Sets the active link role

Parameters

role – The role to be set.

Returns

Returns *common entity* return values

3.3.10 I2C Class

See the *I2C Entity* for generic information.

class **I2CClass** : public Acroname::BrainStem::EntityClass

I2CClass: Interface the I2C buses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entities bus.

Public Functions

I2CClass (void)

Constructor.

virtual **~I2CClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the I2C Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the I2C entity to be addressed.

aErr **read** (const uint8_t address, const uint8_t readLength, uint8_t *buffer)

Read from a device on this I2C bus.

Parameters

- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **readLength** – The length of the data to read in bytes.
- **buffer** – The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAINSTEM_MAXPACKETBYTES - 5

Returns

Returns *common entity* return values

aErr **write** (const uint8_t address, const uint8_t bufferLength, const uint8_t *buffer)

Write to a device on this I2C bus.

Parameters

- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to write.
- **bufferLength** – The length of the data to write in bytes.
- **buffer** – The data to send to the device This array should be no larger than aBRAINSTEM_MAXPACKETBYTES - 5

Returns

Returns *common entity* return values

aErr **setPullup** (const uint8_t enable)

Set bus pull-up state. This call only works with stems that have software controlled pull-ups. Check the datasheet for more information. This parameter is saved when system.save is called.

Parameters

enable – true enables pull-ups false disables them.

Returns

Returns *common entity* return values

aErr **setSpeed** (const uint8_t speed)

Set I2C bus speed.

This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values: 1 - 100Khz 2 - 400Khz 3 - 1MHz

Parameters

speed – The speed setting value.

Returns

Returns *common entity* return values

aErr **getSpeed** (uint8_t *speed)

Get I2C bus speed.

This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values: 1 - 100Khz 2 - 400Khz 3 - 1MHz

Parameters

speed – The speed setting value.

Returns

Returns *common entity* return values

3.3.11 Mux Class

See the *Mux Entity* for generic information.

class **MuxClass** : public Acroname::BrainStem::EntityClass

MuxClass: A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to on or more outputs. One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs; some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

Public Functions

MuxClass (void)

Constructor.

virtual ~**MuxClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Mux Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Mux entity to be addressed.

aErr **getEnable** (uint8_t *enabled)

Get the mux enable/disable status

Parameters

enabled – true: mux is enabled, false: the mux is disabled.

Returns

Returns *common entity* return values

aErr **setEnabled** (const uint8_t enable)

Enable the mux.

Parameters

enable – true: enables the mux for the selected channel.

Returns

Returns *common entity* return values

aErr **getChannel** (uint8_t *channel)

Get the current selected mux channel.

Parameters

channel – Indicates which channel is selected.

Returns

Returns *common entity* return values

aErr **setChannel** (const uint8_t channel)

Set the current mux channel.

Parameters

channel – mux channel to select.

Returns

Returns *common entity* return values

aErr **getChannelVoltage** (const uint8_t channel, int32_t *microvolts)

Get the voltage of the indicated mux channel.

Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

- **channel** – The channel in which voltage was requested.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns *common entity* return values

aErr **getConfiguration** (int32_t *config)

Get the configuration of the mux.

Parameters

config – integer representing the mux configuration either default, or split-mode.

Returns

Returns *common entity* return values

aErr **setConfiguration** (const int32_t config)

Set the configuration of the mux.

Parameters

config – integer representing the mux configuration either muxConfig_default, or mux-Config_splitMode.

Returns

Returns *common entity* return values

aErr **getSplitMode** (int32_t *splitMode)

Get the current split mode mux configuration.

Parameters

splitMode – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Returns

Returns *common entity* return values

aErr **setSplitMode** (const int32_t splitMode)

Sets the mux's split mode configuration.

Parameters

splitMode – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Returns

Returns *common entity* return values

3.3.12 PoE Class

See the *PoE Entity* for generic information.

class **PoEClass** : public Acroname::BrainStem::EntityClass

PoEClass: This entity is only available on certain modules, and provides a Power over Ethernet control ability.

Public Functions

PoEClass (void)

Constructor.

virtual **~PoEClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the PoE Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the PoE entity to be addressed.

aErr **getPairEnabled** (const uint8_t pair, uint8_t *enable)

Gets the current enable value of the indicated POE pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

- **enable** – 1 = Enabled; 0 = Disabled;

Returns

Returns *common entity* return values

aErr **setPairEnabled** (const uint8_t pair, const uint8_t enable)

Enables or disables the indicated POE pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **enable** – 1 = Enable port; 0 = Disable port.

Returns

Returns *common entity* return values

aErr **getPowerMode** (uint8_t *value)

Gets the power mode of the device

Parameters

value – The power mode (PD, PSE, Auto, Off).

Returns

Returns *common entity* return values

aErr **setPowerMode** (const uint8_t value)

Sets the power mode of the device

Parameters

value – The power mode (PD, PSE, Auto, Off).

Returns

Returns *common entity* return values

aErr **getPowerState** (uint8_t *value)

Gets the power state of the device

Parameters

value – The power state (PD, PSE, Off).

Returns

Returns *common entity* return values

aErr **getPairSourcingClass** (const uint8_t pair, uint8_t *value)

Gets the sourcing class for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The POE class being offered by the device (PSE).

Returns

Returns *common entity* return values

aErr **setPairSourcingClass** (const uint8_t pair, const uint8_t value)

Sets the sourcing class for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The POE class being offered by the device (PSE).

Returns

Returns *common entity* return values

aErr **getPairRequestedClass** (const uint8_t pair, uint8_t *value)

Gets the requested class for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The requested POE class by the device (PD).

Returns

Returns *common entity* return values

aErr **getPairDiscoveredClass** (const uint8_t pair, uint8_t *value)

Gets the discovered class for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The negotiated POE class by the device (PSE/PD).

Returns

Returns *common entity* return values

aErr **getPairDetectionStatus** (const uint8_t pair, uint8_t *value)

Gets detected status of the POE connection for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The current detected status of the pairs.

Returns

Returns *common entity* return values

aErr **getPairVoltage** (const uint8_t pair, int32_t *microvolts)

Gets the Voltage for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **microvolts** – The voltage in microvolts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getPairCurrent** (const uint8_t pair, int32_t *microamps)

Gets the Current for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **microamps** – The current in microamps (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getPairResistance** (const uint8_t pair, int32_t *milliohms)

Gets the Resistance for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **milliohms** – The resistance in milliohms (1 == 1e-3Z).

Returns

Returns *common entity* return values

aErr **getPairCapacitance** (const uint8_t pair, int32_t *nanofarads)

Gets the Capacitance for a given pair

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **nanofarads** – The capacitance in nanofarads (1 == 1e-9F).

Returns

Returns *common entity* return values

aErr **getPairPower** (const uint8_t pair, int32_t *power)

Get the instantaneous power consumption for a given pair The equivalent of Voltage x Current

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **power** – Variable to be filled with the pairs power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getTotalPower** (int32_t *power)

Gets the total instantaneous power consumption The equivalent of Pair1(Voltage x Current) + Pair2(Voltage x Current)

Parameters

- **power** – Variable to be filled with the total POE power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPairAccumulatedPower** (const uint8_t pair, int32_t *power)

Gets the accumulated power for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **power** – Variable to be filled with the total accumulated POE power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **setPairAccumulatedPower** (const uint8_t pair, const int32_t power)

Sets the accumulated power for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **power** – The power accumulator value to be set in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getTotalAccumulatedPower** (int32_t *power)

Gets the total Accumulated Power

Parameters

- **power** – Variable to be filled with the total accumulated POE power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **setTotalAccumulatedPower** (const int32_t power)

Sets the total accumulated power

Parameters

power – The power accumulator value to be set in milli-watts (mW).

Returns

Returns *common entity* return values

3.3.13 Pointer Class

See the *Pointer Entity* for generic information.

class **PointerClass** : public Acroname::BrainStem::EntityClass

PointerClass: Allows access to the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the BrainStem scratchpad from the host, and provides a mechanism for allowing the host application and BrainStem relexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via `setOffset`. A read of a character short or int can be made from that cursor position.

In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

Public Functions

PointerClass (void)

Constructor.

virtual **~PointerClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Pointer Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Pointer entity to be addressed.

aErr **getOffset** (uint16_t *offset)

Get the offset of the pointer

Parameters

offset – The value of the offset.

Returns

Returns *common entity* return values

aErr **setOffset** (const uint16_t offset)

Set the offset of the pointer

Parameters

offset – The value of the offset.

Returns

Returns *common entity* return values

aErr **getMode** (uint8_t *mode)

Get the mode of the pointer

Parameters

mode – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Returns

Returns *common entity* return values

aErr **setMode** (const uint8_t mode)

Set the mode of the pointer

Parameters

mode – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Returns

Returns *common entity* return values

aErr **getTransferStore** (uint8_t *handle)

Get the handle to the store.

Parameters

handle – The handle of the store.

Returns

Returns *common entity* return values

aErr **setTransferStore** (const uint8_t handle)

Set the handle to the store.

Parameters

handle – The handle of the store.

Returns

Returns *common entity* return values

aErr **initiateTransferToStore** (const uint8_t transferLength)

Transfer data to the store.

Parameters

transferLength – The length of the data transfer.

Returns

Returns *common entity* return values

aErr **initiateTransferFromStore** (const uint8_t transferLength)

Transfer data from the store.

Parameters

transferLength – The length of the data transfer.

Returns

Returns *common entity* return values

aErr **getChar** (uint8_t *value)

Get a char (1 byte) value from the pointer at this object's index, where elements are 1 byte long.

Parameters

value – The value of a single character (1 byte) stored in the pointer.

Returns

Returns *common entity* return values

aErr **setChar** (const uint8_t value)

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

Parameters

value – The single char (1 byte) value to be stored in the pointer.

Returns

Returns *common entity* return values

aErr **getShort** (uint16_t *value)

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long

Parameters

value – The value of a single short (2 byte) stored in the pointer.

Returns

Returns *common entity* return values

aErr **setShort** (const uint16_t value)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

Parameters

value – The single short (2 byte) value to be set in the pointer.

Returns

Returns *common entity* return values

aErr **getInt** (uint32_t *value)

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

Parameters

value – The value of a single int (4 byte) stored in the pointer.

Returns

Returns *common entity* return values

aErr **setInt** (const uint32_t value)

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

Parameters

value – The single int (4 byte) value to be stored in the pointer.

Returns

Returns *common entity* return values

3.3.14 Port Class

See the *Port Entity* for generic information.

class **PortClass** : public Acroname::BrainStem::EntityClass

PortClass: The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

Public Functions

PortClass (void)

Constructor.

virtual ~**PortClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Port Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Port entity to be addressed.

aErr **getVbusVoltage** (int32_t *microvolts)

Gets the Vbus Voltage

Parameters

microvolts – The voltage in microvolts (1 == 1e-6V) currently present on Vbus.

Returns

Returns *common entity* return values

aErr **getVbusCurrent** (int32_t *microamps)

Gets the Vbus Current

Parameters

microamps – The current in microamps (1 == 1e-6A) currently present on Vbus.

Returns

Returns *common entity* return values

aErr **getVconnVoltage** (int32_t *microvolts)

Gets the Vconn Voltage

Parameters

microvolts – The voltage in microvolts (1 == 1e-6V) currently present on Vconn.

Returns

Returns *common entity* return values

aErr **getVconnCurrent** (int32_t *microamps)

Gets the Vconn Current

Parameters

microamps – The current in microamps (1 == 1e-6A) currently present on Vconn.

Returns

Returns *common entity* return values

aErr **getPowerMode** (uint8_t *powerMode)

Gets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

powerMode – The current power mode.

Returns

Returns *common entity* return values

aErr **setPowerMode** (const uint8_t powerMode)

Sets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

powerMode – The power mode to be set.

Returns

Returns *common entity* return values

aErr **getEnabled** (uint8_t *enable)

Gets the current enable value of the port.

Parameters

enable – 1 = Fully enabled port; 0 = One or more disabled components.

Returns

Returns *common entity* return values

aErr **setEnabled** (const uint8_t enable)

Enables or disables the entire port.

Parameters

enable – 1 = Fully enable port; 0 = Fully disable port.

Returns

Returns *common entity* return values

aErr **getDataEnabled** (uint8_t *enable)

Gets the current enable value of the data lines. Sub-component (Data) of getEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataEnabled** (const uint8_t enable)

Enables or disables the data lines. Sub-component (Data) of setEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataHSEnabled** (uint8_t *enable)

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled.

Parameters

`enable` – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataHSEnabled** (const uint8_t enable)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.

Parameters

`enable` – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataHS1Enabled** (uint8_t *enable)

Gets the current enable value of the High Speed A side (HSA) data lines. Sub-component of getDataHSEnabled.

Parameters

`enable` – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataHS1Enabled** (const uint8_t enable)

Enables or disables the High Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

Parameters

`enable` – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataHS2Enabled** (uint8_t *enable)

Gets the current enable value of the High Speed B side (HSB) data lines. Sub-component of getDataHSEnabled.

Parameters

`enable` – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataHS2Enabled** (const uint8_t enable)

Enables or disables the High Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

Parameters

`enable` – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSSEnabled** (uint8_t *enable)

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled.

Parameters

`enable` – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSSEnabled** (const uint8_t enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Parameters

enable - 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSS1Enabled** (uint8_t *enable)

Gets the current enable value of the Super Speed A side (SSA) data lines. Sub-component of getDataSSEnabled.

Parameters

enable - 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSS1Enabled** (const uint8_t enable)

Enables or disables the Super Speed A side (SSA) data lines. Sub-component of setDataEnabled.

Parameters

enable - 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSS2Enabled** (uint8_t *enable)

Gets the current enable value of the Super Speed B side (SSB) data lines. Sub-component of getDataSSEnabled.

Parameters

enable - 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSS2Enabled** (const uint8_t enable)

Enables or disables the Super Speed B side (SSB) data lines. Sub-component of setDataSSEnabled.

Parameters

enable - 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getPowerEnabled** (uint8_t *enable)

Gets the current enable value of the power lines. Sub-component (Power) of getEnabled.

Parameters

enable - 1 = Power enabled; 0 = Power disabled.

Returns

Returns *common entity* return values

aErr **setPowerEnabled** (const uint8_t enable)

Enables or Disables the power lines. Sub-component (Power) of setEnable.

Parameters

enable - 1 = Enable power; 0 = Disable disable.

Returns

Returns *common entity* return values

aErr **getDataRole** (uint8_t *dataRole)

Gets the Port Data Role.

Parameters

dataRole – The data role to be set. See datasheet for details.

Returns

Returns *common entity* return values

aErr **getVconnEnabled** (uint8_t *enable)

Gets the current enable value of the Vconn lines. Sub-component (Vconn) of getEnabled.

Parameters

enable – 1 = Vconn enabled; 0 = Vconn disabled.

Returns

Returns *common entity* return values

aErr **setVconnEnabled** (const uint8_t enable)

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

Parameters

enable – 1 = Enable Vconn lines; 0 = Disable Vconn lines.

Returns

Returns *common entity* return values

aErr **getVconn1Enabled** (uint8_t *enable)

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled.

Parameters

enable – 1 = Vconn1 enabled; 0 = Vconn1 disabled.

Returns

Returns *common entity* return values

aErr **setVconn1Enabled** (const uint8_t enable)

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

Parameters

enable – 1 = Enable Vconn1 lines; 0 = Disable Vconn1 lines.

Returns

Returns *common entity* return values

aErr **getVconn2Enabled** (uint8_t *enable)

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled.

Parameters

enable – 1 = Vconn2 enabled; 0 = Vconn2 disabled.

Returns

Returns *common entity* return values

aErr **setVconn2Enabled** (const uint8_t enable)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

Parameters

enable – 1 = Enable Vconn2 lines; 0 = Disable Vconn2 lines.

Returns

Returns *common entity* return values

aErr **getCCEnabled** (uint8_t *enable)

Gets the current enable value of the CC lines. Sub-component (CC) of getEnabled.

Parameters

enable – 1 = CC enabled; 0 = CC disabled.

Returns

Returns *common entity* return values

aErr **setCCEnabled** (const uint8_t enable)

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

Parameters

enable – 1 = Enable CC lines; 0 = Disable CC lines.

Returns

Returns *common entity* return values

aErr **getCC1Enabled** (uint8_t *enable)

Gets the current enable value of the CC1 lines. Sub-component of getCCEnabled.

Parameters

enable – 1 = CC1 enabled; 0 = CC1 disabled.

Returns

Returns *common entity* return values

aErr **setCC1Enabled** (const uint8_t enable)

Enables or disables the CC1 lines. Sub-component of setCCEnabled.

Parameters

enable – 1 = Enable CC1 lines; 0 = Disable CC1 lines.

Returns

Returns *common entity* return values

aErr **getCC2Enabled** (uint8_t *enable)

Gets the current enable value of the CC2 lines. Sub-component of getCCEnabled.

Parameters

enable – 1 = CC2 enabled; 0 = CC2 disabled.

Returns

Returns *common entity* return values

aErr **setCC2Enabled** (const uint8_t enable)

Enables or disables the CC2 lines. Sub-component of setCCEnabled.

Parameters

enable – 1 = Enable CC2 lines; 0 = Disable CC2 lines.

Returns

Returns *common entity* return values

aErr **getVoltageSetpoint** (uint32_t *value)

Gets the current voltage setpoint value for the port.

Parameters

value – the voltage setpoint of the port in uV.

Returns

Returns *common entity* return values

aErr **setVoltageSetpoint** (const uint32_t value)

Sets the current voltage setpoint value for the port.

Parameters

value – the voltage setpoint of the port in uV.

Returns

Returns *common entity* return values

aErr **getState** (uint32_t *state)

A bit mapped representation of the current state of the port. Reflects what the port IS which may differ from what was requested.

Parameters

state – Variable to be filled with the current state.

Returns

Returns *common entity* return values

aErr **getDataSpeed** (uint8_t *speed)

Gets the speed of the enumerated device.

Parameters

speed – Bit mapped value representing the devices speed. See “Devices” reference for details.

Returns

Returns *common entity* return values

aErr **getMode** (uint32_t *mode)

Gets current mode of the port

Parameters

mode – Bit mapped value representing the ports mode. See “Devices” reference for details.

Returns

Returns *common entity* return values

aErr **setMode** (const uint32_t mode)

Sets the mode of the port

Parameters

mode – Port mode to be set. See “Devices” documentation for details.

Returns

Returns *common entity* return values

aErr **getErrors** (uint32_t *errors)

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

Parameters

errors – Bit mapped field representing the current errors of the ports

Returns

Returns *common entity* return values

aErr **getCurrentLimit** (uint32_t *limit)

Gets the current limit of the port.

Parameters

limit – Variable to be filled with the limit in microAmps (uA).

Returns

Returns *common entity* return values

aErr **setCurrentLimit** (const uint32_t limit)

Sets the current limit of the port.

Parameters

limit – Current limit to be applied in microAmps (uA).

Returns

Returns *common entity* return values

aErr **getCurrentLimitMode** (uint8_t *mode)

Gets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters

mode – Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setCurrentLimitMode** (const uint8_t mode)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters

mode – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getAvailablePower** (uint32_t *power)

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Parameters

power – Variable to be filled with the available power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getAllocatedPower** (int32_t *power)

Gets the currently allocated power This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Parameters

power – Variable to be filled with the allocated power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPowerLimit** (uint32_t *limit)

Gets the user defined power limit for the port.

Parameters

limit – Variable to be filled with the power limit in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **setPowerLimit** (const uint32_t limit)

Sets a user defined power limit for the port.

Parameters

limit – Power limit to be applied in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPowerLimitMode** (uint8_t *mode)

Gets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

mode – Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setPowerLimitMode** (const uint8_t mode)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

mode – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getName** (uint8_t *buffer, const size_t bufferSize, size_t *unloadedLength)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setName** (const uint8_t *buffer, const size_t bufferSize)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getCCCurrentLimit** (uint8_t *value)

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

Parameters

value – Variable to be filled with an enumerated representation of the CC Current limit.

- 0 = None
- 1 = Default (500/900mA)
- 2 = 1.5A
- 3 = 3.0A

Returns

Returns *common entity* return values

aErr **setCCCurrentLimit** (const uint8_t value)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

Parameters

value – Variable to be filled with an enumerated representation of the CC Current limit.

- 0 = None
- 1 = Default (500/900mA)
- 2 = 1.5A
- 3 = 3.0A

Returns

Returns *common entity* return values

aErr **getDataHSRoutingBehavior** (uint8_t *mode)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataHSRoutingBehavior** (const uint8_t mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getDataSSRoutingBehavior** (uint8_t *mode)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataSSRoutingBehavior** (const uint8_t mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getVbusAccumulatedPower** (int32_t *milliwatthours)

Gets the Vbus Accumulated Power

Parameters

milliwatthours – The accumulated power on Vbus in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setVbusAccumulatedPower** (const int32_t milliwatthours)

Sets the Vbus Accumulated Power

Parameters

milliwatthours – The accumulated power on Vbus in milliwatt-hours to be set.

Returns

Returns *common entity* return values

aErr **resetVbusAccumulatedPower** (void)

Resets the Vbus Accumulated Power to zero.

Deprecated:

Replace with setVbusAccumulatedPower(0)

Returns

Returns *common entity* return values

aErr **getVconnAccumulatedPower** (int32_t *milliwatthours)

Gets the Vconn Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setVconnAccumulatedPower** (const int32_t milliwatthours)

Sets the Vconn Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn to be set.

Returns

Returns *common entity* return values

aErr **resetVconnAccumulatedPower** (void)

Resets the Vconn Accumulated Power to zero.

Deprecated:

Replace with setVconnAccumulatedPower(0)

Returns

Returns *common entity* return values

aErr **setHSBoost** (const uint8_t boost)

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Parameters

boost – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getHSBoost** (uint8_t *boost)

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Parameters

boost – An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getCC1State** (uint16_t *value)

Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Parameters

value – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:

- None = 0 = portCC1State_None
- Invalid = 1 = portCC1State_Invalid
- Rp (default) = 2 = portCC1State_RpDefault
- Rp (1.5A) = 3 = portCC1State_Rp1p5
- Rp (3A) = 4 = portCC1State_Rp3p0
- Rd = 5 = portCC1State_Rd
- Ra = 6 = portCC1State_Ra
- Managed by controller = 7 = portCC1State_Managed
- Unknown = 8 = portCC1State_Unknown

Returns

Returns *common entity* return values

aErr **getCC2State** (uint16_t *value)

Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Parameters

value – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:

- None = 0 = portCC2State_None
- Invalid = 1 = portCC2State_Invalid
- Rp (default) = 2 = portCC2State_RpDefault
- Rp (1.5A) = 3 = portCC2State_Rp1p5
- Rp (3A) = 4 = portCC2State_Rp3p0
- Rd = 5 = portCC2State_Rd
- Ra = 6 = portCC2State_Ra
- Managed by controller = 7 = portCC2State_Managed
- Unknown = 8 = portCC2State_Unknown

Returns

Returns *common entity* return values

aErr **getSBU1Voltage** (int32_t *microvolts)

Get the voltage of SBU1 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getSBU2Voltage** (int32_t *microvolts)

Get the voltage of SBU2 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC1Voltage** (int32_t *microvolts)

Get the voltage of CC1 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC2Voltage** (int32_t *microvolts)

Get the voltage of CC2 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC1Current** (int32_t *microamps)

Get the current through the CC1 for a port.

Parameters

microamps – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC2Current** (int32_t *microamps)

Get the current through the CC2 for a port.

Parameters

microamps – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC1AccumulatedPower** (int32_t *milliwatthours)

Gets the CC1 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setCC1AccumulatedPower** (const int32_t milliwatthours)

Sets the CC1 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn to be set.

Returns

Returns *common entity* return values

aErr **getCC2AccumulatedPower** (int32_t *milliwatthours)

Gets the CC2 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setCC2AccumulatedPower** (const int32_t milliwatthours)

Sets the CC2 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn to be set.

Returns

Returns *common entity* return values

3.3.15 PowerDelivery Class

See the [PowerDelivery Entity](#) for generic information.

class **PowerDeliveryClass** : public Acroname::BrainStem::EntityClass

[PowerDeliveryClass](#): Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface. This Entity will allow you to directly access the vast landscape of PD.

Public Functions

PowerDeliveryClass (void)

Constructor.

virtual ~**PowerDeliveryClass** (void)

Destructor.

void **init** ([Module](#) *pModule, const uint8_t index)

Initialize the PowerDelivery Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the PowerDelivery entity to be addressed.

[aErr](#) **getConnectionState** (uint8_t *state)

Gets the current state of the connection in the form of an enumeration.

Parameters

state – Pointer to be filled with the current connection state.

Returns

Returns [common entity](#) return values

[aErr](#) **getNumberOfPowerDataObjects** (const uint8_t partner, const uint8_t powerRole, uint8_t *numRules)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

Parameters

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **numRules** – Variable to be filled with the number of PDOs.

Returns

Returns [common entity](#) return values

aErr **getPowerDataObject** (const uint8_t partner, const uint8_t powerRole, const uint8_t ruleIndex, uint32_t *pdo)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

Parameters

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Variable to be filled with the requested power rule.

Returns

Returns *common entity* return values

aErr **setPowerDataObject** (const uint8_t powerRole, const uint8_t ruleIndex, const uint32_t pdo)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Power Data Object to be set.

Returns

Returns *common entity* return values

aErr **resetPowerDataObjectToDefault** (const uint8_t powerRole, const uint8_t ruleIndex)

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Returns

Returns *common entity* return values

aErr **getPowerDataObjectList** (uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets all Power Data Objects (PDOs). Equivalent to calling *PowerDeliveryClass::getPowerDataObject()* on all partners, power roles, and index's.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled The order of which is:

- Rules 1-7 Local Source
- Rules 1-7 Local Sink
- Rules 1-7 Partner Source
- Rules 1-7 Partner Sink.
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled. On success this value should be 28 (7 rules * 2 partners * 2 power roles)

Returns

Returns *common entity* return values

aErr **getPowerDataObjectEnabled** (const uint8_t powerRole, const uint8_t ruleIndex, uint8_t *enabled)

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – Variable to be filled with enabled state.

Returns

Returns *common entity* return values

aErr **setPowerDataObjectEnabled** (const uint8_t powerRole, const uint8_t ruleIndex, const uint8_t enabled)

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – The state to be set.

Returns

Returns *common entity* return values

aErr **getPowerDataObjectEnabledList** (const uint8_t powerRole, uint8_t *enabledList)

Gets all Power Data Object enables for a given power role. Equivalent of calling *PowerDeliveryClass::getPowerDataObjectEnabled()* for all indexes.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **enabledList** – Variable to be filled with a mapped representation of the enabled PDOs for a given power role. Values align with a given rule index (bits 1-7, bit 0 is invalid)

Returns

Returns *common entity* return values

aErr **getRequestDataObject** (const uint8_t partner, uint32_t *rdo)

Gets the current Request Data Object (RDO) for a given partner. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

Parameters

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **rdo** – Variable to be filled with the current RDO. Zero indicates the RDO is not active.

Returns

Returns *common entity* return values

aErr **setRequestDataObject** (const uint32_t rdo)

Sets the current Request Data Object (RDO) for a given partner. Only the local partner can be changed. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

Parameters

rdo – Request Data Object to be set.

Returns

Returns *common entity* return values

aErr **getLinkState** (uint32_t *state)

Gets the current state of the connection in the form of a bitmask.

Parameters

state – Pointer to be filled with the current connection state bits.

Returns

Returns *common entity* return values

aErr **getAttachTimeElapsed** (uint32_t *secondsAndMicroseconds, const size_t bufferLength, size_t *unloadedLength)

Gets the length of time that the port has been in the attached state. Returned as a list of two unsigned integers, first seconds, then microseconds.

Parameters

- **secondsAndMicroseconds** – Pointer to list of unsigned integers to fill with attach time elapsed
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getPowerRoleCapabilities** (uint8_t *powerRole)

Gets the power roles that may be advertised by the local partner. (CC Strapping).

Parameters

powerRole – Variable to be filled with the power role

- None = 0 = pdPowerRoleCapabilities_None
- Source = 1 = pdPowerRoleCapabilities_Source
- Sink = 2 = pdPowerRoleCapabilities_Sink
- Source/Sink = 3 = pdPowerRoleCapabilities_DualRole (Dual Role Port)

Returns

Returns *common entity* return values

aErr **getPowerRole** (uint8_t *powerRole)

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

Parameters

powerRole – Variable to be filled with the power role

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Returns

Returns *common entity* return values

aErr **setPowerRole** (const uint8_t powerRole)

Set the current power role to be advertised by the Local partner. (CC Strapping).

Parameters

powerRole – Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Returns

Returns *common entity* return values

aErr **getPowerRolePreferred** (uint8_t *powerRole)

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

Parameters

powerRole – Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Returns

Returns *common entity* return values

aErr **setPowerRolePreferred** (const uint8_t powerRole)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

Parameters

powerRole – Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Returns

Returns *common entity* return values

aErr **getDataRoleCapabilities** (uint8_t *dataRole)

Gets the data roles that may be advertised by the local partner.

Parameters

dataRole – Variable to be filled with the data role

- None = 0 = pdDataRoleCapabilities_None
- DFP = 1 = pdDataRoleCapabilities_DFP
- UFP = 2 = pdDataRoleCapabilities_UFP
- DFP/UFP = 3 = pdDataRoleCapabilities_DualRole (Dual Role Port)

Returns

Returns *common entity* return values

aErr **getCableVoltageMax** (uint8_t *maxVoltage)

Gets the maximum voltage capability reported by the e-mark of the attached cable.

Parameters

maxVoltage – Variable to be filled with an enumerated representation of voltage.

- Unknown/Unattached (0)
- 20 Volts DC (1)
- 30 Volts DC (2)
- 40 Volts DC (3)
- 50 Volts DC (4)

Returns

Returns *common entity* return values

aErr **getCableCurrentMax** (uint8_t *maxCurrent)

Gets the maximum current capability report by the e-mark of the attached cable.

Parameters

maxCurrent – Variable to be filled with an enumerated representation of current.

- Unknown/Unattached (0)
- 3 Amps (1)
- 5 Amps (2)

Returns

Returns *common entity* return values

aErr **getCableSpeedMax** (uint8_t *maxSpeed)

Gets the maximum data rate capability reported by the e-mark of the attached cable.

Parameters

maxSpeed – Variable to be filled with an enumerated representation of data speed.

- Unknown/Unattached (0)
- USB 2.0 (1)
- USB 3.2 gen 1 (2)
- USB 3.2 / USB 4 gen 2 (3)
- USB 4 gen 3 (4)

Returns

Returns *common entity* return values

aErr **getCableType** (uint8_t *type)

Gets the cable type reported by the e-mark of the attached cable.

Parameters

type – Variable to be filled with an enumerated representation of the cable type.

- Invalid, no e-mark and not Vconn powered (0)
- Passive cable with e-mark (1)
- Active cable (2)

Returns

Returns *common entity* return values

aErr **getCableOrientation** (uint8_t *orientation)

Gets the current orientation being used for PD communication

Parameters

orientation – Variable filled with an enumeration of the orientation.

- Unconnected (0)
- CC1 (1)
- CC2 (2)

Returns

Returns *common entity* return values

aErr **request** (const uint8_t request)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

Parameters

request – Request to be issued to the remote partner

- pdRequestHardReset (0)
- pdRequestSoftReset (1)
- pdRequestDataReset (2)
- pdRequestPowerRoleSwap (3)

- `pdRequestPowerFastRoleSwap` (4)
- `pdRequestDataRoleSwap` (5)
- `pdRequestVconnSwap` (6)
- `pdRequestSinkGoToMinimum` (7)
- `pdRequestRemoteSourcePowerDataObjects` (8)
- `pdRequestRemoteSinkPowerDataObjects` (9)
- `pdRequestRemoteSourceExtendedCapabilities` (10)
- `pdRequestRemoteSinkExtendedCapabilities` (11)
- `pdRequestStatus` (12)
- `pdRequestPPSStatus` (13)
- `pdRequestBatteryCapabilities` (14)
- `pdRequestBatteryStatus` (15)
- `pdRequestManufacturerInfoSop` (16)
- `pdRequestManufacturerInfoSopp` (17)
- `pdRequestManufacturerInfoSoppp` (18)
- `pdRequestDiscoverIdentitySop` (19)
- `pdRequestDiscoverIdentitySopp` (20)
- `pdRequestDiscoverIdentitySoppp` (21)
- `pdRequestRevision` (22)
- `pdRequestSourceInfo` (23)
- `pdRequestCountryCode` (24)
- `pdRequestCountryInfo` (25)
- `pdRequestRemoteSourceEPRCapabilities` (26)
- `pdRequestRemoteSinkEPRCapabilities` (27)

Returns

Returns *common entity* return values

aErr **requestStatus** (uint32_t *status)

Gets the status of the last request command sent.

Parameters

status – Variable to be filled with the status

Returns

Returns *common entity* return values

aErr **getOverride** (uint32_t *overrides)

Gets the current enabled overrides

Parameters

overrides – Bit mapped representation of the current override configuration.

Returns

Returns *common entity* return values

aErr setOverride (const uint32_t overrides)

Sets the current enabled overrides

Parameters

overrides – Overrides to be set in a bit mapped representation.

Returns

Returns *common entity* return values

aErr getProtocolRevisionMax (uint8_t *maxProtocolRevision)

Gets the configured maximum USB Power Delivery protocol revision used as a negotiation ceiling (pdProtocolRevision2 = 2, pdProtocolRevision3 = 3). This is unrelated to hardware or product revision.

Parameters

maxProtocolRevision – Filled with pdProtocolRevision2 (2) or pdProtocolRevision3 (3).

Returns

Returns *common entity* return values

aErr setProtocolRevisionMax (const uint8_t maxProtocolRevision)

Sets the maximum USB Power Delivery protocol revision for negotiation. Only pdProtocolRevision2 (2) and pdProtocolRevision3 (3) are valid API values.

Parameters

maxProtocolRevision – pdProtocolRevision2 (2) or pdProtocolRevision3 (3).

Returns

Returns *common entity* return values

aErr getFlagMode (const uint8_t flag, uint8_t *mode)

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Parameters

- **flag** – Flag/Advertisement to be modified
- **mode** – Variable to be filled with the current mode.
 - Disabled (0)
 - Enabled (1)
 - Auto (2) default

Returns

Returns *common entity* return values

aErr setFlagMode (const uint8_t flag, const uint8_t mode)

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Parameters

- **flag** – Flag/Advertisement to be modified
- **mode** – Value to be applied.
 - Disabled (0)

- Enabled (1)
- Auto (2) default

Returns

Returns *common entity* return values

aErr **getPeakCurrentConfiguration** (uint8_t *configuration)

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Parameters

configuration – An enumerated value referring to the current configuration.

- Allowable values are 0 - 4

Returns

Returns *common entity* return values

aErr **setPeakCurrentConfiguration** (const uint8_t configuration)

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Parameters

configuration – An enumerated value referring to the configuration to be set

- Allowable values are 0 - 4

Returns

Returns *common entity* return values

aErr **getFastRoleSwapCurrent** (uint8_t *swapCurrent)

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

Parameters

swapCurrent – An enumerated value referring to current swap value.

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

Returns

Returns *common entity* return values

aErr **setFastRoleSwapCurrent** (const uint8_t swapCurrent)

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

Parameters

swapCurrent – An enumerated value referring to value to be set.

- 0A (0)
- 900mA (1)
- 1.5A (2)

- 3A (3)

Returns

Returns *common entity* return values

Public Static Functions

static *aErr* **packDataObjectAttributes** (uint8_t *attributes, const uint8_t partner, const uint8_t powerRole, const uint8_t ruleIndex)

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

Parameters

- **attributes** – Variable to be filled with packed values.
- **partner** – Indicates which side of the PD connection.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

Returns

Returns *common entity* return values

static *aErr* **unpackDataObjectAttributes** (const uint8_t attributes, uint8_t *partner, uint8_t *powerRole, uint8_t *ruleIndex)

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

Parameters

- **attributes** – Variable to be filled with packed values.
- **partner** – Indicates which side of the PD connection.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

Returns

Returns *common entity* return values

3.3.16 RCServo Class

See the *RCServo Entity* for generic information.

class **RCServoClass** : public Acroname::BrainStem::EntityClass

RCServoClass: Interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

Public Functions

RCServoClass (void)

Constructor.

virtual ~**RCServoClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the RCServo Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the RCServo entity to be addressed.

aErr **setEnabled** (const uint8_t enable)

Enable the servo channel

Parameters

enable – The state to be set. 0 is disabled, 1 is enabled.

Returns

Returns *common entity* return values

aErr **getEnabled** (uint8_t *enable)

Get the enable status of the servo channel.

Parameters

enable – The current enable status of the servo entity. 0 is disabled, 1 is enabled.

Returns

Returns *common entity* return values

aErr **setPosition** (const uint8_t position)

Set the position of the servo channel

Parameters

position – The position to be set. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns

Returns *common entity* return values

aErr **getPosition** (uint8_t *position)

Get the position of the servo channel

Parameters

position – The current position of the servo channel. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns

Returns *common entity* return values

aErr **setReverse** (const uint8_t reverse)

Set the output to be reversed on the servo channel

Parameters

reverse – Reverses the value set by “setPosition”. For example, if the position is set to 64 (1ms pulse) the output will now be 192 (2ms pulse), however “getPosition” will return the set value of 64. 0 = not reversed, 1 = reversed.

Returns

Returns *common entity* return values

aErr **getReverse** (uint8_t *reverse)

Get the reverse status of the servo channel

Parameters

reverse – The current reverse status of the servo entity. 0 = not reversed, 1 = reversed.

Returns

Returns *common entity* return values

3.3.17 Rail Class

See the *Rail Entity* for generic information.

class **RailClass** : public Acroname::BrainStem::EntityClass

RailClass: Provides power rail functionality on certain modules. The *RailClass* can be used to control power to downstream devices. It has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

Public Functions

RailClass (void)

Constructor.

virtual **~RailClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Rail Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Rail entity to be addressed.

aErr **getCurrent** (int32_t *microamps)

Get the rail current.

Parameters

microamps – The current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **setCurrentSetpoint** (const int32_t microamps)

Set the rail supply current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Parameters

microamps – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

Returns

Returns *common entity* return values

aErr **getCurrentSetpoint** (int32_t *microamps)

Get the rail setpoint current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Parameters

microamps – The current in micro-amps (1 == 1e-6A) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setCurrent interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setCurrentLimit** (const int32_t microamps)

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

Parameters

microamps – The current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCurrentLimit** (int32_t *microamps)

Get the rail current limit setting. (Check product datasheet to see if this feature is available)

Parameters

microamps – The current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getTemperature** (int32_t *microcelsius)

Get the rail temperature.

Parameters

microcelsius – The measured temperature associated with the rail in micro-Celsius (1 == 1e-6°C). The temperature may be associated with the module's internal rail circuitry or an externally connected temperature sensors. Refer to the module datasheet for definition of the temperature measurement location and specific capabilities.

Returns

Returns *common entity* return values

aErr **getEnable** (uint8_t *enable)

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Parameters

enable – true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage

Returns

Returns *common entity* return values

aErr **setEnable** (const uint8_t enable)

Set the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Parameters

enable – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

Returns

Returns *common entity* return values

aErr **getVoltage** (int32_t *microvolts)

Get the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setVoltageSetpoint** (const int32_t microvolts)

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

Returns

Returns *common entity* return values

aErr **getVoltageSetpoint** (int32_t *microvolts)

Get the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setVoltageMinLimit** (const int32_t microvolts)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getVoltageMinLimit** (int32_t *microvolts)

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **setVoltageMaxLimit** (const int32_t microvolts)

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getVoltageMaxLimit** (int32_t *microvolts)

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getPower** (int32_t *microwatts)

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

microwatts – The power in micro-watts (1 == 1e-6W) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setPowerSetpoint** (const int32_t microwatts)

Set the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

microwatts – The power in micro-watts (1 == 1e-6W) to be supplied by the rail.

Returns

Returns *common entity* return values

aErr **getPowerSetpoint** (int32_t *microwatts)

Get the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

microwatts – The power in micro-watts (1 == 1e-6W) the rail is trying to achieve. On

some modules this is a measured value so it may not exactly match what was previously set via the `setPower` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr `setPowerLimit` (const int32_t microwatts)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microwatts – The power in micro-watts (1 == 1e-6W).

Returns

Returns *common entity* return values

aErr `getPowerLimit` (int32_t *microwatts)

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microwatts – The power in micro-watts (1 == 1e-6W).

Returns

Returns *common entity* return values

aErr `getResistance` (int32_t *microohms)

Get the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

microohms – The resistance in micro-ohms (1 == 1e-6Ohms) currently drawn by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the `setResistance` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr `setResistanceSetpoint` (const int32_t microohms)

Set the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

microohms – The resistance in micro-ohms (1 == 1e-6Ohms) to be drawn by the rail.

Returns

Returns *common entity* return values

aErr `getResistanceSetpoint` (int32_t *microohms)

Get the rail setpoint resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

microohms – The resistance in micro-ohms (1 == 1e-6Ohms) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the `setResistance` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setKelvinSensingEnable** (const uint8_t enable)

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

enable – enable or disable kelvin sensing.

Returns

Returns *common entity* return values

aErr **getKelvinSensingEnable** (uint8_t *enable)

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

enable – Kelvin sensing is enabled or disabled.

Returns

Returns *common entity* return values

aErr **getKelvinSensingState** (uint8_t *state)

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

state – Kelvin sensing is enabled or disabled.

Returns

Returns *common entity* return values

aErr **setOperationalMode** (const uint8_t mode)

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

Parameters

mode – The operational mode to employ.

Returns

Returns *common entity* return values

aErr **getOperationalMode** (uint8_t *mode)

Determine the current operational mode of the system. Refer to the module datasheet for definition of the rail operational mode capabilities.

Parameters

mode – The current operational mode setting.

Returns

Returns *common entity* return values

aErr **getOperationalState** (uint32_t *state)

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

Parameters

state – The current operational state, hardware configuration, faults, and operating mode.

Returns

Returns *common entity* return values

aErr **clearFaults** (void)

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

Returns

Returns *common entity* return values

3.3.18 Relay Class

See the *Relay Entity* for generic information.

class **RelayClass** : public Acroname::BrainStem::EntityClass

RelayClass: Interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

Public Functions

RelayClass (void)

Constructor.

virtual **~RelayClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Relay Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Relay entity to be addressed.

aErr **setEnabled** (const uint8_t enable)

Set the enable/disable state.

Parameters

enable – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns *common entity* return values

aErr **getEnabled** (uint8_t *enabled)

Get the state.

Parameters

enabled – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns *common entity* return values

aErr **getVoltage** (int32_t *microvolts)

Get the scaled micro volt value with reference to ground.

Note: Not all modules provide 32 bits of accuracy. Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

microvolts – 32 bit signed integer (in micro Volts) based on the boards ground and reference voltages.

Returns

Returns *common entity* return values

3.3.19 Signal Class

See the *Signal Entity* for generic information.

class **SignalClass** : public Acroname::BrainStem::EntityClass

SignalClass: Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Public Functions

SignalClass (void)

Constructor.

virtual ~**SignalClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Signal Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Signal entity to be addressed.

aErr **setEnable** (const uint8_t enable)

Enable/Disable the signal output.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr **getEnable** (uint8_t *enable)

Get the Enable/Disable of the signal.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr **setInvert** (const uint8_t invert)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert – True to invert, false for normal mode.

Returns

Returns *common entity* return values

aErr **getInvert** (uint8_t *invert)

Get the invert status the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert – True to invert, false for normal mode.

Returns

Returns *common entity* return values

aErr **setT3Time** (const uint32_t t3_nsec)

Set the signal period or T3 in nanoseconds.

Parameters

t3_nsec – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Returns

Returns *common entity* return values

aErr **getT3Time** (uint32_t *t3_nsec)

Get the signal period or T3 in nanoseconds.

Parameters

t3_nsec – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Returns

Returns *common entity* return values

aErr **setT2Time** (const uint32_t t2_nsec)

Set the signal active period or T2 in nanoseconds.

Parameters

t2_nsec – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Returns

Returns *common entity* return values

aErr **getT2Time** (uint32_t *t2_nsec)

Get the signal active period or T2 in nanoseconds.

Parameters

t2_nsec – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Returns

Returns *common entity* return values

3.3.20 Store Class

See the *Store Entity* for generic information.

class **StoreClass** : public Acroname::BrainStem::EntityClass

StoreClass: The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

Public Functions

StoreClass (void)

Constructor.

virtual ~**StoreClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Store Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Store entity to be addressed.

aErr **getSlotState** (const uint8_t slot, uint8_t *state)

Get slot state.

Parameters

- **slot** – The slot number.
- **state** – true: enabled, false: disabled.

Returns

Returns *common entity* return values

aErr **loadSlot** (const uint8_t slot, const uint8_t *buffer, const uint16_t bufferLength)

Load the slot.

Parameters

- **slot** – The slot number.
- **buffer** – The data.
- **bufferLength** – The data length.

Returns

Returns *common entity* return values

aErr **unloadSlot** (const uint8_t slot, const size_t bufferLength, uint8_t *buffer, size_t *unloadedLength)

Unload the slot data.

Parameters

- **slot** – The slot number.
- **bufferLength** – The length of buffer buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **buffer** – Byte array that the unloaded data will be placed into.
- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than dataLength.

Returns

Returns *common entity* return values

aErr **slotEnable** (const uint8_t slot)

Enable slot.

Parameters

slot – The slot number.

Returns

Returns *common entity* return values

aErr **slotDisable** (const uint8_t slot)

Disable slot.

Parameters

slot – The slot number.

Returns

Returns *common entity* return values

aErr **getSlotCapacity** (const uint8_t slot, size_t *capacity)

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold.

Parameters

- **slot** – The slot number.
- **capacity** – The slot capacity.

Returns

Returns *common entity* return values

aErr **getSlotSize** (const uint8_t slot, size_t *size)

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes.

Parameters

- **slot** – The slot number.
- **size** – The slot size.

Returns

Returns *common entity* return values

aErr **getSlotLocked** (const uint8_t slot, uint8_t *lock)

Gets the current lock state of the slot Allows for write protection on a slot.

Parameters

- **slot** – The slot number
- **lock** – Variable to be filled with the locked state.

Returns

Returns *common entity* return values

aErr **setSlotLocked** (const uint8_t slot, const uint8_t lock)

Sets the locked state of the slot Allows for write protection on a slot.

Parameters

- **slot** – The slot number
- **lock** – state to be set.

Returns

Returns *common entity* return values

3.3.21 System Class

See the *System Entity* for generic information.

class **SystemClass** : public Acroname::BrainStem::EntityClass

SystemClass: The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc.

Public Functions

SystemClass (void)

Constructor.

virtual **~SystemClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the System Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the System entity to be addressed.

aErr **getModule** (uint8_t *address)

Get the current address the module uses on the BrainStem network.

Parameters

address – The address the module is using on the BrainStem network.

Returns

Returns *common entity* return values

aErr **getModuleBaseAddress** (uint8_t *address)

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

Parameters

address – The address the module is using on the BrainStem network.

Returns

Returns *common entity* return values

aErr **setRouter** (const uint8_t address)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

Parameters

address – The router address to be used.

Returns

Returns *common entity* return values

aErr **getRouter** (uint8_t *address)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

Parameters

address – The address.

Returns

Returns *common entity* return values

aErr **setHBInterval** (const uint8_t interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

Parameters

interval – The desired heartbeat delay.

Returns

Returns *common entity* return values

aErr **getHBInterval** (uint8_t *interval)

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

Parameters

interval – The current heartbeat delay.

Returns

Returns *common entity* return values

aErr **setLED** (const uint8_t ledOn)

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

Parameters

ledOn – true: turn the LED on, false: turn LED off.

Returns

Returns *common entity* return values

aErr **getLED** (uint8_t *ledOn)

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

Parameters

ledOn – true: LED on, false: LED off.

Returns

Returns *common entity* return values

aErr **setLEDMaxBrightness** (const uint8_t brightness)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Parameters

brightness – Brightness value relative to 255

Returns

Returns *common entity* return values

aErr **getLEDMaxBrightness** (uint8_t *brightness)

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum).

Parameters

brightness – Brightness value relative to 255

Returns

Returns *common entity* return values

aErr **setBootSlot** (const uint8_t slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

Parameters

slot – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

Returns

Returns *common entity* return values

aErr **getBootSlot** (uint8_t *slot)

Get the store slot which is mapped when the module boots.

Parameters

slot – The slot number in aSTORE_INTERNAL that is mapped after the module boots.

Returns

Returns *common entity* return values

aErr **getVersion** (uint32_t *build)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

Parameters

build – The build version date code.

Returns

Returns *common entity* return values

aErr **getBuild** (uint32_t *build)

Get the modules firmware build number. The build number is a unique hash assigned to a specific firmware.

Parameters

build – Variable to be filled with build.

Returns

Returns *common entity* return values

aErr **getModel** (uint8_t *model)

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem

model codes". Other codes are be used by Acroname for proprietary module types.

Parameters

model – The module's model enumeration.

Returns

Returns *common entity* return values

aErr **getHardwareVersion** (uint32_t *hardwareVersion)

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

Parameters

hardwareVersion – The module's hardware version information.

Returns

Returns *common entity* return values

aErr **getSerialNumber** (uint32_t *serialNumber)

Get the module's serial number. The serial number is a unique 32 bit integer which is usually communicated in hexadecimal format.

Parameters

serialNumber – The module's serial number.

Returns

Returns *common entity* return values

aErr **save** (void)

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address.

Returns

Returns *common entity* return values

aErr **reset** (void)

Reset the system. A return value of aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

Returns

Returns *common entity* return values

aErr **logEvents** (void)

Saves system log events to a slot defined by the module (usually ram slot 0).

Returns

Returns *common entity* return values

aErr **getUptime** (uint32_t *uptimeCounter)

Get the module's accumulated uptime in minutes

Parameters

uptimeCounter – The module's accumulated uptime in minutes.

Returns

Returns *common entity* return values

aErr **getTemperature** (int32_t *temperature)

Get the module's current temperature in micro-C

Parameters

temperature – The module's system temperature in micro-C

Returns

Returns *common entity* return values

aErr **getMinimumTemperature** (int32_t *minTemperature)

Get the module's minimum temperature ever recorded in micro-C (uC). This value will persists through a power cycle.

Parameters

minTemperature – The module's minimum system temperature in micro-C

Returns

Returns *common entity* return values

aErr **getMaximumTemperature** (int32_t *maxTemperature)

Get the module's maximum temperature ever recorded in micro-C (uC). This value will persists through a power cycle.

Parameters

maxTemperature – The module's maximum system temperature in micro-C

Returns

Returns *common entity* return values

aErr **getInputVoltage** (uint32_t *inputVoltage)

Get the module's input voltage.

Parameters

inputVoltage – The module's input voltage reported in microvolts.

Returns

Returns *common entity* return values

aErr **getInputCurrent** (uint32_t *inputCurrent)

Get the module's input current.

Parameters

inputCurrent – The module's input current reported in microamps.

Returns

Returns *common entity* return values

aErr `getModuleHardwareOffset` (uint8_t *offset)

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

Parameters

offset – The module address offset.

Returns

Returns *common entity* return values

aErr `setModuleSoftwareOffset` (const uint8_t address)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

aErr `getModuleSoftwareOffset` (uint8_t *address)

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

aErr `getRouterAddressSetting` (uint8_t *address)

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

aErr `routeToMe` (const uint8_t enable)

Enables/Disables the route to me function. This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

Parameters

enable – Enable or disable of the route to me function 1 = enable.

Returns

Returns *common entity* return values

aErr **getPowerLimit** (uint32_t *power)

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

Parameters

power – The available power in milli-Watts (mW, 1 t)

Returns

Returns *common entity* return values

aErr **getPowerLimitMax** (uint32_t *power)

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters

power – Variable to be filled with the power limit in milli-Watts (mW)

Returns

Returns *common entity* return values

aErr **setPowerLimitMax** (const uint32_t power)

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters

power – Limit in milli-Watts (mW) to be set.

Returns

Returns *common entity* return values

aErr **getPowerLimitState** (uint32_t *state)

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit().

Parameters

state – Variable to be filled with the state.

Returns

Returns *common entity* return values

aErr **getUnregulatedVoltage** (int32_t *voltage)

Gets the voltage present at the unregulated port.

Parameters

voltage – Variable to be filled with the voltage in micro-Volts (uV).

Returns

Returns *common entity* return values

aErr **getUnregulatedCurrent** (int32_t *current)

Gets the current passing through the unregulated port.

Parameters

current – Variable to be filled with the current in micro-Amps (uA).

Returns

Returns *common entity* return values

aErr **getInputPowerSource** (uint8_t *source)

Provides the source of the current power source in use.

Parameters

source – Variable to be filled with enumerated representation of the source.

Returns

Returns *common entity* return values

aErr **getInputPowerBehavior** (uint8_t *behavior)

Gets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Parameters

behavior – Variable to be filled with an enumerated value representing behavior.

Returns

Returns *common entity* return values

aErr **setInputPowerBehavior** (const uint8_t behavior)

Sets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Parameters

behavior – An enumerated representation of behavior to be set.

Returns

Returns *common entity* return values

aErr **getInputPowerBehaviorConfig** (uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setInputPowerBehaviorConfig** (const uint32_t *buffer, const size_t bufferLength)

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getName** (uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setName** (const uint8_t *buffer, const size_t bufferLength)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **resetDeviceToFactoryDefaults** (void)

Resets the device to it factory default configuration.

Returns

Returns *common entity* return values

aErr **getLinkInterface** (uint8_t *linkInterface)

Gets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters

linkInterface – Variable to be filled with an enumerated value representing interface.

- 0 = Auto= systemLinkAuto
- 1 = Control Port = systemLinkUSBControl
- 2 = Hub Upstream Port = systemLinkUSBHub

Returns

Returns *common entity* return values

aErr **setLinkInterface** (const uint8_t linkInterface)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters

linkInterface – An enumerated representation of interface to be set.

- 0 = Auto= systemLinkAuto
- 1 = Control Port = systemLinkUSBControl
- 2 = Hub Upstream Port = systemLinkUSBHub

Returns

Returns *common entity* return values

aErr **getErrors** (uint32_t *errors)

Gets any system level errors. Calling this function will clear the current errors. If the error persists it will be set again.

Parameters

errors – Bit mapped field representing the devices errors

Returns

Returns *common entity* return values

aErr **getProtocolFeatures** (uint32_t *features)

Gets the firmware protocol features

Parameters

features – Value representing the firmware protocol features

Returns

Returns *common entity* return values

3.3.22 Temperature Class

See the *Temperature Entity* for generic information.

class **TemperatureClass** : public Acroname::BrainStem::EntityClass

TemperatureClass: This entity is only available on certain modules, and provides a temperature reading in microcelsius.

Public Functions

TemperatureClass (void)

Constructor.

virtual **~TemperatureClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Temperature Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Temperature entity to be addressed.

aErr **getValue** (int32_t *temp)

Get the modules temperature in micro-C

Parameters

temp – The temperature in micro-Celsius (1 == 1e-6C).

Returns

Returns *common entity* return values

aErr **getValueMin** (int32_t *minTemp)

Get the module's minimum temperature in micro-C since the last power cycle.

Parameters

minTemp – The module's minimum temperature in micro-C

Returns

Returns *common entity* return values

aErr **getValueMax** (int32_t *maxTemp)

Get the module's maximum temperature in micro-C since the last power cycle.

Parameters

maxTemp – The module's maximum temperature in micro-C

Returns

Returns *common entity* return values

3.3.23 Timer Class

See the *Timer Entity* for generic information.

class **TimerClass** : public Acroname::BrainStem::EntityClass

TimerClass: The Timer Class provides access to a simple scheduler. The timer can set to fire only once, or to repeat at a certain interval. Additionally, a timer entity can execute custom Reflex routines upon firing.

Public Functions

TimerClass (void)

Constructor.

virtual **~TimerClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the Timer Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the Timer entity to be addressed.

aErr **getExpiration** (uint32_t *usecDuration)

Get the currently set expiration time in microseconds. This is not a “live” timer. That is, it shows the expiration time originally set with setExpiration; it does not “tick down” to show the time remaining before expiration.

Parameters

usecDuration – The timer expiration duration in microseconds.

Returns

Returns *common entity* return values

aErr **setExpiration** (const uint32_t usecDuration)

Set the expiration time for the timer entity. When the timer expires, it will fire the associated timer[index]() reflex.

Parameters

usecDuration – The duration before timer expiration in microseconds.

Returns

Returns *common entity* return values

aErr **getMode** (uint8_t *mode)

Get the mode of the timer which is either single or repeat mode.

Parameters

mode – The mode of the timer. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Returns

Returns *common entity* return values

aErr **setMode** (const uint8_t mode)

Set the mode of the timer which is either single or repeat mode.

Parameters

mode – The mode of the timer. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Returns

Returns *common entity* return values

3.3.24 UART Class

See the *UART Entity* for generic information.

class **UARTClass** : public Acroname::BrainStem::EntityClass

UARTClass: A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port. The UART Class allows the enabling and disabling of the UART data lines.

Public Functions

UARTClass (void)

Constructor.

virtual **~UARTClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the UART Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the UART entity to be addressed.

aErr **setEnabled** (const uint8_t enabled)

Enable the UART channel.

Parameters

enabled – true: enabled, false: disabled.

Returns

Returns *common entity* return values

aErr **getEnable** (uint8_t *enabled)

Get the enabled state of the uart.

Parameters

enabled – true: enabled, false: disabled.

Returns

Returns *common entity* return values

aErr **setBaudRate** (const uint32_t rate)

Set the UART baud rate. If zero, automatic baud rate selection is used.

Parameters

rate – baud rate.

Returns

Returns *common entity* return values

aErr **getBaudRate** (uint32_t *rate)

Get the UART baud rate. If zero, automatic baud rate selection is used.

Parameters

rate – Pointer variable to be filled with baud rate.

Returns

Returns *common entity* return values

aErr **setProtocol** (const uint8_t protocol)

Set the UART protocol.

Parameters

protocol – An enumeration of serial protocols.

Returns

Returns *common entity* return values

aErr **getProtocol** (uint8_t *protocol)

Get the UART protocol.

Parameters

protocol – Pointer to where result is placed.

Returns

Returns *common entity* return values

aErr **setLinkChannel** (const uint8_t channel)

Set the index of another UART Entity that should be linked to this UART.

If set to the index of this entity, the channel will not be linked. If set to the index of another UART entity, data will be sent between the two UART entities with no additional processing.

Parameters

channel – Index of the UART Entity to link

Returns

Returns *common entity* return values

aErr **getLinkChannel** (uint8_t *channel)

Gets the index of the UART Entity that this entity is linked to.

Parameters

channel – Pointer to where result is placed.

Returns

Returns *common entity* return values

aErr **setStopBits** (const uint8_t stopBits)

Set the UART stop bit configuration

Parameters

stopBits – Stop Bits of UART Channel. Allowed options:

- uartStopBits_1_Value
- uartStopBits_1p5_Value
- uartStopBits_2_Value

Returns

Returns *common entity* return values

aErr **getStopBits** (uint8_t *stopBits)

Set the UART stop bit configuration

Parameters

stopBits – Pointer to where result is placed. Possible values:

- uartStopBits_1_Value
- uartStopBits_1p5_Value
- uartStopBits_2_Value

Returns

Returns *common entity* return values

aErr **setParity** (const uint8_t parity)

Set the UART parity.

Parameters

parity – Parity of UART Channel. Allowed options:

- uartParity_None_Value
- uartParity_Odd_Value
- uartParity_Even_Value
- uartParity_Mark_Value
- uartParity_Space_Value

Returns

Returns *common entity* return values

aErr **getParity** (uint8_t *parity)

Get the UART parity.

Parameters

parity – Pointer variable to be filled with value. Possible values:

- uartParity_None_Value
- uartParity_Odd_Value
- uartParity_Even_Value
- uartParity_Mark_Value

- uartParity_Space_Value

Returns

Returns *common entity* return values

aErr **setDataBits** (const uint8_t dataBits)

Set the number of bits per character

Parameters

dataBits – Data Bits of UART Channel.

Returns

Returns *common entity* return values

aErr **getDataBits** (uint8_t *dataBits)

Get the number of bits per character

Parameters

dataBits – Pointer to where result is placed.

Returns

Returns *common entity* return values

aErr **setFlowControl** (const uint8_t flowControl)

Set the UART flow control configuration

Parameters

flowControl – Flow Control of UART Channel as a bitmask. Allowed bits:

- uartFlowControl_RTS_CTS_Bit
- uartFlowControl_DSR_DTR_Bit
- uartFlowControl_XON_XOFF_Bit

Returns

Returns *common entity* return values

aErr **getFlowControl** (uint8_t *flowControl)

Set the UART flow control configuration

Parameters

flowControl – Pointer to bitmask where result is placed. Possible bits:

- uartFlowControl_RTS_CTS_Bit
- uartFlowControl_DSR_DTR_Bit
- uartFlowControl_XON_XOFF_Bit

Returns

Returns *common entity* return values

aErr **getCapableProtocols** (uint32_t *protocols)

Returns a bitmask containing a list of protocols that this UART entity is allowed to select. This does not guarantee that selecting a protocol with “setProtocol” will have an available resource.

Parameters

protocols – Bitmask containing list of protocols that may be selected. The value of the uartProtocol is mapped to the bit index (e.g. uartProtocol_Undefined is bit 0, uartProtocol_ExtronResponder_Value is bit 1, etc.)

Returns

Returns *common entity* return values

aErr **getAvailableProtocols** (uint32_t *protocols)

Returns a bitmask containing a list of protocols that this UART entity is capable of selecting, and has an available protocol resource to assign.

Parameters

protocols – Bitmask containing list of protocols that are available to select. The value of the uartProtocol is mapped to the bit index (e.g. uartProtocol_Undefined is bit 0, uartProtocol_ExtronResponder_Value is bit 1, etc.)

Returns

Returns *common entity* return values

3.3.25 USB Class

See the *USB Entity* for generic information.

class **USBClass** : public Acroname::BrainStem::EntityClass

USBClass: The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

Public Functions

USBClass (void)

Constructor.

virtual **~USBClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the USB Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the USB entity to be addressed.

aErr **setPortEnable** (const uint8_t channel)

Enable both power and data lines for a port.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setPortDisable** (const uint8_t channel)

Disable both power and data lines for a port.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setDataEnable** (const uint8_t channel)

Enable the only the data lines for a port without changing the state of the power line.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setDataDisable** (const uint8_t channel)

Disable only the data lines for a port without changing the state of the power line.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setHiSpeedDataEnable** (const uint8_t channel)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setHiSpeedDataDisable** (const uint8_t channel)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setSuperSpeedDataEnable** (const uint8_t channel)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setSuperSpeedDataDisable** (const uint8_t channel)

Disable only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setPowerEnable** (const uint8_t channel)

Enable only the power line for a port without changing the state of the data lines.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setPowerDisable** (const uint8_t channel)

Disable only the power line for a port without changing the state of the data lines.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **getPortCurrent** (const uint8_t channel, int32_t *microamps)

Get the current through the power line for a port.

Parameters

- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getPortVoltage** (const uint8_t channel, int32_t *microvolts)

Get the voltage on the power line for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in microvolts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getHubMode** (uint32_t *mode)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

Parameters

mode – The USB hub mode.

Returns

Returns *common entity* return values

aErr **setHubMode** (const uint32_t mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

Parameters

mode – The USB hub mode.

Returns

Returns *common entity* return values

aErr **clearPortErrorStatus** (const uint8_t channel)

Clear the error status for the given port.

Parameters

channel – The port to clear error status for.

Returns

Returns *common entity* return values

aErr **getUpstreamMode** (uint8_t *mode)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

Parameters

mode – The Upstream port mode.

Returns

Returns *common entity* return values

aErr **setUpstreamMode** (const uint8_t mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0, usbUpstreamModePort1, and usbUpstreamModeNone.

Parameters

mode – The Upstream port mode.

Returns

Returns *common entity* return values

aErr **getUpstreamState** (uint8_t *state)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

Parameters

state – The Upstream port state.

Returns

Returns *common entity* return values

aErr **setEnumerationDelay** (const uint32_t ms_delay)

Set the inter-port enumeration delay in milliseconds.

Parameters

ms_delay – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

Returns *common entity* return values

aErr **getEnumerationDelay** (uint32_t *ms_delay)

Get the inter-port enumeration delay in milliseconds.

Parameters

ms_delay – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

Returns *common entity* return values

aErr **setPortCurrentLimit** (const uint8_t channel, const uint32_t microamps)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a stem.system.save() call.

Parameters

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

Returns

Returns *common entity* return values

aErr **getPortCurrentLimit** (const uint8_t channel, uint32_t *microamps)

Get the current limit for the port.

Parameters

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

Returns

Returns *common entity* return values

aErr **setPortMode** (const uint8_t channel, const uint32_t mode)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of usbPortMode.

Parameters

- **channel** – USB downstream channel to set the mode on.
- **mode** – The port mode setting as packed bit field.

Returns

Returns *common entity* return values

aErr **getPortMode** (const uint8_t channel, uint32_t *mode)

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of usbPortMode.

Parameters

- **channel** – USB downstream channel.
- **mode** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Returns

Returns *common entity* return values

aErr **getPortState** (const uint8_t channel, uint32_t *state)

Get the current State for the Port.

Parameters

- **channel** – USB downstream channel.
- **state** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Returns

Returns *common entity* return values

aErr **getPortError** (const uint8_t channel, uint32_t *error)

Get the current error for the Port.

Parameters

- **channel** – USB downstream channel.

- **error** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Returns

Returns *common entity* return values

aErr **setUpstreamBoostMode** (const uint8_t setting)

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Parameters

setting – Upstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **setDownstreamBoostMode** (const uint8_t setting)

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Parameters

setting – Downstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **getUpstreamBoostMode** (uint8_t *setting)

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Parameters

setting – The current Upstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **getDownstreamBoostMode** (uint8_t *setting)

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Parameters

setting – The current Downstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **getDownstreamDataSpeed** (const uint8_t channel, uint8_t *speed)

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

Parameters

- **channel** – USB downstream channel to check.

- **speed** – Filled with the current port data speed
 - N/A: usbDownstreamDataSpeed_na = 0
 - Hi Speed: usbDownstreamDataSpeed_hs = 1
 - SuperSpeed: usbDownstreamDataSpeed_ss = 2

Returns

Returns *common entity* return values

aErr **setConnectMode** (const uint8_t channel, const uint8_t mode)

Sets the connect mode of the switch.

Parameters

- **channel** – The USB sub channel.
- **mode** – The connect mode
 - usbManualConnect = 0
 - usbAutoConnect = 1

Returns

Returns *common entity* return values

aErr **getConnectMode** (const uint8_t channel, uint8_t *mode)

Gets the connect mode of the switch.

Parameters

- **channel** – The USB sub channel.
- **mode** – The current connect mode

Returns

Returns *common entity* return values

aErr **setCC1Enable** (const uint8_t channel, const uint8_t enable)

Set Enable/Disable on the CC1 line.

Parameters

- **channel** – USB channel.
- **enable** – State to be set
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCC1Enable** (const uint8_t channel, uint8_t *pEnable)

Get Enable/Disable on the CC1 line.

Parameters

- **channel** – USB channel.
- **pEnable** – State to be filled
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **setCC2Enable** (const uint8_t channel, const uint8_t enable)

Set Enable/Disable on the CC2 line.

Parameters

- **channel** – USB channel.
- **enable** – State to be filled
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCC2Enable** (const uint8_t channel, uint8_t *pEnable)

Get Enable/Disable on the CC2 line.

Parameters

- **channel** – USB channel.
- **pEnable** – State to be filled
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCC1Current** (const uint8_t channel, int32_t *microamps)

Get the current through the CC1 for a port.

Parameters

- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC2Current** (const uint8_t channel, int32_t *microamps)

Get the current through the CC2 for a port.

Parameters

- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC1Voltage** (const uint8_t channel, int32_t *microvolts)

Get the voltage of CC1 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC2Voltage** (const uint8_t channel, int32_t *microvolts)

Get the voltage of CC2 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **setSBUEnable** (const uint8_t channel, const uint8_t enable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

Parameters

- **channel** – The USB sub channel.
- **enable** – The state to be set
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getSBUEnable** (const uint8_t channel, uint8_t *pEnable)

Get the Enable/Disable status of the SBU

Parameters

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of the SBU

Returns

Returns *common entity* return values

aErr **setCableFlip** (const uint8_t channel, const uint8_t enable)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters

- **channel** – The USB sub channel.
- **enable** – The state to be set The state to be set
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCableFlip** (const uint8_t channel, uint8_t *pEnable)

Get Cable flip setting.

Parameters

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of cable flip.

Returns

Returns *common entity* return values

aErr **setAltModeConfig** (const uint8_t channel, const uint32_t configuration)

Set USB Alt Mode Configuration.

Parameters

- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

Returns

Returns *common entity* return values

aErr **getAltModeConfig** (const uint8_t channel, uint32_t *configuration)

Get USB Alt Mode Configuration.

Parameters

- **channel** – The USB sub channel
- **configuration** – The USB configuration for the given channel.

Returns

Returns *common entity* return values

aErr **getSBU1Voltage** (const uint8_t channel, int32_t *microvolts)

Get the voltage of SBU1 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getSBU2Voltage** (const uint8_t channel, int32_t *microvolts)

Get the voltage of SBU2 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

3.3.26 USBSystem Class

See the *USBSystem Entity* for generic information.

class **USBSystemClass** : public Acroname::BrainStem::EntityClass

USBSystemClass: The USBSystem class provides high level control of the lower level Port Class.

Subclassed by *aMTMIOSerial::HubClass*, *aUSBExt3c::HubClass*, *aUSBHub2x4::HubClass*, *aUSBHub3c::HubClass*, *aUSBHub3p::HubClass*

Public Functions

USBSystemClass (void)

Constructor.

virtual **~USBSystemClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the USBSystem Class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the USBSystem entity to be addressed.

aErr **getUpstream** (uint8_t *port)

Gets the upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstream** (const uint8_t port)

Sets the upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getEnumerationDelay** (uint32_t *msDelay)

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Parameters

msDelay – the current inter-port delay in milliseconds.

Returns

Returns *common entity* return values

aErr **setEnumerationDelay** (const uint32_t msDelay)

Sets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Parameters

msDelay – The delay in milliseconds to be applied between port enables

Returns

Returns *common entity* return values

aErr **getDataRoleList** (uint32_t *roleList)

Gets the data role of all ports with a single call Equivalent to calling *PortClass::getDataRole()* on each individual port.

Parameters

roleList – A bit packed representation of the data role for all ports.

Returns

Returns *common entity* return values

aErr **getEnabledList** (uint32_t *enabledList)

Gets the current enabled status of all ports with a single call. Equivalent to calling *PortClass::setEnabled()* on each port.

Parameters

enabledList – Bit packed representation of the enabled status for all ports.

Returns

Returns *common entity* return values

aErr **setEnabledList** (const uint32_t enabledList)

Sets the enabled status of all ports with a single call. Equivalent to calling *PortClass::setEnabled()* on each port.

Parameters

enabledList – Bit packed representation of the enabled status for all ports to be applied.

Returns

Returns *common entity* return values

aErr **getModeList** (uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the current mode of all ports with a single call. Equivalent to calling *PortClass::getMode()* on each port.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setModeList** (const uint32_t *buffer, const size_t bufferLength)

Sets the mode of all ports with a single call. Equivalent to calling *PortClass::setMode()* on each port

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getStateList** (uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the state for all ports with a single call. Equivalent to calling *PortClass::getState()* on each port.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getPowerBehavior** (uint8_t *behavior)

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system, i.e. What happens when requested power greater than available power.

Parameters

behavior – Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setPowerBehavior** (const uint8_t behavior)

Sets the behavior of how available power is managed, i.e. What happens when requested power is greater than available power.

Parameters

behavior – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getPowerBehaviorConfig** (uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setPowerBehaviorConfig** (const uint32_t *buffer, const size_t bufferLength)

Sets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getDataRoleBehavior** (uint8_t *behavior)

Gets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters

behavior – Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataRoleBehavior** (const uint8_t behavior)

Sets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters

behavior – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getDataRoleBehaviorConfig** (uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine priority host priority.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setDataRoleBehaviorConfig** (const uint32_t *buffer, const size_t bufferLength)

Sets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine host priority.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getSelectorMode** (uint8_t *mode)

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters

mode – Variable to be filled with the selector mode

Returns

Returns *common entity* return values

aErr **setSelectorMode** (const uint8_t mode)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters

mode – Mode to be set.

Returns

Returns *common entity* return values

aErr **getUpstreamHS** (uint8_t *port)

Gets the USB HighSpeed upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstreamHS** (const uint8_t port)

Sets the USB HighSpeed upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getUpstreamSS** (uint8_t *port)

Gets the USB SuperSpeed upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstreamSS** (const uint8_t port)

Sets the USB SuperSpeed upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getOverride** (uint32_t *overrides)

Gets the current enabled overrides

Parameters

overrides – Bit mapped representation of the current override configuration.

Returns

Returns *common entity* return values

aErr **setOverride** (const uint32_t overrides)

Sets the current enabled overrides

Parameters

overrides – Overrides to be set in a bit mapped representation.

Returns

Returns *common entity* return values

aErr **setDataHSMaxDataRate** (const uint32_t datarate)

Sets the USB HighSpeed Max datarate

Parameters

datarate – Maximum datarate for the USB HighSpeed signals.

Returns

Returns *common entity* return values

aErr **getDataHSMaXDataRate** (uint32_t *dataRate)

Gets the USB HighSpeed Max dataRate

Parameters

dataRate – Current maximum dataRate for the USB HighSpeed signals.

Returns

Returns *common entity* return values

aErr **setDataSSMaXDataRate** (const uint32_t dataRate)

Sets the USB SuperSpeed Max dataRate

Parameters

dataRate – Maximum dataRate for the USB SuperSpeed signals.

Returns

Returns *common entity* return values

aErr **getDataSSMaXDataRate** (uint32_t *dataRate)

Gets the USB SuperSpeed Max dataRate

Parameters

dataRate – Current maximum dataRate for the USB SuperSpeed signals.

Returns

Returns *common entity* return values

3.3.27 Link Class

class **Link**

LinkClass: The *Link* class provides an interface to a BrainStem link. The link is used to create interfaces to modules on a BrainStem network. The link represents a connection to the BrainStem network from a host computer. The link is always associated with a transport (e.g.: USB, Ethernet, etc.) and a link module, but there are several ways to make this association.

- a. The link can be fully specified with a transport and module serial number
- b. The link can be created by searching a transport and connecting to the first module found.

Calling connect on a link will start a connection to the module based on The link specification. Calling disconnect will disconnect the link from the the current connection.

Public Types

enum **STREAM_PACKET**

Enumeration of stream packet types.

Values:

enumerator **kSTREAM_PACKET_UNKNOWN**

enumerator **kSTREAM_PACKET_U8**

enumerator **kSTREAM_PACKET_U16**

enumerator **kSTREAM_PACKET_U32**

enumerator **kSTREAM_PACKET_BYTES**

enumerator **kSTREAM_PACKET_SUBINDEX_U8**

enumerator **kSTREAM_PACKET_SUBINDEX_U16**

enumerator **kSTREAM_PACKET_SUBINDEX_U32**

enumerator **kSTREAM_PACKET_LAST**

enum **STREAM_KEY**

Enumeration for element types within a stream key.

Values:

enumerator **STREAM_KEY_MODULE_ADDRESS**

enumerator **STREAM_KEY_CMD**

enumerator **STREAM_KEY_OPTION**

enumerator **STREAM_KEY_INDEX**

enumerator **STREAM_KEY_SUBINDEX**

enumerator **STREAM_KEY_LAST**

typedef enum Acroname::BrainStem::Link::STREAM_PACKET **STREAM_PACKET_t**

Enumeration of stream packet types.

typedef std::function<uint8_t(const *aPacket* *packet, void *pRef)> **streamCallback_t**

Function signature for streaming callbacks.

Param packet

reference to streaming packet

Param pRef

User provided reference

Return

non zero error code on failure. Return value is not currently used.

typedef struct Acroname::BrainStem::Link::StreamStatusEntry StreamStatusEntry_t
StreamStatusEntry structure - It contains members of streaming entries in the form of key value pairs. Keys are comprised of the devices module address, command, option, index, and subindex API values.

typedef enum Acroname::BrainStem::Link::STREAM_KEY STREAM_KEY_t
Enumeration for element types within a stream key.

Public Functions

aErr getConfig (aEtherConfig *config)

Gets the links current aEther configuration

Parameters

config - Pointer to the configuration to be filled

Returns

aErrNone on success; aErrParam if config is NULL

aErr setConfig (const aEtherConfig config)

Sets the links aEther configuration. Configuration must be applied BEFORE connecting

Parameters

config - Configuration to be applied

Returns

aErrNone on success. aErrPermission if the module is currently connected.

Link (const *linkSpec* linkSpecifier, const char *name = "Link")

Link Constructor. Takes a fully specified *linkSpec* pointer and creates a link instance with this specifier information.

Parameters

- **linkSpecifier** - The connection details for a specific module.
- **name** - A name for the link to be created. This name can be used to reference the link during later interactions.

Link (const char *name = "Link")

Link constructor without a specifier will most likely use the discoverAndConnect call to create a connection to a link module.

Parameters

name - A name for the link to be created.

~Link (void)

Destructor.

aErr discoverAndConnect (const *linkType* type, const uint32_t serialNumber = 0, const uint8_t model = 0)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will be made.

Parameters

- **type** - Transport on which to search for available BrainStem link modules. See the *transport* enum for supported transports.
- **serialNumber** - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

- **model** – Acroname model number for the device.

Return values

- **aErrBusy** – if the module is already in use.
- **aErrParam** – if the transport type is undefined.
- **aErrNotFound** – if the module cannot be found or if no modules found.
- **aErrNone** – If the connect was successful.

aErr connect (void)

Connect to a link with a fully defined specifier.

Return values

- **aErrBusy** – if the module is running, starting or stopping. Try again in a bit.
- **aErrDuplicate** – If the module is already connected and running.
- **aErrConnection** – If there was an error with the connection. User needs to disconnect, then reconnect.
- **aErrConfiguration** – If the link has an invalid *linkSpec*.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr connectThroughLinkModule (*Link* &link)

Connect using a pre-existing link. This member function will connect to the same BrainStem used by given *Link*. If a link module is found on the specified transport, a connection will be made.

Parameters

link – - Reference to the link to be used.

Return values

- **aErrInitialization** – If the referenced link does not exist yet.
- **aErrConnection** – If the connection could not be made.
- **aErrConfiguration** – If the device or connection is in properly configured.
- **aErrNone** – if the connect was successful.

bool isConnected (void)

Check to see if a module is connected. *isConnected* looks for a connection to an active module.

Returns

true: connected, false: not connected.

linkStatus getStatus (void)

Check the status of the module connection.

Returns

linkStatus (see aLink.h for status values)

aErr disconnect (void)

Disconnect from the BrainStem module.

Return values

- **aErrResource** – If there is no valid connection.
- **aErrConnection** – If the disconnect failed, due to a communication issue.
- **aErrNone** – If the disconnect was successful.

aErr reset (void)

Reset The underlying link stream.

Return values

- **aErrResource** – If there is no valid connection.
- **aErrConnection** – If the reset failed, due to a communication issue.
- **aErrNone** – If the reset was successful.

const char *getName (void)

Accessor for link Name. Returns a pointer to the string representing the link. This string is part of the link, and will be destroyed with it. If you need access to the link name beyond the life of the link, then copy the char* returned.

Returns

Pointer to character array containing the name of the link.

aErr getLinkSpecifier (*linkSpec* *spec)

Accessor for current link specification.

Parameters

spec - - an allocated empty link spec reference.

Returns

aErrNotFound - If no *linkSpec* set for current link.

aErr setLinkSpecifier (const *linkSpec* linkSpecifier)

Accessor Set current link specification.

Parameters

linkSpecifier - - The specifier that will replace the current spec.

Returns

aErrBusy - If link is currently connected.

aErr getModuleAddress (uint8_t *address)

Gets the module address of the module the link is connected too. A zero is returned if no module can not be determined or if the link is not connected.

aErr sendUEI (const *uei* packet)

Sends a BrainStem protocol UEI packet on the link. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

packet - The command UEI packet to send.

Return values

- **aErrConnection** - link not connected.
- **aErrParam** - data too long or short.
- **aErrPacket** - invalid module address.
- **aErrNone** - success.

aErr sendUEI (const *uei* packet, const uint8_t subindex)

Sends a BrainStem protocol UEI packet on the link where the packet contains a subindex. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- **packet** - The command UEI packet to send.
- **subindex** - The subindex of the command option.

Return values

- **aErrConnection** - link not connected.
- **aErrParam** - data too long or short.
- **aErrPacket** - invalid module address.
- **aErrNone** - success.

aErr receiveUEI (const uint8_t module, const uint8_t command, const uint8_t option, const uint8_t index, *uei* *packet)

Awaits receipt of the first available matching UEI packet from the link. The first four arguments describe the packet to wait for. When successful, the supplied uei ref is filled with the received UEI. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- **module** – The module address.
- **command** – The command.
- **option** – The uei option.
- **index** – The index of the uei entity.
- **packet** – The uei packet reference to be filled on success.

Return values

- **aErrConnection** – link not connected.
- **aErrPacket** – invalid module address.
- **aErrTimeout** – no packet available.
- **aErrNone** – success.

aErr **receiveUEI** (const uint8_t module, const uint8_t command, const uint8_t option, const uint8_t index, uei *packet, aPacketMatchPacketProc proc)

Awaits receipt of the first available matching UEI packet from the link. The first four arguments and proc describe the packet to wait for. When successful, the supplied uei ref is filled with the received UEI. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- **module** – The module address.
- **command** – The command.
- **option** – The uei option.
- **index** – The index of the uei entity.
- **packet** – The uei packet reference to be filled on success.
- **proc** – The callback used for determining a matching packet.

Return values

- **aErrConnection** – link not connected.
- **aErrPacket** – invalid module address.
- **aErrTimeout** – no packet available.
- **aErrNone** – success.

aErr **dropMatchingUEIPackets** (const uint8_t module, const uint8_t command, const uint8_t option, const uint8_t index)

Drops all existing queued packets that match. from the link. The arguments describe the packets to be matched This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- **module** – The module address.
- **command** – The command.
- **option** – The uei option.
- **index** – The index of the uei entity.

Return values

- **aErrConnection** – link not connected.
- **aErrPacket** – invalid module address.
- **aErrNone** – success.

aErr **sendPacket** (const uint8_t module, const uint8_t command, const uint8_t length, const uint8_t *data)

Sends a raw BrainStem protocol packet on the link. where the length does not include the module or the command. address byte and can be 0 to aBRAINSTEM_MAXPACKETBYTES - 1. This is an advanced interface, please see the relevant section of the reference manual for more information about BrainStem Packet protocol.

Parameters

- **module** – The address of the destination module.
- **command** – The length of the data being sent.

- **length** – The length of the data being sent.
- **data** – The data to send.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – data too long or short.
- **aErrPacket** – invalid module address.
- **aErrNone** – success.

aErr receivePacket (const uint8_t module, const uint8_t *match, uint8_t *length, uint8_t *data)

Awaits receipt of the first available matching raw BrainStem protocol packet from the link where the length does not include the module or command bytes and can be zero. The provided module and match array are compared to packets available and the first match is returned. The supplied data pointer must point to at least aBRAINSTEM_MAXPACKETBYTES - 1 bytes. When successful, the data is filled in with the packet data not including the module and command and the length pointer is updated with the length of the returned data.

This is an advanced interface, please see the relevant section of the reference manual for more information about BrainStem Packet protocol.

Parameters

- **module** – The module address.
- **match** – A byte array of the values to match for received packets.
- **length** – The length of the match data on entry and length of the returned data filled on success.
- **data** – The data filled on success.

Return values

- **aErrConnection** – link not connected.
- **aErrPacket** – invalid module address.
- **aErrTimeout** – no packet available.
- **aErrNone** – success.

aErr loadStoreSlot (const uint8_t module, const uint8_t store, const uint8_t slot, const uint8_t *pData, const size_t length)

Loads data into a BrainStem Slot. See the relevant section of the BrainStem reference for information about BrainStem Slots and Stores.

Parameters

- **module** – - *Module* address.
- **store** – - BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** – - The Slot within the Brainstem store to place the data.
- **pData** – - Pointer to a buffer containing the data to load.
- **length** – - The length in bytes of the data buffer to write.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this call was unable to successfully complete.
- **aErrNone** – success.

aErr unloadStoreSlot (const uint8_t module, const uint8_t store, const uint8_t slot, uint8_t *pData, const size_t dataLength, size_t *pNRead)

Unloads data from a BrainStem Slot. If there are no read.

reference for information about BrainStem Slots and Stores.

Parameters

- **module** -- *Module* address.
- **store** -- BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** -- The Slot within the Brainstem store to place the data.
- **pData** -- Pointer to a buffer with dataLength space in bytes that will be filled by the call.
- **dataLength** -- Expected length of the data, and at most the size of the pData buffer.
- **pNRead** -- The number of bytes actually read.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this call was unable to successfully complete.
- **aErrOverrun** – The read would overrun the buffer, i.e there is more data in the slot than the buffer can handle.
- **aErrNone** – success.

aErr **storeSlotSize** (const uint8_t module, const uint8_t store, const uint8_t slot, size_t *size)

Returns the current size of the data loaded in the slot specified.

Parameters

- **module** -- *Module* address.
- **store** -- BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** -- The Slot within the Brainstem store to place the data.
- **size** -- size in bytes of the data stored in the slot.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this request was unable to successfully complete.
- **aErrNone** – success.

aErr **storeSlotCapacity** (const uint8_t module, const uint8_t store, const uint8_t slot, size_t *capacity)

Returns the maximum data capacity of the slot specified.

Parameters

- **module** -- *Module* address.
- **store** -- BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** -- The Slot within the Brainstem store to place the data.
- **capacity** -- size in bytes of the data stored in the slot.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this request was unable to successfully complete.
- **aErrNone** – success.

aErr **enableStream** (const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const bool enable)

Enables streaming for the supplied criteria.

Parameters

- **moduleAddress** – Address to filter on.

- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - Enables streaming; False - disables streaming

aErr **isLinkStreaming** (const uint8_t moduleAddress, uint8_t *enabled)

Determines if the module is actively streaming. Does not indicate what is streaming, only if streaming is currently active.

Parameters

- **moduleAddress** – The devices module address.
- **enabled** – Variable to be populated.

Returns

Returns *common entity* return values

aErr **registerStreamCallback** (const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const bool enable, *streamCallback_t* cb, void *pRef)

Registers a callback function based on a specific module, cmd, option, and index.

Parameters

- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - installs/updates callback and ref; False - uninstalls callback
- **cb** – Callback to be executed when a new packet matching the criteria is received.
- **pRef** – Pointer to user reference for use inside the callback function.

Returns

aErrNotFound - Item not found (uninstalling only)

Returns

aErrNone - success

aErr **getStreamValue** (const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const uint8_t subindex, uint32_t *value)

Gets stream value based on the search criteria

Parameters

- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)

Returns

aErrStreamStale if the value has not been updated since the last read.

Returns

aErrNotFound if no such stream element exists.

Returns

aErrNone - success

aErr **getStreamStatus** (const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const uint8_t subindex, *StreamStatusEntry_t* *buffer, const size_t bufferLength, size_t *unloadedSize)

Gets all available stream values based on the search criteria.

Parameters

- **moduleAddress** – Address to filter on (supports Wildcards)

- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **subindex** – subindex to filter by (supports Wildcards)
- **buffer** – Buffer of user allocated memory to be filled with stream data Note:
[Link::getStreamKeyElement](#) should be used to decode the keys
- **bufferLength** – Number of elements the buffer can hold.
- **unloadedSize** – Number of elements that were placed in the buffer

Returns

aErrParam if status or unloadedSize is null

Returns

aErrNone - success

```
std::vector<uint64_t> filterActiveStreamKeys (const uint8_t moduleAddress, const
                                             uint8_t cmd, const uint8_t option,
                                             const uint8_t index, const uint8_t
                                             subindex, const bool acquireLock)
```

Provides a list of active stream keys based on the supplied criteria. Exposed for unit-testing purposes only.

Parameters

- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **acquireLock** – Option to acquire mutex before getting list elements.

Returns

List of keys meeting the search criteria

```
aErr enablePooledPackets (const bool enable)
```

Enables pooling of outgoing BrainStem packets. Allowing multiple packets to be packed into a single transaction frame.

Parameters

enable – True = Enables; False = Disables

```
aErr enablePacketLog (const char *logname)
```

Enable Packet logging.

Enable packet logging for this link. Enables the packet logging buffer, and writes packet traffic out to the file specified by logname.

Parameters

logname – the path and filename indicating where to write the packet log.

Returns

aErr returns appropriate errors if it fails to enable the packet log.

```
aErr disablePacketLog (void)
```

Disable Packet logging.

disable packet logging for this link. Disables the packet log.

Returns

aErr returns appropriate errors if it fails to disable the debug log.

```
aErr getFactoryData (const uint8_t module, const uint8_t command, uint8_t *pData,
                     const size_t dataLength, size_t *unloadedLength)
```

For Internal use only!

```
aErr setFactoryData (const uint8_t module, const uint8_t command, const uint8_t
                     *pData, const size_t dataLength)
```

For Internal use only!

Public Static Functions

```
static inline aErr sDiscover (const linkType type, aDiscoveryModuleFoundProc
                             cbLinkFound, void *vpCRef, const uint32_t
                             networkInterface = LOCALHOST_IP_ADDRESS)
```

Discover is called with a specified transport to search for link modules on that transport. The callback is called with a fully filled in specifier for any link module found. The sDiscover returns aErrNone if the discovery process is successful, regardless of if any links are found. An error is only returned if the link discovery process fails. Discovery can take some time. The callback will occur in the same thread context as this routine call.

Parameters

- **type** – Transport to search for available BrainStem link modules on. See the *transport* enum for supported transports.
- **cbLinkFound** – Process that is called when a module is discovered.
- **vpCRef** – This is passed to cbLinkFound when a module is discovered.

Return values

- **aErrNone** – on success.
- **aErrNotFound** – if no devices were found.

```
static inline bContinueSearch sFindAll (const linkSpec *spec, bool *bSuccess, void
                                         *vpCRef)
```

sFindAll is a callback function which matches any found stem. SFindAll is used by sDiscover(const linkType, list<linkSpec>*) to fill the list provided with any found modules on the specified link type.

Parameters

- **spec** – The linkspec pointer for the device currently being evaluated.
- **bSuccess** – a returned value indicating whether the search has succeeded.
- **vpCRef** – Reference pointer to the std::list that was passed in.

Return values

- **true** – Caller should continue to call this function.
- **false** – Caller should stop calling this function.

```
static inline aErr sDiscover (const linkType type, list<linkSpec> *devices, const uint32_t
                             networkInterface = LOCALHOST_IP_ADDRESS)
```

Discover is called with a specified transport to search for link modules on that transport. The devices list is filled with device specifiers. sDiscover returns aErrNone if the discovery process is successful, regardless of whether any links are found. An error is only returned if the link discovery process fails. Discovery can take some time.

Parameters

- **type** – Transport to search for available BrainStem link modules on. See the *transport* enum for supported transports.
- **devices** – an empty list of specifiers that will be filled in.

Return values

- **aErrNotFound** – if no devices were found.
- **aErrNone** – on success.

```
static bool getStreamPacketType (const aPacket *packet, STREAM_PACKET_t *type)
```

Decodes the streaming packet type from a provided packet.

Parameters

- **packet** – - The packet to be interrogated.

- **type** – - variable to be populated. Filled with kSTREAM_PACKET_UNKNOWN on failure.

Returns

true on success; false on failure.

static bool **isSubindexType** (*STREAM_PACKET_t* type)

Helper function for indicating whether the packet is a subindex type. The subindex can be queried through [Link::getStreamKeyElement](#)

Parameters

type – - The element to evaluate.

Returns

true if the type contains a subindex; false if it does not.

static uint8_t **getStreamKeyElement** (const uint64_t key, *STREAM_KEY_t* element)

Convenience function to unpack a stream key. Note: This function will assert if an out of range STREAM_KEY_t is used.

Parameters

- **key** – The key to be unpacked
- **element** – The element to unpack from the key.

Returns

The requested element from the key.

static bool **isStreamPacket** (const *aPacket* *packet)

Convenience function to determine whether the value is a stream packet. Stream “Packets” encompass all STREAM_PACKET_t valid elements.

Parameters

packet – UEI stream packet to be checked.

Returns

Whether the packet is a stream sample or not

static bool **isStreamSample** (const *aPacket* *packet)

Convenience function to determine whether the value is a stream sample. Stream “Sample” encompasses all STREAM_KEY_t except for kSTREAM_PACKET_BYTES which have a varied structure and depend on the cmd/option/index. Calling isStreamPacket prior is not required as this function will verify the packet type

Parameters

packet – UEI stream packet to be checked.

Returns

Whether the packet is a stream sample or not

static *aErr* **getStreamSample** (const *aPacket* *packet, uint64_t *timestamp = NULL, uint32_t *value = NULL, uint8_t *subindex = NULL)

Convenience function to unpack the stream samples timestamp and value. Calling isStreamSample prior is not required as this function will verify the packet type.

Parameters

- **packet** – UEI stream packet to be unpacked.
- **timestamp** – Variable to be filled with stream sample timestamp. (optional)
- **value** – Variable to be filled with the stream sample (optional). May require casting to signed value depending on the cmd/option code.

Return values

- **aErrPacket** – Not a stream packet
- **aErrUnknown** – Unknown decoding issue.
- **aErrNone** – success.

static void **getTimestampParts** (const uint64_t timestamp, uint32_t *seconds, uint32_t *uSeconds)

Helper function for extracting the parts of a timestamp.

Parameters

- **timestamp** -- Value acquired from [Link::getStreamSample](#)
- **seconds** -- Seconds element from timestamp. Refers to the seconds since firmware boot.
- **uSeconds** -- Micro second element from the timestamp. Refers to the micro seconds from firmware boot. Micro seconds rolls over to seconds. Value range: 0-99999

static bool **linkStreamFilter** (const [aPacket](#) *packet, void *ref)

Filter function for Streaming packets. This is used internally whenever streaming is enabled. Exposed for unit-testing purposes only.

Parameters

- **packet** -- UEI stream packet to be checked/filtered.
- **ref** -- Opaque reference handle

class **impl**

struct **StreamStatusEntry**

[StreamStatusEntry](#) structure - It contains members of streaming entries in the form of key value pairs. Keys are comprised of the devices module address, command, option, index, and subindex API values.

Public Members

uint64_t **key**

The stream key.

uint32_t **value**

The value associated with the key

3.3.28 Module Class

class **Module**

ModuleClass: The [Module](#) class provides a generic interface to a BrainStem hardware module. The [Module](#) class is the parent class for all BrainStem modules. Each module inherits from [Module](#) and implements its hardware specific features.

Subclassed by [a40PinModule](#), [aMTMDAQ1](#), [aMTMDAQ2](#), [aMTMIOSerial](#), [aMTMLoad1](#), [aMTMPM1](#), [aMTMRelay](#), [aMTMStemModule](#), [aPD3M](#), [aUSBCSwitch](#), [aUSBCSwitchPro](#), [aUSBExt3c](#), [aUSBHub2x4](#), [aUSBHub3c](#), [aUSBHub3p](#)

Public Functions

Module (const uint8_t address, const uint8_t model = 0)

Constructor. Implicitly creates a link object with no specifier. Most often objects created with this constructor will use `linkDiscoverAndConnect` to find and connect to a module.

Parameters

- **address** – The BrainStem network address of the module. The default address (or base address for modules that support address offsets) is defined in each module's "Defs.h" header.
- **model** – Acroname model number.

virtual ~**Module** (void)

Destructor.

aErr **connect** (const *linkType* type, const uint32_t serialNum)

Connect using the current link specifier.

Parameters

- **type** – - Transport on which to search for available BrainStem link modules. See the *transport* enum for supported transports.
- **serialNum** – - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Return values

- **aErrBusy** – if the module is already in use.
- **aErrParam** – if the type is incorrect or serialNum is not specified
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr **connectFromSpec** (const *linkSpec* linkSpecifier)

Connect to a link with a fully defined specifier.

Parameters

linkSpecifier – - Connect to module with specifier.

Return values

- **aErrInitialization** – If there is currently no link object.
- **aErrBusy** – If the link is currently connected.
- **aErrParam** – if the specifier is incorrect.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr **discoverAndConnect** (*linkType* type, const uint32_t serialNum = 0)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will

Parameters

- **type** – - Transport on which to search for available BrainStem link modules. See the *transport* enum for supported transports.
- **serialNum** – - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Return values

- **aErrBusy** – if the module is already in use.
- **aErrParam** – if the transport type is undefined.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr **connectThroughLinkModule** (*Module* *pModule)

Connect using link from another *Module*. This member function will connect to the same BrainStem used by given *Module*. If a link module is found on the specified transport, a connection will be made

Parameters

pModule – - Pointer to a valid *Module* class object.

Return values

- **aErrParam** – if the module is undefined.
- **aErrNone** – if the connect was successful.

bool **isConnected** (void)

Is the link connected to the BrainStem *Module*.

linkStatus **getStatus** (void)

Check the status of the BrainStem module connection.

Returns

linkStatus (see aLink.h for status values)

aErr **disconnect** (void)

Disconnect from the BrainStem module.

Return values

- **aErrResource** – If the there is no valid connection.
- **aErrConnection** – If the disconnect failed, due to a communication issue.
- **aErrNone** – If the disconnect was successful.

aErr **reconnect** ()

Reconnect using the current link specifier.

Return values

- **aErrBusy** – if the module is already in use.
- **aErrParam** – if the specifier is incorrect.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

Link ***getLink** (void) const

Get the current link object.

Returns

The link associated with the module.

aErr **getConfig** (aEtherConfig *config)

Gets the links current network configuration

Parameters

config – Variable to be filled with the config

Returns

aErrNone on success. aErrNotReady if the module does not have a link aErrParam if config is NULL

aErr **setConfig** (const aEtherConfig config)

Sets the links network configuration. Configuration must be applied BEFORE connecting

Parameters

config – Configuration to be applied

Returns

aErrNone on success. aErrPermission if the module is currently connected. aErrNotReady if the module does not have a link

uint8_t **getModuleAddress** (void) const

Accessor to get the address of the BrainStem module associated with the instance on

the host machine. (Not to be confused with the System entity which effects the device hardware.)

Returns

The module address.

void **setModuleAddress** (const uint8_t address)

Accessor to set the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

Parameters

address – The module address.

aErr **getLinkSpecifier** (*linkSpec* *spec)

Get linkSpecifier

Parameters

spec – - allocated linkspec struct will be filled with spec.

Returns

aErrNone - If the module does not have a spec.

aErr **getBuild** (uint32_t *build)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Parameters

build – Variable to be filled with build.

aErr **hasUEI** (const uint8_t command, const uint8_t option, const uint8_t index, const uint8_t flags)

Queries the module to determine if it implements a UEI. Each UEI has a command, option or variant, index and flag. The hasUEI method queries for a fully specified UEI. Returns aErrNone if the variation is supported and an appropriate error if not. This call is blocking for up to the nMSTimeout period.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **option** – The option or variant of the command.
- **index** – The entity index.
- **flags** – The flags (ueiOPTION_SET or ueiOPTION_GET).

Return values

- **aErrNone** – The module supports this command and access flags.
- **aErrMode** – The module supports the command but not the access flag.
- **aErrNotFound** – The module does not support the command, option, or index.
- **aErrTimeout** – The request timed out without a response.
- **aErrConnection** – There is no active link

aErr **classQuantity** (const uint8_t command, uint8_t *count)

Queries the module to determine how many entities of the specified class are implemented by the module. Zero is a valid return value. For example, calling classQuantity with the command parameter of cmdANALOG would return the number of analog entities implemented by the module.

Parameters

- **command** – One of UEI commands (cmdXXX).
- **count** – When the request is successful count is updated with the number of entities found.

Return values

- **aErrNone** – Success.

- **aErrTimeout** – The request timed out without a response.
- **aErrConnection** – There is no active link.

aErr subClassQuantity (const uint8_t command, const uint8_t index, uint8_t *count)

Queries the module to determine how many subclass entities of the specified class are implemented by the module for a given entity index. This is used for entities which may be 2-dimensional. E.g. cmdMUX subclasses are the number of channels supported by a particular mux type (index); as a specific example, a module may support 4 UART channels, so subClassQuantity(cmdMUX, aMUX_UART...) could return 4. Zero is a valid return value.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **count** – The number of subclasses found.

Return values

- **aErrNone** – Success.
- **aErrTimeout** – The request timed out waiting for response.
- **aErrConnection** – There is no active link.

aErr entityGroup (const uint8_t command, const uint8_t index, uint8_t *group)

Queries the module the group assigned to an entity and index. Entities groups are used to specify when certain hardware features are fundamentally related. E.g. certain hardware modules may have some digital pins associated with an adjustable voltage rail; these digitals would be in the same group as the rail. Zero is the default group.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **group** – Upon success, group is filled with the entities group value.

Return values

- **aErrNone** – Success.
- **aErrTimeout** – The request timed out without response.
- **aErrConnection** – There is no active link.

aErr debug (const uint8_t *pData, const uint8_t length)

Sends a debug packet to the module containing the provided data. Modules receiving debug packets simply echo the packet back to the sender. If the round-trip is successful, the reply data will match the data sent. This method returns aErrNone when successful, if not successful, an appropriate error is returned.

Parameters

- **pData** – A pointer to an array of data to be sent in the debug packet.
- **length** – The length of the data array.

Return values

- **aErrNone** – Success.
- **aErrTimeout** – Timeout occurred without response.
- **aErrConnection** – No active link exists.

void **setNetworkingMode** (const bool mode)

Sets the networking mode of the module object. By default the module object is configure to automatically adjust its address based on the devices current module address. So that, if the device has a software or hardware offset it will still be able to communication with the device. If advanced networking is required the auto networking mode can be turned off.

Parameters

mode – True/1 for Auto Networking, False/0 for manual networking

3.3.29 PDChannelLogger

class PDChannelLogger

Public Functions

PDChannelLogger (Acroname::BrainStem::Module *module, const uint8_t index, const uint16_t bufferSize = 1024)

Manages BrainStem Power Delivery logging packets.

Parameters

- **module** – Reference to an existing BrainStem *Module*
- **index** – Index/channel logging should be enabled for.
- **bufferLength** – Number of packets the class should queue before dropping.

aErr **setEnabled** (bool enable)

Enables Power Delivery logging.

Parameters

enable – True enables logging; False disables logging

Returns

aErrNone on success

aErr **getPacket** (BS_PD_Packet_t &packet)

Attempts to take a packet from the internal buffer.

Parameters

packet – Reference to a packet to be filled by the function.

Returns

aErrNone if the function successfully acquired any number of packets.

aErrNotReady if no packets were available.

aErrInitialization if this module is not in a valid state to execute.

aErr **getPackets** (std::vector<BS_PD_Packet_t> &packets, const unsigned int maxPackets = 200)

Attempts to take a multiple packets (up to a maximum) from the internal buffer.

Parameters

- **packets** – reference to a vector to be filled by the function
- **maxPackets** – The maximum number of packets to acquire.

Returns

aErrNone if the function successfully acquired any number of packets.

aErrNotReady if no packets were available.

aErrInitialization if this module is not in a valid state to execute.

inline uint8_t **getIndex** (void) const

Provides the index for which this object was created for.

Returns

The index of the object.

3.4 C API Reference

3.4.1 aDefs.h

group **aDefs**

Acroname Specific Universal Defines and includes.

The C-Interface requires some specific defines for cross platform compatibility. The [aDefs.h](#) file contains those defines and includes that are necessary at a global level across platforms.

Things like a cross platform way to specify line endings, safe c-string copy and concatenation operations, and boolean typedefs when they are not defined by default.

We rely on the following std headers:

- assert.h
- stddef.h
- stdint.h
- stdbool.h
- string.h
- stdio.h
- stdlib.h

aSHOWERR (msg, error) (printf("Error in File; %s on line; %d, %s: %d\n", __FILE__, __LINE__, msg, error))

A macro that will emit an error on stdout including file and line number when compiled without NDEBUG flag. When NDEBUG is defined it emits nothing.

OS_NEW_LN "\n"

A macro containing the appropriate line ending characters for a given platform \r\n on windows and \n elsewhere.

aStringCopySafe (d, l, s) strncpy((d), (s), (l))

A macro that maps to platform specific safe string copy. Parameters are;

- d: destination
- l: length
- s: source

aStringCatSafe (d, l, s) strncat((d), (s), (l))

A macro that maps to platform specific safe string concatenation. Parameters are;

- d: destination
- l: length
- s: source

aSNPRINTF snprintf

A macro that maps to the platform specific safe printf output.

aLIBEXPORT __attribute__((visibility ("default")))

A macro that expands for dynamic library linking on a given platform.

aMemPtr void *

An acronym specific semantic define for a pointer to a chunk of memory.

const char ***aDefs_GetModelName** (const int modelNum)

Returns a printable model string.

3.4.2 aDiscovery.h

group **aDiscovery**

Link Discovery Interface.

[aDiscovery.h](#) provides an interface for locating BrainStem modules accross multiple transports. It provides a way to find all modules for a give transport as well as specific modules by serial number, or first found.

enum **linkType**

Enum [linkType](#).

The linkType enum specifies the connection transport type.

Values:

enumerator **INVALID**

- Undefined link type.

enumerator **USB**

- USB link type.

enumerator **TCPIP**

- TCPIP link type.

enumerator **SERIAL**

- Serial link type.

enumerator **AETHER**

- aEther client link type.

enumerator **AETHER_USB**

- aEther USB server link type.

enumerator **AETHER_TCPIP**

- aEther TCPIP server link type.

enumerator **AETHER_SERIAL**

- aEther SERIAL server link type.

struct **linkSpec**

Struct *linkSpec*.

The *linkSpec* contains the necessary information for connecting to a BrainStem module.

Module Specifics

linkType type

The transport type of this spec.

uint32_t **serial_num**

The serial number of the module

uint32_t **module**

The module address

uint32_t **router**

The BrainStem network router address

uint32_t **router_serial_num**

The BrainStem network router serial number

uint32_t **model**

The model type

Transport Specifics

The transport specifics are contained in a union named *t*. The union contains either of two structs *usb* or *ip*.

The USB struct contains a single element:

- **usb_id** - *uint32_t* the *usb_id* of the BrainStem module.

The TCP/IP struct contains two elements:

- **ip_address** - *uint32_t* the IP4 address of the module.
- **ip_port** - *uint32_t* the TCP port for socket connection on the module.

The Serial struct contains two elements:

- **baudrate** - *uint32_t* the serial port baudrate

- **port** - **char*** the serial port path or name

Address this member like `spec.t.usb` or `spec.t.ip`

union *linkSpec*::[anonymous] t
transport union member.

typedef bool **bContinueSearch**

Typedef *bContinueSearch*.

Semantic typedef for continuing the search for modules.

typedef *bContinueSearch* (**aDiscoveryModuleFoundProc**)(const *linkSpec* *spec, bool *bSuccess, void *vpRef)

Typedef *aDiscoveryModuleFoundProc*.

This procedure is the callback to determine whether modules match the ones we are looking for.

- **spec** - *linkSpec* passed into the continueSearch callback.
- **bSuccess** - *bool* Filled with true if a module was found. false otherwise
- **vpRef** - *void** A reference to environment, or other element needed within the callback.
- **bContinueSearch* - Return true to continue, false to stop the search.

uint8_t **aDiscovery_EnumerateModules** (const *linkType* type, *aDiscoveryModuleFoundProc* cbFound, void *vpCRef, const uint32_t networkInterface)

Function *aDiscovery_EnumerateModules*.

Enumerates the discoverable modules for the given link type. Takes a *aDiscoveryModuleFoundProc* which will determine when to stop the enumeration.

Parameters

- **type** – The transport type on which search for devices. Valid *linkType* “linktypes” are accepted
- **cbFound** – The *aDiscoveryModuleFoundProc* to call for each module found.
- **vpCRef** – The vpRef passed into the callback.
- **networkInterface** – Defines the network interface to use when multiple are present. A value of 0 or LOCALHOST_IP_ADDRESS will result in local searches only. Values other than this will have firewall implications.

Returns

Returns the number of modules found.

linkSpec ***aDiscovery_FindModule** (const *linkType* type, const uint32_t serialNum, const uint32_t networkInterface)

Function *aDiscovery_FindModule*.

Finds the module with the given serial number on the given transport type.

Parameters

- **type** – The transport type on which search for devices. Valid *linkType* “linktypes” are accepted

- **serialNum** – The serial number of the Module to find.
- **networkInterface** – Defines the network interface to use when multiple are present. A value of 0 or LOCALHOST_IP_ADDRESS will result in local searches only. Values other than this will have firewall implications.

Returns

A pointer to the *linkSpec* for the requested module if found or NULL otherwise. This call Allocates memory that must be freed by a call to *aLinkSpec_Destroy*.

linkSpec *aDiscovery_FindFirstModule (const *linkType* type, const uint32_t networkInterface)

Function *aDiscovery_FindFirstModule*.

Finds the first module found on the given transport.

Parameters

- **type** – The transport type on which search for devices. Valid *linkType* “linktypes” are accepted
- **networkInterface** – Defines the network interface to use when multiple are present. A value of 0 or LOCALHOST_IP_ADDRESS will result in local searches only. Values other than this will have firewall implications.

Returns

A pointer to the *linkSpec* for the requested module if found or NULL otherwise. This call Allocates memory that must be freed by a call to *aLinkSpec_Destroy*.

LinkSpec Functions

linkSpec *aLinkSpec_Create (const *linkType* type)

Function *aLinkSpec_Create*.

Creates a *linkSpec* object with transport set to the given type.

Parameters

type – The transport type on which search for devices. Valid *linkType* “linktypes” are accepted

Returns

A pointer to the *linkSpec* for the requested module or NULL if there was an error allocating memory. This call Allocates memory that must be freed by a call to *aLinkSpec_Destroy*.

aErr aLinkSpec_Destroy (*linkSpec* **spec)

Function *aLinkSpec_Destroy*.

Destroys and clears the referenced *linkSpec*.

Parameters

spec – A pointer to the *linkSpec* pointer previously allocated.

Returns

aErrNone on success or an error if there was an error encountered deallocating the *linkSpec*.

3.4.3 Error Codes

enum **aErr**

The *aErr* enum lists the possible error codes for library calls. BrainStem commands generally return a set of unified Error codes. The API tries to be consistent and return these errors from every interaction with the stem.

Values:

enumerator **aErrNone**

0 - Success, no error.

enumerator **aErrMemory**

1 - Memory allocation.

enumerator **aErrParam**

2 - Invalid parameter.

enumerator **aErrNotFound**

3 - Not found.

enumerator **aErrFileNameLength**

4 - File name too long.

enumerator **aErrBusy**

5 - Resource busy.

enumerator **aErrIO**

6 - Input/Output error.

enumerator **aErrMode**

7 - Invalid Mode.

enumerator **aErrWrite**

8 - Write error.

enumerator **aErrRead**

9 - Read error.

enumerator **aErrEOF**

10 - End of file.

enumerator **aErrNotReady**

11 - Not ready, no bytes available.

enumerator **aErrPermission**

12 - Insufficient permissions.

- enumerator **aErrRange**
13 - Value out of range.
- enumerator **aErrSize**
14 - Invalid Size.
- enumerator **aErrOverrun**
15 - Buffer/queue overrun.
- enumerator **aErrParse**
16 - Parse error.
- enumerator **aErrConfiguration**
17 - Configuration error.
- enumerator **aErrTimeout**
18 - Timeout occurred.
- enumerator **aErrInitialization**
19 - Initialization error.
- enumerator **aErrVersion**
20 - Invalid version.
- enumerator **aErrUnimplemented**
21 - Functionality unimplemented.
- enumerator **aErrDuplicate**
22 - Duplicate request.
- enumerator **aErrCancel**
23 - Cancellation occurred, or did not complete.
- enumerator **aErrPacket**
24 - Packet byte invalid.
- enumerator **aErrConnection**
25 - Connection error.
- enumerator **aErrIndexRange**
26 - Index out of range.
- enumerator **aErrShortCommand**
27 - BrainStem command too short.

enumerator **aErrInvalidEntity**

28 - Invalid entity error.

enumerator **aErrInvalidOption**

29 - Invalid option code.

enumerator **aErrResource**

30 - Resource unavailable.

enumerator **aErrMedia**

31 - Media error.

enumerator **aErrAsyncReturn**

32 - Asynchronous return.

enumerator **aErrStreamStale**

33 - Stream value is stale.

enumerator **aErrUnknown**

34 - Unknown error.

const char ***aError_GetErrorText** (*aErr* err)

Returns a printable error string.

3.4.4 aFile.h

group **aFile**

Platform Independent File Access Interface.

aFile.h provides a platform independent interface for opening, reading, and writing files.

typedef void ***aFileRef**

Typedef *aFileRef* Opaque reference to a file handle.

enum **aFileMode**

Enum *aFileMode*.

Represents whether the file is to be opened in read or write mode.

Values:

enumerator **aFileModeReadOnly**

File read mode.

enumerator **aFileModeWriteOnly**

File write mode.

enumerator **aFileModeAppend**

File write mode from end of current file.

enumerator **aFileModeUnknown**

File in unknown mode.

enum **aFileSeekMode**

Enum *aFileSeekMode*.

Represents the seek start location.

Values:

enumerator **aSeekStart**

Perform a seek from the beginning of the file.

enumerator **aSeekCurrent**

Perform a seek from the current location.

enumerator **aSeekEnd**

Perform a seek from the end of the file.

bool **aFile_Exists** (const char *pFilename)

Does the File Exist.

Checks for the existence of a file at filename.

Parameters

pFilename – path to file.

Returns

bool True if file exists, false otherwise.

aFileRef **aFile_Open** (const char *pFilename, const *aFileMode* eMode)

Open a File.

Opens the file Given in pFilename with the given fileMode eMode.

Parameters

- **pFilename** – path to file.
- **eMode** – Open the file for Reading or Writing.

Returns

aFileRef on success or NULL on failure.

aErr **aFile_Close** (*aFileRef* *fileRef)

Close an open file.

Close an open file. The fileRef is set to NULL on success.

Parameters

fileRef – Pointer to the handle to the open file.

Return values

- **aErrNone** – Success.

- **aErrParam** – invalid file reference.

Returns

Function returns aErr values.

aErr **aFile_Read** (*aFileRef* fileRef, uint8_t *pBuffer, const size_t nLength, size_t *pActuallyRead)

Read from an open file.

Read from an open file.

Parameters

- **fileRef** – The handle to the open file.
- **pBuffer** – The data buffer to read into.
- **nLength** – The length of the read buffer.
- **pActuallyRead** – The Number of bytes actually read from the file.

Return values

- **aErrNone** – Success.
- **aErrMode** – The file is not readable.
- **aErrIO** – An error occurred reading from the file.
- **aErrEOF** – Read reached the end of the file.

Returns

Function returns aErr values.

aErr **aFile_Write** (*aFileRef* fileRef, const uint8_t *pBuffer, const size_t nLength, size_t *pActuallyWritten)

Write to an open file.

Write to an open file.

Parameters

- **fileRef** – The handle to the open file.
- **pBuffer** – The data to write.
- **nLength** – The length of the data to write.
- **pActuallyWritten** – The Number of bytes actually written to the file.

Return values

- **aErrNone** – Success.
- **aErrMode** – The file is not writable.
- **aErrIO** – An error occurred writing to the file.

Returns

Function returns aErr values.

aErr **aFile_Seek** (*aFileRef* fileRef, const long nOffset, *aFileSeekMode* seekFrom)

Seek within an open file.

Seek within an open file.

Parameters

- **fileRef** – The handle to the open file.
- **nOffset** – The number of bytes to move within the file.

- **seekFrom** – The location to begin the seek from.

Return values

- **aErrNone** – Success.
- **aErrEOF** – Seek would run off the end of the file.
- **aErrRange** – Seek would run off the beginning of the file.
- **aErrIO** – An error occurred moving the file pointer.

Returns

Function returns aErr values.

aErr **aFile_GetSize** (*aFileRef* fileRef, size_t *pulSize)

Get the size of an open file.

Get the size of an open file.

Parameters

- **fileRef** – The handle to the open file.
- **pulSize** – Out param filled with the size of the open file.

Return values

- **aErrNone** – Success.
- **aErrParam** – the fileRef is invalid.
- **aErrIO** – an error occurred calculating the size.

Returns

Function returns aErr values.

aErr **aFile_Delete** (const char *pFilename)

Delete a File.

Deletes the given file pFilename.

Parameters

pFilename – Path to file.

Return values

- **aErrNone** – Success.
- **aErrPermission** – user has insufficient privileges.
- **aErrNotFound** – if the file cannot be located.

Returns

Function returns aErr values.

3.4.5 aLink.h

group **aLink**

BrainStem Link Interface.

[aLink.h](#) provides the interface for creating and maintaining the link to a BrainStem module, and the BrainStem network. It includes facilities for starting and stopping links, as well as sending and receiving BrainStem protocol packets.

typedef uint32_t **aLinkRef**

Typedef [aLinkRef](#) Opaque reference to a BrainStem link.

Typedef for aLinkRef for an opaque reference to BrainStem Link.

enum **linkStatus**

Represents the current state of the BrainStem link.

Values:

enumerator **STOPPED**

Link currently stopped.

enumerator **INITIALIZING**

Starting communication with module.

enumerator **RUNNING**

Link running.

enumerator **STOPPING**

Link is in the process of stopping.

enumerator **SYNCING**

Packet framing lost re-syncing.

enumerator **INVALID_LINK_STREAM**

Link stream provided is not valid.

enumerator **IO_ERROR**

Communication error occurred on link, could not resync.

enumerator **RESETTING**

Resetting the link connection

enumerator **UNKNOWN_ERROR**

Something really bad happened, but we couldn't determine what.

aLinkRef **aLink_CreateUSB** (const uint32_t serialNumber)

Create a USB BrainStem link reference.

Creates a reference to a USB BrainStem link. The linkStream is now maintained by the BrainStem link. If the link already exists, the use count for that link will be incremented and the linkRef for that entry will be returned.

Links created with this procedure must use aLink_Destroy to properly dispose of the link reference and associated connections.

Parameters

serialNumber – Unique identifier of the device.

Returns

aLinkRef identifier if successful or 0 otherwise.

aLinkRef **aLink_CreateTCPIP** (const uint32_t serialNumber, const uint32_t address, const uint16_t port)

Creates a reference to a BrainStem link. The linkStream is now maintained by the BrainStem link. If the link already exists, the use count for that link will be incremented and the linkRef for that entry will be returned.

Links created with this procedure must use aLink_Destroy to properly dispose of the link reference and associated connections.

Parameters

- **serialNumber** – the device serial number
- **address** – the TCPIP address
- **port** – the TCPIP port

Returns

aLinkRef identifier if successful or 0 otherwise.

aErr **aLink_Destroy** (*aLinkRef* *linkRef)

Destroy a BrainStem link reference.

Destroys a Link reference. deallocating associated resources cleanly.

Links created with aLink_Create must use aLink_Destroy to clean up resources used by the link Ref.

Parameters

linkRef – a Pointer to a valid LinkRef. The linkRef will be set to NULL on succesful completion of the Destroy call.

Returns

aStreamRef Return value will always be NULL. The return value has been left for backwards compatability.

aErr **aLink_Reset** (const *aLinkRef* linkRef)

Reset a connection to a BrainStem module.

Stop the active connection to the BrainStem if the Link contains a valid stream Reference, and clear out the communication buffers, and restart the link.

Parameters

linkRef – A valid LinkRef.

Return values

- **aErrNone** – the call completed successfully, a subsequent call to aLink_GetStatus should return the current state of the link.

- **aErrParam** – No valid LinkRef provided.

Returns

Function returns aErr values.

linkStatus **aLink_GetStatus** (const *aLinkRef* linkRef)

Return the current status of the BrainStem link.

Return the current status of the BrainStem link.

Parameters

linkRef – A valid LinkRef.

Returns

linkStatus See the possible linkStatus values.

aPacket ***aLink_GetPacket** (const *aLinkRef* linkRef)

Return the first packet in the Link incoming FIFO.

Return the first packet in the Link incoming FIFO. This call is non blocking, and will return immediately.

Parameters

linkRef – A valid LinkRef.

Returns

aPacket Returns a BrainStem packet on success or NULL.

aPacket ***aLink_AwaitPacket** (const *aLinkRef* linkRef, const unsigned long msTimeout)

Return the first packet in the Link incoming FIFO.

Return the first packet in the Link incoming FIFO. This call blocks waiting for msTimeout milliseconds.

Parameters

- **linkRef** – A valid LinkRef.
- **msTimeout** – The maximum amount of time in milliseconds to wait for a packet.

Returns

aPacket Returns a BrainStem packet on success or NULL.

aPacket ***aLink_GetFirst** (const *aLinkRef* linkRef, aPacketMatchPacketProc proc, const void *vpRef)

Return the first packet matched by proc in the Link incoming FIFO.

Return the first packet matched by proc in the Link incoming FIFO. This call is non blocking and returns immediately.

Parameters

- **linkRef** – A valid LinkRef.
- **proc** – The callback used for determining a matching packet.
- **vpRef** – A resource passed to the callback proc.

Returns

aPacket Returns the first packet that is matched by proc or NULL.

aPacket ***aLink_AwaitFirst** (const *aLinkRef* linkRef, aPacketMatchPacketProc proc, const void *vpRef, const unsigned long msTimeout)

Return the first packet matched by proc in the Link incoming FIFO.

Return the first packet matched by proc in the Link incoming FIFO. This call blocks for up to msTimeout milliseconds waiting for a matching packet.

Parameters

- **linkRef** – A valid LinkRef.
- **proc** – The callback used for determining a matching packet.
- **vpRef** – A resource passed to the callback proc.
- **msTimeout** – The maximum amount of time in milliseconds to wait for a matching packet.

Returns

aPacket Returns the first packet that is matched by proc or NULL.

size_t **aLink_DrainPackets** (const *aLinkRef* linkRef, aPacketMatchPacketProc proc, const void *vpRef)

Drain all matching packets from the incoming FIFO.

Drain all matching packets from the incoming FIFO. This call does not block.

Parameters

- **linkRef** – A valid LinkRef.
- **proc** – The callback used for determining a matching packet.
- **vpRef** – A resource passed to the callback proc.

Returns

aPacket Returns the first packet that is matched by proc or NULL.

aErr **aLink_PutPacket** (const *aLinkRef* linkRef, const *aPacket* *packet)

Put a packet into the outgoingBackend link FIFO.

Put a packet into the outgoingBackend link FIFO.

Parameters

- **linkRef** – A valid LinkRef.
- **packet** – A BrainStem packet.

Return values

- **aErrNone** – Call successfully added the packet.
- **aErrParam** – Invalid LinkRef or packet.
- **aErrResource** – Unable to create memory for packet in FIFO.

Returns

Function returns aErr values.

3.4.6 aMutex.h

group aMutex

Platform Independent Synchronization Primitive.

aMutex.h Provides a platform independent synchronization mechanism. The link interface and the packet fifos both use this interface for synchronization between threads. Includes facilities for creating, locking and unlocking mutex primitives.

typedef void ***aMutexRef**

Typedef *aMutexRef* Opaque pointer to cross platform Mutex.

aMutexRef **aMutex_Create** (const char *name)

Create a Mutex.

Creates a Mutex element and uses the character array as the name of the mutex.

Returns

aMutexRef on success or NULL on failure.

const char ***aMutex_Identifier** (*aMutexRef* mutex)

Mutex Identifier.

Gets the character array that represents the mutex' name.

Returns

A const null terminated character array. This call does not copy the character array, only presents it for use.

aErr **aMutex_Destroy** (*aMutexRef* *mutex)

Mutex Destroy.

Safely destroys a MutexRef, and frees its associated memory. Free should not be called on a MutexRef directly, and all Mutexs created with aMutex_Create must use aMutex_Destroy to free associated resources properly.

Parameters

mutex -- Valid MutexRef

Return values

- **aErrNone** -- If the Destruction was successful.
- **aErrParam** -- If the MutexRef was invalid.

Returns

Function returns aErr values.

aErr **aMutex_Lock** (*aMutexRef* mutex)

Mutex Lock.

Blocking attempt to Lock the mutex. The call will not return until, the requesting thread gains control of the mutex, and successfully locks it or some unrecoverable error occurred.

Return values

- **aErrNone** -- Successfully aquired the lock.
- **aErrParam** -- If the MutexRef was invalid.

Returns

Function returns aErr values.

Returns

aErrDuplicate - If a specific error occurred locking the mutex.

aErr **aMutex_TryLock** (*aMutexRef* mutex)

Mutex TryLock.

Non Blocking attempt to Lock the mutex. The call will return immediately with aErrBusy if another process or thread owns the lock.

Return values

- **aErrNone** – - Successfully aquired the lock.
- **aErrParam** – - If the MutexRef was invalid.
- **aErrBusy** – - If the lock was already in use.

Returns

Function returns aErr values.

aErr **aMutex_Unlock** (*aMutexRef* mutex)

Mutex Unlock.

Relenquish the lock on the mutex.

Return values

- **aErrNone** – - Successfully unlocked mutex.
- **aErrParam** – - If the MutexRef was invalid.
- **aErrPermission** – - If the lock is owned by another thread.

Returns

Function returns aErr values.

3.4.7 aPacket.h

group **aPacket**

BrainStem Packet.

aPacket.h Provides an interface for creating and destroying BrainStem Protocol packets.

const uint16_t **VALIDPACKET**

Const value used to check packet validity.

struct **aPacket**

Struct for BrainStem packets.

the check member is for checking the validity of the packet structure in memory. Current size is used during link stream processing. Address, dataSize and data fulfill the requirements of the BrainStem protocol.

bool **aVALIDPACKET** (const *aPacket* *packet)

Check packet pointer for validity.

Checks to make sure a packet was allocated using aPacket_Create.

Parameters

packet – - valid packet pointer.

Returns

bool - True for valid false otherwise.

aPacket Functions

aPacket *aPacket_Create (void)

Create a BrainStem packet.

Create a BrainStem packet.

Returns

aPacket - Pointer or NULL on error.

aPacket *aPacket_CreateWithData (const uint8_t address, const uint8_t dataLength, const uint8_t *data)

Create a BrainStem packet, containing the given data.

Create a BrainStem packet with data.

Parameters

- **address** -- Module address of the BrainStem module.
- **dataLength** -- The length of the data array.
- **data** -- Pointer to the beginning of the packet data.

Returns

aPacket - Pointer or NULL on error.

aErr aPacket_Reset (*aPacket* *packet)

Reset an existing packet.

Zero out any data the packet contains.

Return values

- **aErrNone** -- If the reset was successful.
- **aErrParam** -- If the packet is not valid.

Returns

Function returns aErr values.

aErr aPacket_AddByte (*aPacket* *packet, const uint8_t byte)

Accumulate a Byte into a packet.

A packet can be constructed byte by byte. the first byte added will be the BrainStem module address, the second byte the data length, and subsequent bytes will be data payload. This call will fail if more than dataLength bytes are added, or if address is an invalid module address (i.e. an odd number).

Return values

- **aErrNone** -- Adding the byte was successful.
- **aErrParam** -- The packet was invalid.
- **aErrPacket** -- The byte added violates the BrainStem protocol.

Returns

Function returns aErr values.

bool aPacket_IsComplete (const *aPacket* *packet)

Determine whether a packet is complete.

A packet can be constructed byte by byte. This call determines whether such a packet has been completed. It checks that dataSize is equal to the currentSize minus the Address and dataSize bytes.

Returns

bool - True if complete false if not complete.

aErr **aPacket_Destroy** (*aPacket* **packet)

Destroy a BrainStem packet.

Safely destroy a brainstem packet and deallocate the associated resources.

Parameters

packet -- A pointer to a pointer of a valid packet. The packet pointer will be set to NULL on successful destruction of the packet.

Return values

- **aErrNone** -- The packet was successfully destroyed.
- **aErrParam** -- The packetRef is invalid.

Returns

Function returns aErr values.

3.4.8 aProtocoldefs.h

group **aProtocoldefs**

BrainStem Protocol Definitions.

aProtocoldefs.h Provides protocol and BrainStem specific defines for entities, communication, and protocol specifics.

aBRAINSTEM_MAXPACKETBYTES 28

8 Bytes - Packet protocol payload maximum.

group **UEI_Defines**

UEI and Command support for C/C++ and Reflex languages.

Defines

ueiSPECIFIER_INDEX_MASK 0x1F

0x1F - Mask bits for Index on index byte.

ueiSPECIFIER_RETURN_MASK 0xE0

0xE0 - Mask bits for Return value on index byte.

ueiSPECIFIER_RETURN_HOST 0x20

1 << 5 - Specifier Bit for UEI response to host.

ueiSPECIFIER_RETURN_I2C 0x40

2 << 5 - Specifier Bit for UEI response to Module over I2C.

ueiSPECIFIER_RETURN_VM 0x60

3 << 5 - Specifier Bit for UEI response to VM on module.

ueiREPLY_ERROR 0x80

1 << 7 - Error flag on response in index byte.

ueiREPLY_STREAM 0x40

1 << 6 - Stream flag on response in index byte.

ueiOPTION_GET 0x40

0x40 - Option byte code for UEI Get request.

ueiOPTION_VAL 0x00

0x00 - Option byte code for UEI Val response.

ueiOPTION_SET 0x80

0x80 - Option byte code for UEI Set request.

ueiOPTION_ACK 0xC0

0xC0 - Option byte code for UEI Ack response.

ueiOPTION_MASK 0x3F

0x3F - Mask for getting command option from option byte.

ueiOPTION_OP_MASK 0xC0

0xC0 - Mask for getting Operation Get/Set/Val/Ack

ueiBYTES_CONTINUE 0x80

ueiBYTES_CONTINUE_MASK 0x7F

Analog Entity

group **cmdANALOG_Defines**

Analog Command defines.

Defines

cmdANALOG 30

30 - Analog command code.

group **cmdANALOG_Command_Options**

Defines

analogConfiguration 1
1 - Analog Configuration Option Code

analogConfigurationInput 0
0 - Input configuration

analogConfigurationOutput 1
1 - Output configuration

analogConfigurationHiZ 2
2 - High Impedance configuration

analogValue 2
2 - Analog Value Option Code

analogVoltage 3
3 - Analog Voltage Option Code

analogBulkCaptureSampleRate 4
4 - Analog Bulk Capture Sample Rate Option Code

analog_Hz_Minimum 7000
7000 - Minimum hertz sample rate

analog_Hz_Maximum 200000
200000 - Maximum hertz sample rate

analogBulkCaptureNumberOfSamples 5
5 - Bulk Capture number of samples Option Code

analogBulkCapture 6
6 - Bulk Capture Option Code

analogBulkCaptureState 7
7 - Bulk Capture State Option Code

bulkCaptureIdle 0
0 - Idle state

bulkCapturePending 1
1 - Pending state

bulkCaptureFinished 2
2 - Finished state

bulkCaptureError 3

3 - Error state

analogRange 8

8 - Analog Range Option Code

analogRange_P0V064N0V064 0

0 - +/- 64mV range

analogRange_P0V64N0V64 1

1 - +/- 640mV range

analogRange_P0V128N0V128 2

2 - +/- 128mV range

analogRange_P1V28N1V28 3

3 - +/- 1.28V range

analogRange_P1V28N0V0 4

4 - 0-1.28V range

analogRange_P0V256N0V256 5

5 - +/- 256mV range

analogRange_P2V56N2V56 6

6 - +/- 2.56V range

analogRange_P2V56N0V0 7

7 - 0-2.56V range

analogRange_P0V512N0V512 8

8 - +/- 512mV range

analogRange_P5V12N5V12 9

9 - +/- 5.12V range

analogRange_P5V12N0V0 10

10 - 0-5.12V range

analogRange_P1V024N1V024 11

11 - +/- 1.024V range

analogRange_P10V24N10V24 12

12 - +/- 10.24V range

analogRange_P10V24N0V0 13

13 - 0-10.24V range

analogRange_P2V048N0V0 14

14 - 0-2.048V range

analogRange_P4V096N0V0 15

15 - 0-4.096V range

analogEnable 9

9 - Analog Enable Option Code

analogNumberOfOptions 10

10 - Number of Options for analog, always last entry

App Entity

group **cmdAPP_Defines**

App Command defines.

Defines

cmdAPP 5

5 - App command code.

group **cmdAPP_Command_Options**

Defines

appExecute 1

1 - Execute Option Code

appReturn 2

2 - Return Option Code

Capacity Entity

group **cmdCAPACITY_Defines**

Capacity Command defines.

Defines

cmdCAPACITY 73

73 - Capacity command code.

group **cmdCAPACITY_Command_Options**

Defines

capacityUEI 1

1 - UEI Option Code

capacitySubClassSize 3

3 - SubClass size Option Code

capacityClassQuantity 4

4 - Class Quantity Option Code

capacitySubClassQuantity 5

5 - SubClass Quantity Option Code

capacityEntityGroup 6

6 - Entity Group Option Code

capacityBuild 255

255 - Build Option Code

Clock Entity

group **cmdCLOCK_Defines**

Clock Command defines.

Defines

`cmdCLOCK 83`

83 - Clock command code.

group `cmdCLOCK_Command_Options`

Defines

`clockYear 1`

1 - Year Option Code

`clockMonth 2`

2 - Month Option Code

`clockDay 3`

3 - Day Option Code

`clockHour 4`

4 - Hour Option Code

`clockMinute 5`

5 - Minute Option Code

`clockSecond 6`

6 - Second Option Code

`clockNumberOfOptions 7`

7 - Number of Options for clock, always last entry

Digital Entity

group `cmdDIGITAL_Defines`

Digital Command defines.

Defines

`cmdDIGITAL 31`

31 - Digital command code.

group `cmdDIGITAL_Command_Options`

Defines

digitalConfiguration 1

1 - Digital configuration Option Code

digitalConfigurationInput 0x00

0x00 - Input Configuration

digitalConfigurationOutput 0x01

0x01 - Output Configuration

digitalConfigurationRCServoInput 0x02

0x02 - RCServo Input Configuration

digitalConfigurationRCServoOutput 0x03

0x03 - RCServo Output Configuration

digitalConfigurationHiZ 0x04

0x04 - High Impedance Configuration

digitalConfigurationInputPullUp 0x00

0x00 - Input Configuration with pull-up

digitalConfigurationInputNoPull 0x04

0x04 - Input Configuration without pull-up/pull-down

digitalConfigurationInputPullDown 0x05

0x05 - Input Configuration with pull-down

digitalConfigurationSignalOutput 0x06

0x06 - Signal Output Configuration

digitalConfigurationSignalInput 0x07

0x07 - Signal Input Configuration

digitalConfigurationSignalCounterInput 0x08

0x08 - Signal Input Counter Configuration

digitalConfigurationLinkInput 0x09

0x09 - Input from a linked digital channel

digitalConfigurationLinkOutput 0x0A

0x0A - Output to a linked digital channel

digitalState 2

2 - State Option Code

digitalStateAll 3

3 - Full State Option Code

digitalTimeDelay 4

4 - Time Delay Option Code

digitalValue 5

5 - Value to set the pin to

digitalLinkChannel 6

6 - Link channel for linking digital entities

digitalNumberOfOptions 7

7 - Number of Options for digital, always last entry

Equalizer Entity

group **cmdEQUALIZER_Defines**

Equalizer Command defines.

Defines

cmdEQUALIZER 15

15 - Equalizer command code.

group **cmdEQUALIZER_Command_Options**

Defines

equalizerReceiverConfig 1

1 - Receiver Configuration Option Code

equalizerTransmitterConfig 2

2 - Transmitter (Driver) Configuration Option Code

equalizerManualConfig 3

3 - Manual Configuration Option Code

equalizerNumberOfOptions 4

4 - Number of Options for equalizer, always last entry

group **cmdEQUALIZER_Command_Constants**

Defines

equalizer2p0 0
Equalizer Index for USB2.0.

equalizer3p0 1
Equalizer Index for USB3.0.

Ethernet Entity

group **cmdETHERNET_Defines**
Ethernet Command defines.

Defines

cmdETHERNET 39
39 - Ethernet command code.

group **cmdETHERNET_Command_Options**

Defines

ethernetEnabled 1
1 - Enabled

ethernetNetworkConfiguration 2
2 - Network IP Assignment Configuration

ethernetConfigurationNone 0
0 - Disable IP Assignment Option Code

ethernetConfigurationStatic 1
1 - Static IP Address Option Code

ethernetConfigurationDhcp 2
2 - DHCP Address Option Code

ethernetStaticIPv4Address 3
3 - Static IP Address

ethernetStaticIPv4Netmask 4
4 - Static IP Netmask

ethernetStaticIPv4Gateway 5
5 - Static IP Gateway

ethernetIPv4Address 6
6 - Current IP Address

ethernetIPv4Netmask 7
7 - Current IP Netmask

ethernetIPv4Gateway 8
8 - Current IP Gateway

ethernetStaticIPv4DNSAddress 9
9 - Static DNS Address

ethernetIPv4DNSAddress 10
10 - Current DNS Address

ethernetHostname 11
11 - Hostname

ethernetMACAddress 12
12 - MAC Address

ethernetInterfacePort 13
13 - Port number of TCPIP Interface

ethernetInterfacePort_RestServer_HTTP 0
0 - REST Service (HTTP) Port

ethernetInterfacePort_RestServer_HTTPS 1
1 - REST Service (HTTP) Port

ethernetInterfacePort_BrainStem_TCP 2
2 - BrainStem TCPIP Transport Port

ethernetInterfacePort_BrainStem_DiscoveryRequest 3
3 - BrainStem Multicast Discovery Request Port

ethernetInterfacePort_BrainStem_DiscoveryReply 4
4 - BrainStem Multicast Discovery Reply Port

ethernetInterfacePort_Max 5
5 - Number of options for ethernetInterfacePort

ethernetNumberOfOptions 14

14 - Number of Options for ethernet, always last entry

General Definitions

group **General_Defines**

Defines

cmdDEBUG 23

Debug Command.

cmdNOTIFY 24

Notify Command.

cmdLAST 96

Maximum Command.

HDBaseT Entity

group **cmdHDBASET_Defines**

HDBaseT Command defines.

Defines

cmdHDBASET 41

41 - HDBaseT command code.

group **cmdHDBASET_Command_Options**

Defines

hdbasetSerialNumber 1

1 - Serial Number Option Code

hdbasetFirmwareVersion 2

2 - Firmware Version Option Code

hdbasetState 3

3 - State Option Code

hdbasetState_devicePresent_Bit 0

0 - Device Present Bit

hdbasetState_linkUp_Bit 1

1 - Link Up Bit

hdbasetState_linkRole_Offset 2

2 - Link Role Offset

hdbasetState_linkRole_Mask 0x3

0x3 - Link Role Mask

hdbasetState_linkRole_Unknown 0

0 - Unknown Link Role

hdbasetState_linkRole_Leader 1

1 - Leader/Master Link Role

hdbasetState_linkRole_Follower 2

2 - Follower/Slave Link Role

hdbasetCableLength 4

4 - Cable Length Option Code

hdbasetMSEA 5

5 - Mean Squared Error Pair A Option Code

hdbasetMSEB 6

6 - Mean Squared Error Pair B Option Code

hdbasetRetransmissionRate 7

7 - Retransmission Rate Option Code

hdbasetLinkUtilization 8

8 - Link Utilization Option Code

hdbasetEncodingState 9

9 - HDBaseT Encoding State Option Code

hdbasetEncodingState_Unknown 0

0 - Unknown

hdbasetEncodingState_PAM16 1

1 - PAM16

hdbasetEncodingState_PAM8 2

2 - PAM8

hdbasetEncodingState_PAM4 3

3 - PAM4

hdbasetUSB2DeviceTree 10

10 - USB2 Device Tree Option Code

hdbasetUSB3DeviceTree 11

11 - USB3 Device Tree Option Code

hdbasetLinkRole 12

12 - Link Role Option Code

hdbasetLinkRole_AutoDetect 0

0 - Auto-Detected Link Role

hdbasetLinkRole_Leader 1

1 - Leader/Master Link Role

hdbasetLinkRole_Follower 2

2 - Follower/Slave Link Role

hdbasetNumberOfOptions 13

13 - Number of Options for HDBaseT, always last entry

Heartbeat Entity

group **cmdHB_Defines**

Heartbeat Command defines.

Defines

cmdHB 0

0 - Heartbeat command code.

group **cmdHB_Command_Constants**

Defines

`val_HB_S2H_UP` 0

`val_HB_S2H_DOWN` 1

`val_HB_H2S_UP` 2

`val_HB_H2S_DOWN` 3

`val_HB_M2R_UP` 4

`val_HB_M2R_DOWN` 5

I2C Entity

group `cmdI2C_Defines`
I2C Command defines.

Defines

`cmdI2C` 8
8 - I2C command code.

group `cmdI2C_Command_Options`

Defines

`i2cSetPullup` 1
1 - Set Pullup Option Code

group `cmdI2C_Command_Constants`

Defines

`i2cDefaultSpeed` 0
Default I2C Speed Setting.

`i2cSpeed_100Khz` 1
100kHz I2C Speed Setting

i2cSpeed_400Khz 2
400kHz I2C Speed Setting

i2cSpeed_1000Khz 3
1000kHz I2C Speed Setting

bitI2CError 0x80
Bit I2C error code.

bitI2CAck 0x40
Bit I2C ack code.

Mux Entity

group **cmdMUX_Defines**
Mux Command defines.

Defines

cmdMUX 6
6 - Mux command code.

group **cmdMUX_Command_Options**

Defines

muxEnable 1
1 - Channel Enable Option Code

muxChannel 2
2 - Channel Select Option Code

muxVoltage 3
3 - Channel Voltage Measurement Option Code

muxConfig 4
4 - Mux Mode Configuration Option Code

muxConfig_default 0
0 - Default (Manual)

muxConfig_splitMode 1
1 - Split Mode

`muxConfig_channelpriority` 2

2 - Channel Priority

`muxSplit` 5

5 - Split Mode Configuration Option Code

`muxNumberOfOptions` 6

6 - Number of Options for mux, always last entry

POE Entity

group `cmdPOE_Defines`

POE Command defines.

Defines

`cmdPOE` 40

40 - POE command code.

group `cmdPOE_Command_Options`

Defines

`poePowerMode` 1

1 - Power Mode Option Code

`poePowerMode_Off` 0

`poePowerMode_PSE` 1

`poePowerMode_PD` 2

`poePowerMode_Auto` 3

`poePowerState` 2

2 - Power State Option Code

`poePowerState_Off` 0

`poePowerState_PSE` 1

poePowerState_PD 2

poePairEnabled 3

3 - Pair Enabled Option Code

poePairSourcingClass 4

4 - Pair Sourcing Class Option Code

poePairSourcingClass_Unknown 0

poePairSourcingClass_0 1

poePairSourcingClass_1 2

poePairSourcingClass_2 3

poePairSourcingClass_3 4

poePairSourcingClass_4 5

poePairSourcingClass_4_PlusType1Limited 6

poePairSourcingClass_5_4PairSingleSignature 7

poePairSourcingClass_5_4PairDualSignature 8

poePairSourcingClass_6_4PairSingleSignature 9

poePairSourcingClass_7_4PairSingleSignature 10

poePairSourcingClass_8_4PairSingleSignature 11

poePairSourcingClass_Mismatch 12

poePairSourcingClass_OverCurrent 13

poePairRequestedClass 5

5 - Pair Requested Class Option Code

poePairRequestedClass_Unknown 0

poePairRequestedClass_0 1

poePairRequestedClass_1 2

poePairRequestedClass_2 3

poePairRequestedClass_3 4

poePairRequestedClass_4 5

poePairRequestedClass_4_PlusType1Limited 6

poePairRequestedClass_5_4PairSingleSignature 7

poePairRequestedClass_5_4PairDualSignature 8

poePairRequestedClass_6_4PairSingleSignature 9

poePairRequestedClass_7_4PairSingleSignature 10

poePairRequestedClass_8_4PairSingleSignature 11

poePairRequestedClass_Mismatch 12

poePairRequestedClass_OverCurrent 13

poePairDiscoveredClass 6

 6 - Pair Discovered Class Option Code

poePairDiscoveredClass_Unknown 0

poePairDiscoveredClass_0 1

poePairDiscoveredClass_1 2

poePairDiscoveredClass_2 3

poePairDiscoveredClass_3 4

poePairDiscoveredClass_4 5

poePairDiscoveredClass_4_PlusType1Limited 6

poePairDiscoveredClass_5_4PairSingleSignature 7

poePairDiscoveredClass_5_4PairDualSignature 8

poePairDiscoveredClass_6_4PairSingleSignature 9

poePairDiscoveredClass_7_4PairSingleSignature 10

poePairDiscoveredClass_8_4PairSingleSignature 11

poePairDiscoveredClass_Mismatch 12

poePairDiscoveredClass_OverCurrent 13

poePairDetectionStatus 7
7 - Pair Detection Status Option Code

poePairDetectionStatus_Unknown 0

poePairDetectionStatus_Short_Circuit 1

poePairDetectionStatus_Open_Circuit 2

poePairDetectionStatus_Low_Resistance 3

poePairDetectionStatus_High_Resistance 4

poePairDetectionStatus_Valid 5

poePairDetectionStatus_Switch_Failure 6

poePairVoltage 8
8 - Pair Voltage Option Code

poePairCurrent 9
9 - Pair Current Option Code

poePairPower 10
10 - Pair Power Option Code

poeTotalPower 11
11 - Combined Pair Power Option Code

poePairAccumulatedPower 12
12 - Pair Accumulated Power Option Code

poeTotalAccumulatedPower 13
13 - Combined Pair Accumulated Power Option Code

poePairResistance 14
14 - Pair Resistance Option Code

poePairCapacitance 15
15 - Pair Capacitance Option Code

poeNumberOfOptions 16
16 - Number of Options for POE, always last entry

Pointer Entity

group **cmdPOINTER_Defines**
Pointer Command defines.

Defines

cmdPOINTER 7
7 - Pointer command code.

group **cmdPOINTER_Command_Options**

Defines

pointerOffset 1
1 - Offset Option Code

pointerMode 2
2 - Mode Option Code

pointerModeStatic 0
0 - Static Mode

pointerModeIncrement 1
1 - Increment Mode

pointerModeDefault 0
0 - Default Mode

pointerTransferStore 3
3 - Set Transfer Store Option Code

pointerChar 4

4 - Char Option Code

pointerShort 5

5 - Short Option Code

pointerInt 6

6 - Integer Option Code

pointerTransferToStore 7

7 - Transfer to Store Option Code

pointerTransferFromStore 8

8 - Transfer From Store Option Code

Port Entity

group **cmdPORT_Defines**

Port Command defines.

Defines

cmdPORT 37

37 - Port command code.

group **cmdPORT_Command_Options**

Defines

portVbusVoltage 1

1 - Vbus Voltage Option Code

portVbusCurrent 2

2 - Vbus Current Option Code

portVconnVoltage 3

3 - Vconn Voltage Option Code

portVconnCurrent 4

4 - Vconn Current Option Code

portPortEnabled 5

5 - Port Enabled Option Code

portPowerEnabled 6
6 - Port Power Enabled Option Code

portDataEnabled 7
7 - Port Data Enabled Option Code

portDataHSEnabled 8
8 - Port HS Data Enabled Option Code

portDataHS1Enabled 9
9 - Port HS1 Data Enabled Option Code

portDataHS2Enabled 10
10 - Port HS2 Data Enabled Option Code

portDataSSEnabled 11
11 - Port SS Data Enabled Option Code

portDataSS1Enabled 12
12 - Port SS1 Data Enabled Option Code

portDataSS2Enabled 13
13 - Port SS2 Data Enabled Option Code

portCCEnabled 14
14 - CC Enabled Option Code

portCC1Enabled 15
15 - CC1 Enabled Option Code

portCC2Enabled 16
16 - CC2 Enabled Option Code

portVconnEnabled 17
17 - Vconn Enabled Option Code

portVconn1Enabled 18
18 - Vconn1 Enabled Option Code

portVconn2Enabled 19
19 - Vconn2 Enabled Option Code

portPortState 20
20 - Port State Option Code

`portPortState_powerEnabled_Bit 0`

0 - Power enable bit

`portPortState_HS1Enabled_Bit 1`

1 - USB 2.0 (HS) 1 side enable bit.

`portPortState_HS2Enabled_Bit 2`

2 - USB 2.0 (HS) 2 side enable bit

`portPortState_SS1Enabled_Bit 3`

3 - USB 3.0 (SS) 1 side enable bit

`portPortState_SS2Enabled_Bit 4`

4 - USB 3.0 (SS) 2 side enable bit

`portPortState_CC1Enabled_Bit 5`

5 - CC 1 enable bit

`portPortState_CC2Enabled_Bit 6`

6 - CC 2 enable bit

`portPortState_Vconn1Enabled_Bit 7`

7 - VConn 1 enable bit

`portPortState_Vconn2Enabled_Bit 8`

8 - VConn 2 enable bit

`portPortState_SBU1Enabled_Bit 9`

9 - SBU 1 enable bit

`portPortState_SBU2Enabled_Bit 10`

10 - SBU 2 enable bit

`portPortState_CC_Flipped_Bit 11`

11 - CC Flipped bit

`portPortState_HS_Flipped_Bit 12`

12 - HS Flipped bit

`portPortState_SS_Flipped_Bit 13`

13 - SS Flipped bit

`portPortState_SBU_Flipped_Bit 14`

14 - SBU Flipped bit

portPortState_KACEnabled_Bit 15

15 - KAC Enabled bit

portErrors 21

21 - Port Errors Option Code

portCurrentLimit 22

22 - Port Current Limit. Option Code

portCurrentLimitMode 23

23 - Port Current Limit Mode. Option Code

portPowerLimit 24

24 - Port Power Limit. Option Code

portPowerLimitMode 25

25 - Port Power Limit Mode. Option Code

portAvailablePower 26

26 - Port available power Option Code

portName 27

27 - Port Name Option Code

portCCCurrentLimit 28

28 - Port CC Current Limit Option Code

portCCCurrentLimit_None 0

0 - None Current value

portCCCurrentLimit_Default 1

1 - Default Current value (500/900mA)

portCCCurrentLimit_1p5 2

2 - 1.5 Amp Current value

portCCCurrentLimit_3p0 3

3 - 3.0 Amp Current value

portPowerMode 30

30 - Port Power mode Option Code

portPowerMode_none_Value 0

0 - None/Disabled

portPowerMode_sdp_Value 1

1 - Standard Downstream Port (SDP)

portPowerMode_cdp_dcp_Value 2

2 - Charging Downstream Port (CDP) or Dedicated Charging Port (DCP)

portPowerMode_qc_Value 3

3 - Qualcomm Quick Charge (QC)

portPowerMode_pd_Value 4

4 - Power Delivery (PD)

portPowerMode_ps_Value 5

5 - Power Supply Mode

portPowerMode_usbc_Value 6

6 - USB-C Mode

portDataRole 31

31 - Port Data Role Option Code

portDataRole_Disabled_Value 0

0 - Role - Disabled

portDataRole_Upstream_Value 1

1 - Role - Upstream Port

portDataRole_Downstream_Value 2

2 - Role:Downstream Port

portDataRole_Control_Value 3

3 - Role:Control Port

portDataSpeed 32

32 - Port Data Speed Option Code

portDataSpeed_ls_1p5M_Bit 0

0 - Speed - Low Speed (1.5Mbps) bit indicator

portDataSpeed_fs_12M_Bit 1

1 - Speed - Full Speed (12Mbps) bit indicator

portDataSpeed_hs_480M_Bit 2

2 - Speed - High Speed (480Mbps) bit indicator

portDataSpeed_ss_5G_Bit 3
3 - Speed - Super Speed (5Gbps) bit indicator

portDataSpeed_ss_10G_Bit 4
4 - Speed - Super Speed Plus (10Gbps) bit indicator

portDataSpeed_Connected_2p0_Bit 6
6 - USB 2.0 Connected

portDataSpeed_Connected_3p0_Bit 7
7 - USB 3.0 Connected

portPortMode 33
33 - Port Mode Option Code

portPortMode_powerEnabled_Bit 0
0 - Power enable bit

portPortMode_HS1Enabled_Bit 1
1 - USB 2.0 (HS) 1 side enable bit.

portPortMode_HS2Enabled_Bit 2
2 - USB 2.0 (HS) 2 side enable bit

portPortMode_SS1Enabled_Bit 3
3 - USB 3.0 (SS) 1 side enable bit

portPortMode_SS2Enabled_Bit 4
4 - USB 3.0 (SS) 2 side enable bit

portPortMode_CC1Enabled_Bit 5
5 - CC 1 enable bit

portPortMode_CC2Enabled_Bit 6
6 - CC 2 enable bit

portPortMode_Vconn1Enabled_Bit 7
7 - VCONN 1 enable bit

portPortMode_Vconn2Enabled_Bit 8
8 - VCONN 2 enable bit

portPortMode_SBU1Enabled_Bit 9
9 - SBU 1 enable bit

`portPortMode_SBU2Enabled_Bit` 10

10 - SBU 2 enable bit

`portPortMode_HSFlipEnabled_Bit` 11

11 - HS Flipped bit

`portPortMode_SSFlipEnabled_Bit` 12

12 - SS Flipped bit

`portPortMode_CCFlipEnabled_Bit` 13

13 - CC Flipped bit

`portPortMode_SBUFlipEnabled_Bit` 14

14 - SBU Flipped bit

`portPortMode_KACEnabled_Bit` 15

15 - KAC Enabled bit

`portPortMode_portPowerMode_Offset` 16

16 - Port Power Mode offset within Port Mode

`portPortMode_portPowerMode_Mask` 0x7

0x7 - Port Power Mode offset (Pre offset shift)

`portPortMode_portPowerMode_none_Value` 0

0 - Port Power None/Disabled

`portPortMode_portPowerMode_sdp_Value` 1

1 - Port Power Standard Downstream Port (SDP)

`portPortMode_cdp_dcp_Value` 2

2 - Charging Downstream Port (CDP) or Dedicated Charging Port (DCP)

`portPortMode_portPowerMode_qc_Value` 3

3 - Port Power Qualcomm Quick Charge (QC)

`portPortMode_portPowerMode_pd_Value` 4

4 - Port Power Power Delivery (PD)

`portPortMode_portPowerMode_ps_Value` 5

5 - Port Power Power Supply Mode

`portPortMode_portPowerMode_usbc_Value` 6

6 - Port Power USB-C Mode

portVoltageSetpoint 34

34 - Port Voltage Setpoint for VBUS Override Option Code

portAllocatedPower 35

35 - Port Allocated Power Option Code

portDataHSRoutingBehavior 36

36 - Port Change HighSpeed Data Signal Routing Behavior Option Code

portDataHSRoutingBehavior_FollowCC 0

0 - Auto Follow CC

portDataHSRoutingBehavior_Side1 1

1 - Side 1 Only

portDataHSRoutingBehavior_Side2 2

2 - Side 2 Only

portDataHSRoutingBehavior_Shorted 3

3 - Side 1 and 2 Shorted

portDataSSRoutingBehavior 37

37 - Port Change SuperSpeed Data Signal Routing Behavior Option Code

portDataSSRoutingBehavior_FollowCC 0

0 - Auto Follow CC

portDataSSRoutingBehavior_Side1 1

1 - Side 1 Only

portDataSSRoutingBehavior_Side2 2

2 - Side 2 Only

portVbusAccumulatedPower 38

38 - Vbus Accumulated Power Option Code

portResetVbusAccumulatedPower 39

39 - Reset Vbus Accumulated power Option Code

portVconnAccumulatedPower 40

40 - Vconn Accumulated Power Option Code

portResetVconnAccumulatedPower 41

41 - Reset Vconn Accumulated power Option Code

portHSBoost 42

42 - Port USB 2.0 High Speed Boost Settings Option Code

portHSBoost_m5Percent 0

0 - -5%

portHSBoost_Nominal 1

1 - Nominal

portHSBoost_p5Percent 2

2 - +5%

portHSBoost_p10Percent 3

3 - +10%

portHSBoost_p15Percent 4

4 - +15%

portHSBoost_p20Percent 5

5 - +20%

portHSBoost_p25Percent 6

6 - +25%

portHSBoost_p30Percent 7

7 - +30%

portResetEntityToFactoryDefaults 44

44 - Port Reset to Factory Defaults Option Code

portCC1State 45

45 - Port CC1 Bias Option Code

portCC1State_None 0

0 - None value

portCC1State_Invalid 1

1 - Invalid value

portCC1State_RpDefault 2

2 - Default (500/900/1500mA) Rp value

portCC1State_Rp1p5 3

3 - 1.5 Amp Rp value

`portCC1State_Rp3p0` 4

4 - 3.0 Amp Rp value

`portCC1State_Rd` 5

5 - Rd value

`portCC1State_Ra` 6

6 - Ra value

`portCC1State_Managed` 7

7 - Managed by FW

`portCC1State_Unknown` 8

8 - Unknown

`portCC2State` 46

46 - Port CC2 State Option Code

`portCC2State_None` 0

0 - None value

`portCC2State_Invalid` 1

1 - Invalid value

`portCC2State_RpDefault` 2

2 - Default (500/900/1500mA) Rp value

`portCC2State_Rp1p5` 3

3 - 1.5 Amp Rp value

`portCC2State_Rp3p0` 4

4 - 3.0 Amp Rp value

`portCC2State_Rd` 5

5 - Rd value

`portCC2State_Ra` 6

6 - Ra value

`portCC2State_Managed` 7

7 - Managed by FW

`portCC2State_Unknown` 8

8 - Unknown

portCC1Voltage 47
47 - CC1 get voltage Option Code

portCC1Current 48
48 - CC1 get current Option Code

portCC2Voltage 49
49 - CC2 get voltage Option Code

portCC2Current 50
50 - CC2 get current Option Code

portSBU1Voltage 51
51 - SBU1 get voltage Option Code

portSBU2Voltage 52
52 - SBU2 get voltage Option Code

portCC1AccumulatedPower 53
53 - CC1 Accumulated Power Option Code

portCC2AccumulatedPower 54
54 - CC2 Accumulated Power Option Code

portNumberOfOptions 55
55 - Number of Options for port, always last entry

Power Delivery Entity

group **cmdPOWERDELIVERY_Defines**
Power Delivery Command defines.

Defines

cmdPOWERDELIVERY 36
36 - Power Delivery command code.

group **cmdPOWERDELIVERY_Command_Options**

Defines

powerdeliveryPowerDataObject 1

powerdeliveryPowerDataObjectList 2

powerdeliveryPowerDataObjectEnabled 3

powerdeliveryPowerDataObjectEnabledList 4

powerdeliveryNumberOfPowerDataObjects 5

powerdeliveryRequestDataObject 6

powerdeliveryConnectionState 7

pdConnectionState_None 0

pdConnectionState_Source 1

pdConnectionState_Sink 2

pdConnectionState_PoweredCable 3

pdConnectionState_PoweredCableWithSink 4

pdConnectionState_AudioAccessory 5

pdConnectionState_DebugAccessory 6

powerdeliveryLinkState 8

pdLinkState_linkType_Offset 0

pdLinkState_linkType_Mask 0x7

pdLinkState_linkType_None 0

pdLinkState_linkType_Legacy 1

pdLinkState_linkType_SPR 2

pdLinkState_linkType_EPR 3

pdLinkState_connectionState_Offset 4

pdLinkState_connectionState_Mask 0xF

pdLinkState_connectionState_None 0

pdLinkState_connectionState_Source 1

pdLinkState_connectionState_Sink 2

pdLinkState_connectionState_PoweredCable 3

pdLinkState_connectionState_PoweredCableWithSink 4

pdLinkState_connectionState_AudioAccessory 5

pdLinkState_connectionState_DebugAccessory 6

pdLinkState_powerRole_Offset 8

pdLinkState_powerRole_Mask 0x7

pdLinkState_powerRole_None 0

pdLinkState_powerRole_Source 1

pdLinkState_powerRole_Sink 2

pdLinkState_dataRole_Offset 12

pdLinkState_dataRole_Mask 0x7

pdLinkState_dataRole_None 0

pdLinkState_dataRole_DFP 1

pdLinkState_dataRole_UFP 2

pdLinkState_vconnState_Offset 16

pdLinkState_vconnState_Mask 0x7

pdLinkState_vconnState_Off 0

pdLinkState_vconnState_Source 1

pdLinkState_vconnState_NotSource 2

pdLinkState_specRevision_Offset 20

pdLinkState_specRevision_Mask 0x7

pdLinkState_specRevision_Unknown 0

pdLinkState_specRevision_1 1

pdLinkState_specRevision_2 2

pdLinkState_specRevision_3 3

powerdeliveryAttachTimeElapsed 9

powerdeliveryResetPowerDataObjectToDefault 15

powerdeliveryCableVoltageMax 16

pdCableVoltage_Invalid 0

pdCableVoltage_20VDC 1

pdCableVoltage_30VDC 2

pdCableVoltage_40VDC 3

pdCableVoltage_50VDC 4

powerdeliveryCableCurrentMax 17

pdCableCurrent_Invalid 0

pdCableCurrent_3Amps 1

pdCableCurrent_5Amps 2

powerdeliveryCableSpeedMax 18

pdCableSpeed_Invalid 0

pdCableSpeed_USB2p0 1

pdCableSpeed_USB3p2_Gen1 2

pdCableSpeed_USB3p2_USB4p0_Gen2 3

pdCableSpeed_USB4p0_Gen3 4

pdCableSpeed_USB4p0_Gen4 5

powerdeliveryCableType 19

pdCableType_Invalid 0

pdCableType_Passive 1

pdCableType_Active 2

powerdeliveryCableOrientation 20

pdCableOrientation_Invalid 0

pdCableOrientation_CC1 1

pdCableOrientation_CC2 2

powerdeliveryPowerRoleCapabilities 21

pdPowerRoleCapabilities_None 0

pdPowerRoleCapabilities_Source 1

pdPowerRoleCapabilities_Sink 2

pdPowerRoleCapabilities_DualRole 3

powerdeliveryPowerRole 22

pdPowerRole_None 0

pdPowerRole_Source 1

pdPowerRole_Sink 2

pdPowerRole_SourceSink 3

powerdeliveryPowerRolePreferred 23

pdPowerRolePreferred_None 0

pdPowerRolePreferred_Source 1

pdPowerRolePreferred_Sink 2

pdPowerRolePreferred_FollowData 3

pdPowerRolePreferred_Auto 4

powerdeliveryPeakCurrentConfiguration 24

powerdeliveryFastRoleSwapCurrent 25

powerdeliveryDataRoleCapabilities 26

pdDataRoleCapabilities_None 0

pdDataRoleCapabilities_DFP 1

pdDataRoleCapabilities_UFP 2

pdDataRoleCapabilities_DualRole 3

powerdeliveryOverride 41

pdOverrideCableCurrent 0

0 - Override the cable current limit and provide max of 5A.

`pdOverridePortPower` 1

1 - Override the port power limit and provide the port power limit instead of a derating.

`pdOverrideAutoDiscovery` 2

2 - Override the auto discovery and stop message traffic at a negotiated contract.

`pdOverrideDataPathDisable` 3

3 - Override the automatic disable of data lines when the PD data role and ports configured data role are different.

`powerdeliveryRequestCommand` 42

`pdRequestHardReset` 0

`pdRequestSoftReset` 1

`pdRequestDataReset` 2

`pdRequestPowerRoleSwap` 3

`pdRequestPowerFastRoleSwap` 4

`pdRequestDataRoleSwap` 5

`pdRequestVconnSwap` 6

`pdRequestSinkGoToMinimum` 7

`pdRequestRemoteSourcePowerDataObjects` 8

`pdRequestRemoteSinkPowerDataObjects` 9

`pdRequestRemoteSourceExtendedCapabilities` 10

`pdRequestRemoteSinkExtendedCapabilities` 11

`pdRequestStatus` 12

`pdRequestPPSStatus` 13

`pdRequestBatteryCapabilities` 14

`pdRequestBatteryStatus` 15

pdRequestManufacturerInfoSop 16

pdRequestManufacturerInfoSopp 17

pdRequestManufacturerInfoSoppp 18

pdRequestDiscoverIdentitySop 19

pdRequestDiscoverIdentitySopp 20

pdRequestDiscoverIdentitySoppp 21

pdRequestRevision 22

pdRequestSourceInfo 23

pdRequestCountryCode 24

pdRequestCountryInfo 25

pdRequestRemoteSourceEPRCapabilities 26

pdRequestRemoteSinkEPRCapabilities 27

powerdeliveryRequestStatus 43

powerdeliveryFlagMode 44

pdFlagDualRoleData 1

pdFlagDualRolePower 2

pdFlagUnconstrainedPower 3

pdFlagSuspendPossible 4

pdFlagUSBComPossible 5

pdFlagUnchunkedMessageSupport 6

pdFlagHigherCapability 7

pdFlagCapabilityMismatch 8

pdFlagGivebackFlag 9

pdFlagLast 10

powerdeliveryLogEnable 45

powerdeliveryLogPacket 46

powerdeliveryLogEvent 47

pdEventNone 0

pdEventPacket 1

pdEventConnect 2

pdEventDisconnect 3

pdEventCableResetReceived 4

pdEventCableResetSent 5

pdEventHardResetReceived 6

pdEventHardResetSent 7

pdEventMessageTransmitFailed 8

pdEventMessageTransmitDiscarded 9

pdEventPDFunctionDisabled 10

pdEventVBUSEnabled 11

pdEventVBUSDisabled 12

pdEventVCONNEnabled 13

pdEventVCONNDisabled 14

`pdEventRp1A5` 15

`pdEventRp3A0` 16

`pdEventBistEnter` 17

`pdEventBistExit` 18

`pdEventInvalidPacket` 19

`pdEventBistSharedCapacityEnter` 20

`pdEventBistSharedCapacityExit` 21

`pdEventLast` 22

`powerdeliveryVDM` 48

`powerdeliveryProtocolRevisionMax` 49

49 - Maximum USB Power Delivery protocol revision for negotiation (API values `pdProtocolRevision2` or `pdProtocolRevision3` only; unrelated to hardware revision)

`pdProtocolRevision2` 2

2 - PD specification revision 2.0

`pdProtocolRevision3` 3

3 - PD specification revision 3.x

`powerdeliveryNumberOfOptions` 55

group `cmdPOWERDELIVERY_Command_Constants`

Defines

`powerdeliveryPartnerLocal` 0

`powerdeliveryPartnerRemote` 1

`powerdeliveryPowerRoleDisabled` 0

`powerdeliveryPowerRoleSource` 1

`powerdeliveryPowerRoleSink` 2

`powerdeliveryPowerRoleSourceSink` 3

`powerDeliveryRequestStatus` 43

Backwards compatibility for `powerDeliveryRequestStatus`.

Rail Entity

group `cmdRAIL_Defines`

Rail Command defines.

Defines

`cmdRAIL` 32

32 - Rail command code.

group `cmdRAIL_Command_Options`

Defines

`railVoltage` 1

1 - Rail Voltage Option Code

`railCurrent` 2

2 - Rail Current Option Code

`railCurrentLimit` 3

3 - Rail Current limit Option Code

`railTemperature` 4

4 - Rail Temperature Option Code

`railEnable` 5

5 - Rail Enable Option Code

`railValue` 6

6 - Rail Value Option Code

`railKelvinSensingEnable` 7

7 - Rail Kelvin sensing Mode Option Code

kelvinSensingOff_Value 0

0 - Kelvin Sensing off mode for Kelvin Sensing mode

kelvinSensingOn_Value 1

1 - Kelvin Sensing on mode for Kelvin Sensing mode

railKelvinSensingState 8

8 - Kelving Sensing state Option Code

railOperationalMode 9

9 - Operational Mode Option Code

railOperationalMode_HardwareConfiguration_Offset 0

0 - Operational Mode hardware configuration offset region (bits[0:3])

railOperationalModeAuto_Value 0

0 - Auto operational mode for operational mode

railOperationalModeLinear_Value 1

1 - Linear mode for operational mode

railOperationalModeSwitcher_Value 2

2 - Switcher mode for operational mode

railOperationalModeSwitcherLinear_Value 3

3 - Switcher Linear mode for operational mode

railOperationalMode_Mode_Offset 4

4 - Operational Mode offset region (bits[4:7])

railOperationalModeConstantCurrent_Value 0

0 - Constant Current mode for operational mode

railOperationalModeConstantVoltage_Value 1

1 - Constant Voltage mode for operational mode

railOperationalModeConstantPower_Value 2

2 - Constant Power mode for operational mode

railOperationalModeConstantResistance_Value 3

3 - Constant Resistance mode for operational mode

railOperationalModeFactoryReserved_Value 0xF

0xF - Factory Reserved Operating Mode.

DefaultOperationalRailMode_Value 0
0 - Default operational mode for operational mode

railOperationalState 10
10 - Operational state Option Code

railOperationalState_Initializing_Bit 0
0 - Initializing bit for operational state

railOperationalState_Enabled_Bit 1
1 - Enabled bit for operational state

railOperationalState_Fault_Bit 2
2 - Fault bit for operational state

railOperationalState_HardwareConfiguration_Offset 8
8 - Hardware Configuration region (bits[8-15]) for operational state.

railOperationalStateLinear_Value 0
0 - Linear state for operational state option mode.

railOperationalStateSwitcher_Value 1
1 - Switcher state for operational state option mode.

railOperationalStateSwitcherLinear_Value 2
2 - Switcher Linear state for operational state option mode.

railOperationalStateOverVoltageFault_Bit 16
16 - Over Voltage Fault bit for operational state option mode.

railOperationalStateUnderVoltageFault_Bit 17
17 - Under Voltage Fault bit for operational state option mode.

railOperationalStateOverCurrentFault_Bit 18
18 - Over Current Fault bit for operational state option mode.

railOperationalStateOverPowerFault_Bit 19
19 - Over Power Fault bit for operational state option mode.

railOperationalStateReversePolarityFault_Bit 20
20 - Reverse Polarity Fault bit for operational state option mode.

railOperationalStateOverTemperatureFault_Bit 21
21 - Over Temperature Fault bit for operational state option mode.

railOperationalStateReverseCurrentFault_Bit 22
22 - Reverse Current Fault bit for operational state option mode.

railOperationalStateOperatingMode_Offset 24
24 - Operating Mode region (bits[24:31]) for operational state.

railOperationalStateConstantCurrent_Value 0
0 - Constant Current mode for operational state

railOperationalStateConstantVoltage_Value 1
1 - Constant Voltage mode for operational state

railOperationalStateConstantPower_Value 2
2 - Constant Power mode for operational state option codes.

railOperationalStateConstantResistance_Value 3
3 - Constant Resistance mode for operational state

railVoltageSetpoint 11
11 - Rail Setpoint Voltage option code

railCurrentSetpoint 12
12 - Rail Setpoint Current Option Code

railVoltageMinLimit 13
13 - Rail Voltage min limit Option Code

railVoltageMaxLimit 14
14 - Rail Voltage max limit Option Code

railPower 15
15 - Rail power Option Code

railPowerSetpoint 16
16 - Rail Setpoint power Option Code

railPowerLimit 17
17 - Rail power limit Option Code

railResistance 18
18 - Rail resistance Option Code

railResistanceSetpoint 19
19 - Rail Setpoint resistance Option Code

railClearFaults 20

20 - Rail Clear Fault Codes. Option Code

railNumberOfOptions 21

21 - Number of Options for Rail, always last entry

group **cmdRAIL_Command_Constants**

Defines

railFactoryReserved 62

Factory Reserved Code.

railFactoryReserved2 63

Factory Reserved Code.

Relay Entity

group **cmdRELAY_Defines**

Relay Command defines.

Defines

cmdRELAY 34

34 - Relay command code.

group **cmdRELAY_Command_Options**

Defines

relayEnable 1

1 - Enable Option Code

relayVoltage 2

2 - Voltage Option Code

relayNumberOfOptions 3

3 - Number of Options for relay, always last entry

RCServo Entity

group **cmdSERVO_Defines**

RCServo Command defines.

Defines

cmdSERVO 13

13 - RCServo command code.

group **cmdSERVO_Command_Options**

Defines

servoEnable 1

1 - Enable Option Code

servoPosition 2

2 - Position Option Code

servoReverse 3

3 - Reverse Option Code

servoNumberOfOptions 4

4 - Number of Options for servo, always last entry

Signal Entity

group **cmdSIGNAL_Defines**

Signal Command defines.

Defines

cmdSIGNAL 14

14 - Signal command code.

group **cmdSIGNAL_Command_Options**

Defines

signalEnable 1

1 - Enable Option Code

signalInvert 2

2 - Inversion of Duty Cycle Option Code

signalT3Time 3

3 - Period Option Code

signalT2Time 4

4 - Active Time Option Code

signalNumberOfOptions 5

5 - Number of Options for signal, always last entry

Slot Entity

group **cmdSLOT_Defines**

Slot Command defines.

Defines

cmdSLOT 4

4 - Slot command code.

group **cmdSLOT_Command_Options**

Defines

slotCapacity 1

1 - Capacity Option Code

slotSize 2

2 - Size Option Code

slotOpenRead 3

3 - Open Read Only Option Code

slotOpenWrite 4

4 - Open Read Write Option Code

slotSeek 5

5 - Seek Option Code

slotRead 6

6 - Read Option Code

slotWrite 7

7 - Write Option Code

slotClose 8

8 - Close Option Code

slotNumberOfOptions 9

9 - Number of Options for slot, always last entry

group **cmdSLOT_Command_Constants**

Defines

bitSlotError 0x80

Bit Slot Error.

Store Entity

group **cmdSTORE_Defines**

Store Command defines.

Defines

cmdSTORE 77

77 - Store command code.

group **cmdSTORE_Command_Options**

Defines

storeSlotEnable 1

1 - Slot Enable Option Code

storeSlotDisable 2

2 - Slot Disable Option Code

storeSlotState 3

3 - Slot State Option Code

storeWriteSlot 4

4 - Slot Write Option Code

storeReadSlot 5

5 - Slot Read Option Code

storeCloseSlot 6

6 - Slot Close Option Code

storeLock 7

7 - Slot Lock Option Code

storeNumberOfOptions 8

8 - Number of Options for store, always last entry

group **cmdSTORE_Command_Constants**

Defines

storeInternalStore 0

Internal Store Type.

storeRAMStore 1

RAM Store Type.

storeSDStore 2

SD Store Type.

storeEEPROMStore 3

EEPROM Store Type.

storeMaxStoreIndex 3

Maximum Store Type Index.

Stream Entity

group **cmdSTREAM_Defines**
Stream Command defines.

Defines

cmdSTREAM 93
93 - Stream command code.

group **cmdSTREAM_Command_Options**

Defines

streamEnable 1
1 - Enable Stream Key Option Code

streamDisable 2
2 - Disable Stream Key Option Code

streamCapacity 3
3 - Stream Key Capacity Option Code

streamNumberOfOptions 4
4 - Number of Options for stream, always last entry

System Entity

group **cmdSYSTEM_Defines**
System Command defines.

Defines

cmdSYSTEM 3
3 - System command code.

group **cmdSYSTEM_Command_Options**

Defines

systemModule 1

1 - Module address Option Code

systemRouter 2

2 - Router address Option Code

systemHBInterval 3

3 - Heartbeat interval Option Code

systemLED 4

4 - User LED Option Code

systemSleep 5

5 - Sleep Option Code

systemBootSlot 6

6 - Boot Slot Option Code

aSystemBootSlotNone 255

255 - Disable boot slot value for Boot Slot option.

systemVersion 7

7 - Firmware Version Option Code

systemModel 8

8 - Model Option Code

systemSerialNumber 9

9 - Serial Number Option Code

systemSave 10

10 - System save Option Code

systemReset 11

11 - System reset Option Code

systemInputVoltage 12

12 - Input voltage Option Code

systemModuleHardwareOffset 13

13 - Module Offset Option Code

systemModuleBaseAddress 14

14 - Module Base address Option Code

systemModuleSoftwareOffset 15
15 - Module Software offset Option Code

systemRouterAddressSetting 16
16 - Router address setting Option Code

systemIPConfiguration 17
17 - IP configuration setting option code

systemIPModeDHCP 0
0 - DHCP Configuration

systemIPModeStatic 1
1 - Static Configuration

systemIPModeDefault 0
0 - Default Configuration

systemIPAddress 18
18 - IP address setting option code

systemIPStaticAddressSetting 19
19 - Static IP address setting option code

systemRouteToMe 20
20 - Route to me setting option code

systemInputCurrent 21
21 - Input current Option Code

systemUptime 22
22 - System uptime Option Code

systemMaxTemperature 23
23 - System max temperature Option Code

systemLogEvents 24
24 - System log events Option Code

systemUnregulatedVoltage 25
25 - Unregulated System Voltage Option Code

systemUnregulatedCurrent 26
26 - Unregulated System Current Option Code

systemTemperature 27

27 - System temperature option code

systemMinTemperature 28

28 - System min temperature option code

systemInputPowerSource 29

29 - System input power source option code

systemInputPowerBehavior 30

30 - System input power behavior option code

systemInputPowerBehaviorConfig 31

31 - System input power behavior config option code

systemName 32

32 - System name option code

systemPowerLimit 33

33 - System power limit option code

systemPowerLimitMax 34

34 - System power limit max option code

systemPowerLimitState 35

35 - System power limit state option code

systemResetEntityToFactoryDefaults 36

systemResetDeviceToFactoryDefaults 37

systemLinkInterface 38

38 - Setting the link interface for control Option Code

systemLinkAuto 0

0 - System Link is automatically defined

systemLinkUSBControl 1

1 - System Link through control port

systemLinkUSBHub 2

2 - System Link through the Hub (upstream connection)

systemLinkUSBA11 3

3 - System Link through the all available connections

systemReserved 39

39 - Reserved Option Code for Acroname Internal Use Only Option Code

systemHardwareVersion 40

40 - Hardware Version option code

systemErrors 41

41 - System Error option code

systemErrors_ThermalProtection_Bit 0

0 - Thermal Protection bit for operational Errors

systemErrors_OutputPowerProtection_Bit 1

1 - Output Power Protection bit for operational Errors

systemLEDMaxBrightness 42

42 - System LED Brightness option code

systemBuild 43

43 - Firmware build option code

systemProtocolFeatures 45

45 - Firmware protocol features Option Code

systemProtocolFeatures_pooledPackets_Bit 31

31 - Defines bit if the protocol supports pooling of brainstem packets into a single USB frame.

systemNumberOfOptions 46

46 - Number of Options for System, always last entry

Temperature Entity

group **cmdTEMPERATURE_Defines**

Temperature Command defines.

Defines

cmdTEMPERATURE 33

33 - Temperature command code.

group **cmdTEMPERATURE_Command_Options**

Defines

temperatureMicroCelsius 1

1 - Temperature Value Option Code

temperatureMinimumMicroCelsius 2

2 - Minimum Recorded Temperature Option Code

temperatureMaximumMicroCelsius 3

3 - Maximum Recorded Temperature Option Code

temperatureResetLoggedValues 4

4 - Temperature Logged Value reset option code

temperatureNumberOfOptions 5

5 - Number of Options for temperature, always last entry

Timer Entity

group **cmdTIMER_Defines**

Timer Command defines.

Defines

cmdTIMER 79

79 - Timer command code.

group **cmdTIMER_Command_Options**

Defines

timerExpiration 1

1 - Expiration Value Option Code

timerMode 2

2 - Mode Option Code

timerModeSingle 0

0 - Single Shot Mode

timerModeRepeat 1

1 - Repeated Mode

DefaultTimerMode 0

0 - Default Mode

timerNumberOfOptions 3

3 - Number of Options for timer, always last entry

UART Entity

group **cmdUART_Defines**

UART Command defines.

Defines

cmdUART 35

35 - UART command code.

group **cmdUART_Command_Options**

Defines

uartEnable 1

1 - Enable Option Code

uartBaudRate 2

2 - Baud Rate Option Code

uartBaudRate_Auto_Value 0

0 - Automatically Selected Baud Rate

uartProtocol 3

3 - Protocol Option Code

uartProtocol_Undefined 0

0 - Not Yet Defined Protocol

uartProtocol_Extron_Value 1

1 - Extron Responder (backward compatibility)

uartProtocol_ExtronResponder_Value 1

1 - Extron Responder

uartProtocol_Brainstem_Value 2

2 - Brainstem Transport

`uartProtocol_ExtronInitiator_Value 3`

3 - Extron Initiator

`uartProtocol_Reserved4_Value 4`

4 - Reserved

`uartProtocol_Reserved5_Value 5`

5 - Reserved

`uartProtocol_Loopback_Value 6`

6 - Loopback

`uartProtocol_RestSerial_Value 7`

7 - REST serial API (get/set api/v1/brainstem/...)

`uartLinkChannel 4`

4 - UART Linked Channel Option Code

`uartStopBits 5`

5 - UART Stop Bits Option Code

`uartStopBits_1_Value 0`

0 - 1 Stop Bit

`uartStopBits_1p5_Value 1`

1 - 1.5 Stop Bits

`uartStopBits_2_Value 2`

2 - 2 Stop Bits

`uartParity 6`

6 - UART Parity Option Code

`uartParity_None_Value 0`

0 - No Parity Bit

`uartParity_Odd_Value 1`

1 - Odd Parity Bit

`uartParity_Even_Value 2`

2 - Even Parity Bit

`uartParity_Mark_Value 3`

3 - Mark Parity Bit

`uartParity_Space_Value` 4
4 - Space Parity Bit

`uartDataBits` 7
7 - UART Data Bits per Character Option Code

`uartFlowControl` 8
8 - UART Flow Control Option Code

`uartFlowControl_RTS_CTS_Bit` 0
0 - RTS/CTS Enable

`uartFlowControl_DSR_DTR_Bit` 1
1 - DSR/DTR Enable

`uartFlowControl_XON_XOFF_Bit` 2
2 - XON/XOFF Enable

`uartCapableProtocols` 9
9 - Option Code for UART Capable Protocols

`uartAvailableProtocols` 10
10 - Option Code for UART Available Protocols

`uartNumberOfOptions` 11
11 - Number of Options for UART, always last entry

USB Entity

group `cmdUSB_Defines`
USB Command defines.

Defines

`cmdUSB` 18
18 - USB command code.

group `cmdUSB_Command_Options`

Defines

usbPortEnable 1
1 - Port Enable Option Code

usbPortDisable 2
2 - Port Disable Option Code

usbDataEnable 3
3 - Data Enable Option Code

usbDataDisable 4
4 - Data Disable Option Code

usbPowerEnable 5
5 - Power Enable Option Code

usbPowerDisable 6
6 - Power Disable Option Code

usbPortCurrent 7
7 - Port Current Option Code

usbPortVoltage 8
8 - Port Voltage Option Code

usbHubMode 9
9 - Hub Mode Option Code

usbPortClearErrorStatus 12
12 - Hub Clear Error Status Option Code

usbUpstreamMode 14
14 - Upstream Mode Option Code

usbUpstreamModeAuto 2
2 - UpstreamMode Auto for upstream mode

usbUpstreamModePort0 0
0 - UpstreamMode Port 0 for upstream mode

usbUpstreamModePort1 1
1 - UpstreamMode Port 1 for upstream mode

usbUpstreamModeNone 255
255 - UpstreamMode None to turn off all upstream connections.

usbUpstreamModeDefault 2
2 - UpstreamMode default for upstream mode

usbUpstreamState 15
15 - UpstreamState Option Code

usbUpstreamStateNone 2
2 - UpstreamMode Auto for upstream mode

usbUpstreamStatePort0 0
0 - UpstreamMode Port 0 for upstream mode

usbUpstreamStatePort1 1
1 - UpstreamMode Port 1 for upstream mode

usbHubEnumerationDelay 16
16 - Downstream ports enumeration delay Option Code

usbPortCurrentLimit 17
17 - Set or get the port current limit Option Code

usbUpstreamBoostMode 18
18 - Set/Get upstream boost mode. Option Code

usbDownstreamBoostMode 19
19 - Set/Get downstream boost mode. Option Code

usbBoostMode_0 0
0 - Boost mode off, no boost.

usbBoostMode_4 1
1 - Boost mode 4%.

usbBoostMode_8 2
2 - Boost mode 8%.

usbBoostMode_12 3
3 - Boost mode 12%.

usbPortMode 20
20 - Set/Get Port mode (bit-packed) Option Code

usbPortMode_sdp 0
0 - Standard Downstream port (0.5 Amp).

`usbPortMode_cdp` 1

1 - Charging Downstream port (2.1 Amp).

`usbPortMode_charging` 2

2 - Trickle charging functionality.

`usbPortMode_passive` 3

3 - Electrical passthrough of VBUS.

`usbPortMode_USB2AEnable` 4

4 - USB2 dataline A side enabled.

`usbPortMode_USB2BEnable` 5

5 - USB2 dataline B side enabled.

`usbPortMode_VBusEnable` 6

6 - USB VBUS enabled.

`usbPortMode_SuperSpeed1Enable` 7

7 - USB SS Speed dataline side A enabled.

`usbPortMode_SuperSpeed2Enable` 8

8 - USB SS Speed dataline side B enabled.

`usbPortMode_USB2BoostEnable` 9

9 - USB2 Boost Mode Enabled.

`usbPortMode_USB3BoostEnable` 10

10 - USB3 Boost Mode Enabled.

`usbPortMode_AutoConnectEnable` 11

11 - Auto-connect Mode Enabled.

`usbPortMode_CC1Enable` 12

12 - CC1 Enabled.

`usbPortMode_CC2Enable` 13

13 - CC2 Enabled.

`usbPortMode_SBUEnable` 14

14 - SBU1 Enabled.

`usbPortMode_CCFlipEnable` 15

15 - Flip CC1 and CC2.

usbPortMode_SSFlipEnable 16
16 - Flip Super speed data lines

usbPortMode_SBUFlipEnable 17
17 - Flip Side Band Unit lines.

usbPortMode_USB2FlipEnable 18
18 - Flip Side Band Unit lines.

usbPortMode_CC1InjectEnable 19
19 - Internal Use.

usbPortMode_CC2InjectEnable 20
20 - Internal Use.

usbHiSpeedDataEnable 21
21 - Hi-Speed Data Enable Option Code

usbHiSpeedDataDisable 22
22 - Hi-Speed Data Disable Option Code

usbSuperSpeedDataEnable 23
23 - SuperSpeed Data Enable Option Code

usbSuperSpeedDataDisable 24
24 - SuperSpeed Data Disable Option Code

usbDownstreamDataSpeed 25
25 - Get downstream port speed Option Code

usbDownstreamDataSpeed_na 0
0 - Unknown.

usbDownstreamDataSpeed_hs 1
1 - Hi-Speed (2.0).

usbDownstreamDataSpeed_ss 2
2 - SuperSpeed (3.0).

usbDownstreamDataSpeed_ls 3
3 - TODO

usbConnectMode 26
26 - USB connect mode Option Code

usbManualConnect 0

0 - Auto connect disabled.

usbAutoConnect 1

1 - Auto connect enabled.

usbCC1Enable 27

27 - CC1 Enable option code (USB Type C). Option Code

usbCC2Enable 28

28 - CC2 Enable option code (USB Type C). Option Code

usbSBUEnable 29

29 - SBU1/2 enable option code (USB Type C). Option Code

usbCC1Current 30

30 - CC1 get current option code (USB Type C). Option Code

usbCC2Current 31

31 - CC2 get current option code (USB Type C). Option Code

usbCC1Voltage 32

32 - CC1 get voltage option code (USB Type C). Option Code

usbCC2Voltage 33

33 - CC2 get voltage option code (USB Type C). Option Code

usbPortState 34

34 - TODO Option Code

usbPortError 35

35 - TODO Option Code

usbCableFlip 36

36 - TODO Option Code

usbAltMode 37

37 - USB Alt Mode configuration. Option Code

usbAltMode_disabled 0

0 - Disabled mode.

usbAltMode_normal 1

1 - Normal mode (USB 3.1).

usbAltMode_4LaneDP_ComToHost 2

2 - Alt Mode - 4 lanes of display port "Common" side connected to host.

usbAltMode_4LaneDP_MuxToHost 3

3 - Alt Mode - 4 lanes of display port "Mux" side connected to host.

usbAltMode_2LaneDP_ComToHost_wUSB3 4

4 - Alt Mode - 2 lanes of display port "Common" side connected to host with USB3.1.

usbAltMode_2LaneDP_MuxToHost_wUSB3 5

5 - Alt Mode - 2 lanes of display port "Mux" side connected to host with USB3.1.

usbAltMode_2LaneDP_ComToHost_wUSB3_Inverted 6

6 - Alt Mode - 2 lanes of display port "Common" side connected to host with USB3.1 with channels 1,2 and 3,4 inverted.

usbAltMode_2LaneDP_MuxToHost_wUSB3_Inverted 7

7 - Alt Mode - 2 lanes of display port "Mux" side connected to host with USB3.1 with channels 1,2 and 3,4 inverted.

usbAltMode_USB4_ComToHost 8

8 - Alt Mode - USB4 with "Common" side connected to host.

usbAltMode_USB4_MuxToHost 9

9 - Alt Mode - USB4 with "Mux" side connected to host.

usbAltMode_USB4_ComToHost_Inverted 10

10 - Alt Mode - USB4 with "Common" side connected to host, inverted configuration

usbAltMode_USB4_MuxToHost_Inverted 11

11 - Alt Mode - USB4 with "Mux" side connected to host, inverted configuration

usbSBU1Voltage 38

38 - SBU1 get voltage option code (USB Type C). Option Code

usbSBU2Voltage 39

39 - SBU2 get voltage option code (USB Type C). Option Code

usbNumberOfOptions 40

40 - Number of Options for USB, always last entry. Option Code

USB System Entity

group cmdUSBSYSTEM_Defines

USB System Command defines.

Defines

cmdUSBSYSTEM 38

38 - USB System command code.

group cmdUSBSYSTEM_Command_Options

Defines

usbsystemPowerDataMode 1

1 - Power and Data Mode option code

usbsystemUpstreamPort 2

2 - Upstream Port option code (default port is 0) Option Code

usbsystemUpstreamPortNone 255

255 - UpstreamPort None to turn off all upstream connections or invalid state.

usbsystemEnumerationDelay 3

3 - Enumeration Delay option code

usbsystemDataRoleList 4

4 - Data Role List option code

usbsystemEnabledList 5

5 - Enabled List option code

usbsystemModeList 6

6 - Mode List option code

usbsystemStateList 7

7 - State List option code

usbsystemPowerBehavior 8

8 - Power behavior option code

usbsystemPowerBehavior_Balancing 0

0 - usbsystemPowerBehavior - Balancing between active ports

usbsystemPowerBehavior_Even 1
1 - usbsystemPowerBehavior - Even between all ports

usbsystemPowerBehaviorConfig 9
9 - Power behavior config option code

usbsystemDataBehavior 10
10 - Data behavior option code

usbsystemDataBehavior_HardCoded 0
0 - usbsystemDataBehavior - HardCoded ports

usbsystemDataBehavior_FollowPD 1
1 - usbsystemDataBehavior - FollowPD

usbsystemDataBehavior_PortPriority 2
2 - usbsystemDataBehavior - Use port priority

usbsystemDataBehavior_LastConnected 3
3 - usbsystemDataBehavior - Most Recently Connected Upstream

usbsystemDataBehaviorConfig 11
11 - Data behavior config option code

usbsystemSelectorMode 12
12 - Selector mode option code

usbsystemSelectorMode_Disabled 0
0 - Selector Disabled

usbsystemSelectorMode_Upstream 1
1 - Selector Upstream - Toggles through upstreams

usbsystemSelectorMode_Mux 2
2 - Selector Mux - Toggles through mux ports

usbsystemResetEntityToFactoryDefaults 13
13 - Resets USBSystem reset to default option code.

usbsystemOverride 14
14 - Behavior overrides for USB System Option Code

usbsystemOverride_AutoVbusToggle_Bit 0
0 - USB System Override - Auto VBUS Toggle Disable Bit

usbsystemOverride_VbusDetect_Bit 1

1 - USB System Override - VBUS Detect Disable Bit

usbsystemUpstreamHSPort 15

15 - Upstream Port option code (default port is 0) Option Code

usbsystemUpstreamSSPort 16

16 - Upstream Port option code (default port is 0) Option Code

usbsystemDataHSMaxDatarate 17

17 - Max HighSpeed Data Rate that can be negotiated Option Code

usbsystemDataHSMaxDatarate_None 0

0 - Max HighSpeed Data Rate of None 0Mbps

usbsystemDataHSMaxDatarate_LowSpeed 1

1 - Max HighSpeed Data Rate of Low Speed 1.5Mbps

usbsystemDataHSMaxDatarate_FullSpeed 2

2 - Max HighSpeed Data Rate of Full Speed 12Mbps

usbsystemDataHSMaxDatarate_HighSpeed 3

3 - Max HighSpeed Data Rate of High Speed 480Mbps

usbsystemDataSSMaxDatarate 18

18 - Max SuperSpeed Data Rate that can be negotiated Option Code

usbsystemDataSSMaxDatarate_None 0

0 - Max SuperSpeed Data Rate of None 0Gbps

usbsystemDataSSMaxDatarate_SuperSpeed 1

1 - Max SuperSpeed Data Rate of Super Speed 5Gbps (Gen 1)

usbsystemDataSSMaxDatarate_SuperSpeedPlus 2

2 - Max SuperSpeed Data Rate of Super Speed 10Gbps (Gen 2)

usbsystemNumberOfOptions 19

19 - Number of Options for USB System, always last entry

UserConfig Entity

group **cmdUSERCONFIG_Defines**
UserConfig Command defines.

Defines

cmdUSERCONFIG 92
92 - UserConfig command code.

group **cmdUSERCONFIG_Command_Options**

Defines

userconfigSaveEntityToStore 0
0 - Save Entity option code.

userconfigResetEntityToFactoryDefaults 1
1 - Reset Entity to factory defaults option code.

userconfigLoadEntityFromStore 2
2 - Load Entity from saved values option code.

userconfigNumberOfOptions 3
3 - Number of Options for UserConfig, always last entry

3.4.9 aStream.h

group **aStream**
Platform Independent Stream Abstraction.

[*aStream.h*](#) provides a platform independent stream abstraction for common I/O streams. Provides facilities for creating and destroying as well as writing and reading from streams.

typedef void *aStreamRef
Typedef [*aStreamRef*](#) Opaque reference to stream primitive.

group **StreamCallbacks**

Typedefs

```
typedef aErr (*aStreamGetProc)(uint8_t *pData, void *ref)
```

Typedef *aStreamGetProc*.

This callback is defined to read one byte from the concrete stream implementation.

Param pData

The data Buffer to fill.

Param ref

Opaque reference to concrete stream implementation.

Retval aErrNone

Successfully read the byte.

Retval aErrNotReady

No bytes in stream to read.

Retval aErrEOF

Reached the end of the stream.

Retval aErrIO

An error encountered reading from stream.

Return

Function returns aErr values.

```
typedef aErr (*aStreamPutProc)(const uint8_t *pData, void *ref)
```

Typedef *aStreamPutProc*.

This callback is defined to write one byte to the concrete stream implementation.

Param pData

The data Buffer to write.

Param ref

opaque reference to concrete stream implementation.

Retval aErrNone

Successfully wrote the byte.

Retval aErrIO

An error encountered reading from stream.

Return

Function returns aErr values.

```
typedef aErr (*aStreamDeleteProc)(void *ref)
```

Typedef *aStreamDeleteProc*.

This callback is defined to destroy the concrete stream implementation.

Param ref

opaque reference to concrete stream implementation.

Retval aErrNone

Successfully destroyed.

Retval aErrParam

Invalid ref.

Return

Function returns aErr values.

typedef *aErr* (*aStreamWriteProc*)(const uint8_t *pData, const size_t nSize, void *ref)

Typedef *aStreamWriteProc*. (Optional)

Optional multi-byte write for efficiency, not required..

Param pData

The pointer to the data to write to the stream.

Param nSize

The size of the data buffer in bytes.

Param ref

Opaque reference to concrete stream implementation.

Retval aErrNone

Successfully destroyed.

Retval aErrIO

An error encountered reading from stream.

Return

Function returns aErr values.

enum aBaudRate

Enum *aBaudRate*.

Accepted serial stream baudrates.

Values:

enumerator **aBAUD_2400**

2400 baud

enumerator **aBAUD_4800**

4800 baud

enumerator **aBAUD_9600**

9600 baud

enumerator **aBAUD_19200**

19,200 baud

enumerator **aBAUD_38400**

38,400 baud

enumerator **aBAUD_57600**

57,600 baud

enumerator **aBAUD_115200**

115,200 baud

enumerator **aBAUD_230400**
230,400 buad

enum **aSerial_Bits**

Enum *aSerial_Bits*.

The accepted number of serial bits per byte.

Values:

enumerator **aBITS_8**
8 bits

enumerator **aBITS_7**
7 bits

enum **aSerial_Stop_bits**

Enum *aSerial_Stop_bits*.

The accepted number of serial stop bits.

Values:

enumerator **aSTOP_BITS_1**
1 stop bit

enumerator **aSTOP_BITS_2**
2 stop bits

aStreamRef **aStream_Create** (*aStreamGetProc* getProc, *aStreamPutProc* putProc, *aStreamWriteProc* writeProc, *aStreamDeleteProc* deleteProc, const void *procRef)

Base Stream creation procedure.

Creates a Stream Reference.

Parameters

- **getProc** – - Callback for reading bytes from the underlying stream.
- **putProc** – - Callback for writing bytes to the underlying stream.
- **writeProc** – - Optional callback for optimized writing of multiple bytes.
- **deleteProc** – - Callback for safe destruction of underlying resource.
- **procRef** – - opaque reference to the underlying resource,

Returns

Function returns *aStreamRef* on success and NULL on error.

aErr **aStream_CreateFileInput** (const char *pFilename, *aStreamRef* *pStreamRef)

Create a file input stream.

Creates a file input stream.

Parameters

- **pFilename** – - The filename and path of the file to read from.

- **pStreamRef** - - The resulting stream accessor for the input file.

Return values

- **aErrNone** - - Successful creation.
- **aErrNotFound** - - The file to read was not found.
- **aErrIO** - - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateFileOutput** (const char *pFilename, *aStreamRef* *pStreamRef)

Create a file output stream.

Creates a file output stream.

Parameters

- **pFilename** - - The filename and path of the file to write to.
- **pStreamRef** - - The resulting stream accessor for the output file.

Return values

- **aErrNone** - - Successful creation.
- **aErrIO** - - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateSerial** (const char *pPortName, const *aBaudRate* nBaudRate, const bool parity, const *aSerial_Bits* bits, const *aSerial_Stop_bits* stop, *aStreamRef* *pStreamRef)

Create a serial communication stream.

Creates a serial stream.

Parameters

- **pPortName** - - The portname of the serial device.
- **nBaudRate** - - The baudrate to connect to the device at.
- **parity** - - Whether serial parity is enabled.
- **bits** - - The number of bits per serial byte.
- **stop** - - The number of stop bits per byte.
- **pStreamRef** - - The resulting stream accessor for the serial device.

Return values

- **aErrNone** - - Successful creation.
- **aErrConnection** - - The connection was unsuccessful.
- **aErrIO** - - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateSocket** (const uint32_t address, const uint16_t port, *aStreamRef* *pStreamRef)

Create a TCP/IP socket stream.

Creates a TCP/IP socket stream.

Parameters

- **address** -- The IP4 address of the connection.
- **port** -- The TCP port to connect to.
- **pStreamRef** -- The resulting stream accessor for the TCP connection.

Return values

- **aErrNone** -- Successful creation.
- **aErrConnection** -- The connection was unsuccessful.
- **aErrIO** -- A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateMemory** (const aMemPtr pMemory, const size_t size, *aStreamRef* *pStreamRef)

Create a stream accessor for a block of memory.

Creates a stream accessor for a block of allocated memory. Reads and Writes like any other stream. The memory stream does not make a copy of the memory and doesn't free it but rather provides a stream layer to access it.

Parameters

- **pMemory** -- a pointer to a block of memory.
- **size** -- The size of the block in bytes.
- **pStreamRef** -- The resulting stream accessor for the memory block.

Return values

- **aErrNone** -- Successful creation.
- **aErrParam** -- The memory block is invalid.
- **aErrIO** -- A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateUSB** (const uint32_t serialNum, *aStreamRef* *pStreamRef)

Create a stream to a USB device.

Creates a BrainStem link stream to a USB based module.

Parameters

- **serialNum** -- The BrainStem serial number.
- **pStreamRef** -- The resulting stream accessor for the BrainStem module.

Return values

- **aErrNone** -- Successful creation.
- **aErrNotFound** -- The brainstem device was not found.
- **aErrIO** -- A communication error occurred.

Returns

Function returns aErr values.

aErr **aStreamBuffer_Create** (const size_t nIncSize, *aStreamRef* *pBufferStreamRef)

Create a stream buffer.

Creates a stream buffer.

StreamBuffers are typically used to aggregate a bunch of output into a single pile of bytes. This pile can then be checked for size or accessed as a single block of bytes using the aStreamBuffer_Get call. Finally, these bytes can then be read back out of the buffer until it is empty when it will report an error of aErrEOF. While this stream is thread-safe for different threads doing reads and writes, it is not the best candidate for managing a pipe between threads. Use the aStream_CreatePipe in that scenario as it can be filled and emptied over and over which is typically the use case for cross-thread pipes.

Parameters

- **nIncSize** -- The Increment size to expand the buffer by when it becomes full.
- **pBufferStreamRef** -- The buffer stream resulting from the call.

Return values

- **aErrNone** -- The buffer was successfully created.
- **aErrResource** -- The resources were not available to create the buffer.

Returns

Function returns aErr values.

aErr **aStreamBuffer_Get** (*aStreamRef* pBufferStreamRef, size_t *aSize, uint8_t **ppData)

Get the contents of the buffer.

Get the contents of the buffer.

StreamBuffers are typically used to aggregate a bunch of output into a single pile of bytes. This pile can then be checked for size or accessed as a single block of bytes using the aStreamBuffer_Get call. Finally, these bytes can then be read back out of the buffer until it is empty when it will report an error of aErrEOF. While this stream is thread-safe for different threads doing reads and writes, it is not the best candidate for managing a pipe between threads. Use the aStream_CreatePipe in that scenario as it can be filled and emptied over and over which is typically the use case for cross-thread pipes.

Parameters

- **pBufferStreamRef** -- The buffer stream resulting from the call.
- **aSize** -- The size of the buffered data in bytes.
- **ppData** -- The resulting buffer of the bytes.

Return values

- **aErrNone** -- The buffer was successfully created.
- **aErrParam** -- An invalid stream ref was given.

Returns

Function returns aErr values.

aErr **aStream_CreatePipe** (*aStreamRef* *pBufferStreamRef)

Create a pipe buffered stream.

Get the contents of the buffer. Offers a pipe that is thread-safe for reading and writing between two different contexts. Returns aErrNotReady when data is not available on reads. Expands a buffer internally to hold data when written to until it is read out (FIFO).

Parameters

pBufferStreamRef - - The buffered stream to create the pipe out of.

Return values

- **aErrNone** - - Successful creation.
- **aErrParam** - - The bufferStream is invalid.

Returns

Function returns aErr values.

aErr **aStreamBuffer_Flush** (*aStreamRef* bufferStreamRef, *aStreamRef* flushStream)

Flush the contents of the buffer.

Flushes the content of the buffer into the flushStream.

Parameters

- **bufferStreamRef** - - The buffered stream to flush.
- **flushStream** - - the stream to flush the buffer into.

Return values

- **aErrNone** - - The flush succeeded.
- **aErrParam** - - The bufferStream is invalid.
- **aErrIO** - - IO error writing to flushStream.

Returns

Function returns aErr values.

aErr **aStream_CreateLogStream** (const *aStreamRef* streamToLog, const *aStreamRef* upStreamLog, const *aStreamRef* downStreamLog, *aStreamRef* *pLogStreamRef)

Create a Logging stream.

Creates a stream which contains an upstream log stream and a downstream log stream. The logging stream logs reads to the upstream log and writes to the downstream log, while passing all data to and from the pLogStreamRef.

Parameters

- **streamToLog** - - The reference to the stream to log.
- **upStreamLog** - - Log stream for reads.
- **downStreamLog** - - Log stream for writes.
- **pLogStreamRef** - - The logged stream reference.

Return values

- **aErrNone** - - Successful creation.
- **aErrParam** - - The stream to log is invalid.

Returns

Function returns aErr values.

Returns

aErrIO - A communication error occurred creating the logging stream.

aErr **aStream_Read** (*aStreamRef* streamRef, uint8_t *pBuffer, const size_t length)

Read a byte array record from a stream.

Read a byte array record from a stream.

Parameters

- **streamRef** -- The reference to the stream to read from.
- **pBuffer** -- byte array buffer to read into.
- **length** -- the length of the read buffer.

Return values

- **aErrNone** -- Successful read.
- **aErrMode** -- The streamRef is not readable.
- **aErrIO** -- An error occurred reading the data.

Returns

Function returns aErr values.

aErr **aStream_Write** (*aStreamRef* streamRef, const uint8_t *pBuffer, const size_t length)

Write a byte array to a Stream.

Write a byte array to a Stream.

Parameters

- **streamRef** -- The reference to the stream to write to.
- **pBuffer** -- byte array to write out to the stream.
- **length** -- the byte array length

Return values

- **aErrNone** -- Successful write.
- **aErrMode** -- The streamRef is not writable.
- **aErrIO** -- An error occurred writing the data.

Returns

Function returns aErr values.

aErr **aStream_ReadRecord** (*aStreamRef* streamRef, uint8_t *pBuffer, size_t *lengthRead, const size_t maxLength, const uint8_t *recordTerminator, const size_t terminatorLength)

Read a byte array record from a stream with a record terminator.

Read a byte array record from a stream with a record terminator.

Parameters

- **streamRef** -- The reference to the stream to read from.
- **pBuffer** -- Byte array buffer to read into.
- **lengthRead** -- The length of the read buffer.
- **maxLength** -- The Maximum record length.
- **recordTerminator** -- The byte array representing the record terminator.
- **terminatorLength** -- The length of the record terminator.

Return values

aErrNone - - Successful read.

Returns

Function returns aErr values.

Returns

aErrMode - The streamRef is not readable.

Returns

aErrIO - An error occurred reading the data.

aErr **aStream_WriteRecord** (*aStreamRef* streamRef, const uint8_t *pBuffer, const size_t bufferSize, const uint8_t *recordTerminator, const size_t terminatorLength)

Write a byte array with a record terminator to a Stream.

Write a byte array with a record terminator to a Stream.

Parameters

- **streamRef** - - The reference to the stream to write to.
- **pBuffer** - - byte array to write out to the stream.
- **bufferLength** - - the byte array length
- **recordTerminator** - - the byte array representing the record terminator
- **terminatorLength** - - the length of the record terminator.

Return values

- **aErrNone** - - Successful write.
- **aErrMode** - - The streamRef is not writable.
- **aErrIO** - - An error occurred writing the data.

Returns

Function returns aErr values.

aErr **aStream_ReadCString** (*aStreamRef* streamRef, char *pBuffer, const size_t maxLength)

Read a null terminated string from Stream.

Read a null terminated string from Stream.

Parameters

- **streamRef** - - The reference to the stream to read from.
- **pBuffer** - - Character array buffer to read into.
- **maxLength** - - The maximum length of the string.

Return values

- **aErrNone** - - Successful read.
- **aErrMode** - - The streamRef is not readable.
- **aErrIO** - - An error occurred reading the data.

Returns

Function returns aErr values.

aErr **aStream_WriteCString** (*aStreamRef* streamRef, const char *pBuffer)

Write a null terminated string.

Write a null terminated string.

Parameters

- **streamRef** - - The reference to the stream to write to.
- **pBuffer** - - character array to write.

Return values

- **aErrNone** - - Successful write.
- **aErrMode** - - The streamRef is not writable.
- **aErrIO** - - An error occurred writing the data.

Returns

Function returns aErr values.

aErr **aStream_ReadCStringRecord** (*aStreamRef* streamRef, char *pBuffer, const size_t maxLength, const char *recordTerminator)

Read a null terminated string with a record terminator to pBuffer.

Read a null terminated string with a record terminator to pBuffer.

Parameters

- **streamRef** - - The reference to the stream to read to.
- **pBuffer** - - character array to read to.
- **maxLength** - - The maximum number of characters to read.
- **recordTerminator** - - The record terminator to read to.

Return values

- **aErrNone** - - Successful read.
- **aErrMode** - - The streamRef is not readable.
- **aErrIO** - - An error occurred reading the data.

Returns

Function returns aErr values.

aErr **aStream_WriteCStringRecord** (*aStreamRef* streamRef, const char *pBuffer, const char *recordTerminator)

Write a null terminated string with a record terminator to the stream.

Write a null terminated string with a record terminator to the stream.

Parameters

- **streamRef** - - The reference to the stream to be written to.
- **pBuffer** - - Null terminated string to write to the stream.
- **recordTerminator** - - The record terminator to write after the contents.

Return values

- **aErrNone** - - Successful write.
- **aErrMode** - - The streamRef is not writable.

- **aErrIO** – - An error occurred writing the data.

Returns

Function returns aErr values.

aErr **aStream_Flush** (*aStreamRef* inStreamRef, *aStreamRef* outStreamRef)

Flush contents of inStream into outStream.

Flush the entire current content of the instream into the outstream.

Parameters

- **inStreamRef** – - The reference to the stream to be flushed into the outstream.
- **outStreamRef** – - The reference to the stream instream is flushed into.

Return values

- **aErrNone** – - Successful Flush.
- **aErrMode** – - The outstream is not writable or instream is not readable.
- **aErrIO** – - An error occurred flushing the data.

Returns

Function returns aErr values.

aErr **aStream_Destroy** (*aStreamRef* *pStreamRef)

Destroy a Stream.

Safely destroy a stream and deallocate the associated resources.

Parameters

pStreamRef – - A pointer to a pointer of a valid streamRef. The StreamRef will be set to NULL on successful destruction of the stream.

Return values

- **aErrNone** – - The stream was successfully destroyed.
- **aErrParam** – - If the streamRef is invalid.

Returns

Function returns aErr values.

3.4.10 aTime.h

group aTime

Basic Time procedures Sleep and Get process tics.

aTime.h provides a platform independent interface for millisecond sleep, and for getting process tics.

unsigned long **aTime_GetMSTicks** (void)

Get the current tick count in milliseconds.

This call returns a number of milliseconds. Depending on the platform, this can be the number of milliseconds since the last boot, or from the epoc start. As such, this call should not be used as an external reference clock. It is accurate when used as a differential, i.e. internal, measurement only.

Returns

unsigned long number of milliseconds elapsed.

aErr **aTime_MSSleep** (const unsigned long msTime)

Sleep the current process for msTime milliseconds.

Sleeps the current process. This is not an active sleep, there are no signals which will “wake” the process.

Parameters

msTime – Milliseconds to sleep.

Return values

- **aErrNone** – The call returned successfully.
- **aErrUnknown** – Um unknown what went wrong.

Returns

Function returns aErr values.

3.4.11 aUEI.h

group **aUEI**

UEI Utilities.

aUEI.h Provides structs and utilities for working with UEIs.

enum **dataType**

Typedef Enum *dataType*.

UEI datatype

Values:

enumerator **aUEI_VOID**

Void datatype.

enumerator **aUEI_BYTE**

Char datatype.

enumerator **aUEI_SHORT**

Short datatype.

enumerator **aUEI_INT**

Int datatype.

enumerator **aUEI_BYTES**

Bytes datatype.

struct **uei**

Typedef Struct *uei*.

UEI data struct.

Public Members**uint8_t module**

Module address.

uint8_t command

Command code.

uint8_t option

option code & UEI operation.

uint8_t specifier

Entity index & response specifier.

uint8_t byteVal

Char value union member.

uint16_t shortVal

Short value union member.

uint32_t intVal

Int value union member.

dataType **type**

Union dataType.

uint16_t aUEI_RetrieveShort (const uint8_t *p)

Retrieve a short from a UEI.

Parameters**p** – Pointer to byte array containing short.**Returns****uint16_t** The short value.**void aUEI_StoreShort** (uint8_t *p, uint16_t v)

Store a short in a UEI.

Parameters

- **p** – Pointer to uei shortVal.
- **v** – Short value to store.

uint32_t aUEI_RetrieveInt (const uint8_t *p)

Retrieve an Int from a UEI.

Parameters**p** – Pointer to byte array containing the Int.**Returns****uint32_t** The integer value.

void **aUEI_StoreInt** (uint8_t *p, uint32_t v)

Store an Int in a UEI.

Parameters

- **p** – Pointer to the IntVal of a UEI.
- **v** – The value to store.

3.4.12 aVersion.h

group **aVersion**

Library version interface.

[aVersion.h](#) Provides version information for the BrainStem2 library.

aVERSION_MAJOR 2

Major revision level of library.

Major revision bumps will break compatibility with existing versions and may introduce protocol changes or other fundamental differences.

aVERSION_MINOR 12

Minor revision level of library.

Minor revisions should largely be compatible, however new features may be added with a minor revision change.

aVERSION_PATCH 3

Patch revision level of library.

Patch revisions are bug fixes and small performance changes. They add no significant new features or interfaces.

group **Firmware_version_parsing**

Functions

uint8_t **aVersion_ParseMajor** (uint32_t build)

Parse out the major revision number.

Parses the major revision level from the given uint32.

Parameters

build – The packed version number returned from the system.getVersion call.

Returns

The major revision number.

uint8_t **aVersion_ParseMinor** (uint32_t build)

Parse out the minor revision number.

Parses the minor revision level from the given uint32.

Parameters

build – The packed version number returned from the system.getVersion call.

Returns

The minor revision number.

uint32_t **aVersion_ParsePatch** (uint32_t build)

Parse out the revision patch number.

Parses the revision patch level from the given uint32.

Parameters

build – The packed version number returned from the system.getVersion call.

Returns

The revision patch number.

void **aVersion_ParseString** (uint32_t build, char *string, size_t len)

Parse the Version number into a human readable format.

Fills the string parameter with a human readable formatted version number.

Parameters

- **build** – The packed version number returned from the system.getVersion call.
- **string** – The string to fill with the version string.
- **len** – The length of the filled string, not longer than MAX_VERSION_STRING.

bool **aVersion_IsLegacyFormat** (uint32_t build)

Check if the given build version is of the legacy packing format.

Parses the revision format from the given uint32.

Parameters

build – The packed version number returned from the system.getVersion call.

Returns

Whether the revision format is the old packing format.

uint8_t **aVersion_GetMajor** (void)

Return the major revision number.

Returns

The major revision number.

uint8_t **aVersion_GetMinor** (void)

Return the minor revision number.

Returns

The minor revision number.

uint32_t **aVersion_GetPatch** (void)

Return the revision patch number.

Returns

The revision patch number.

const char ***aVersion_GetString** (void)

Return a human readable version string.

Returns

char* human readable version string.

bool **aVersion_IsAtLeast** (const uint8_t major, const uint8_t minor, const uint8_t patch)

Check that the current software version is at least major.minor.patch.

Parameters

- **major** – The major revision level.
- **minor** – The minor revision.
- **patch** – The patch level.

Returns

True when current version is at least what is given, false otherwise

char ***aVersion_GetFeatureList** (void)

Get an array of the features the library supports.

Returns

an array of c strings describing the features the library supports.

void **aVersion_DestroyFeatureList** (char **featureList)

Destroy the feature list.

Parameters

featureList – pointer to featurelist.

3.4.13 PortMapping.h

group **PortMapping**

BrainStem Port Mapping Interface.

[*PortMapping.h*](#) provides an interface for usb descriptor information of devices downstream of Acroname hub products.

enum **PORT_SPEED**

Port speed enumeration

Values:

enumerator **kPORT_SPEED_UNKNOWN**

kPORT_SPEED_UNKNOWN (0)

enumerator **kPORT_SPEED_LOW**

kPORT_SPEED_LOW (1)

enumerator **kPORT_SPEED_FULL**

kPORT_SPEED_FULL (2)

enumerator **kPORT_SPEED_HIGH**

kPORT_SPEED_HIGH (3)

enumerator **kPORT_SPEED_SUPER**

kPORT_SPEED_SUPER (4)

enumerator **kPORT_SPEED_SUPER_PLUS**
 kPORT_SPEED_SUPER_PLUS (5)

struct **DeviceNode**

Device Node Structure - Contains information linking the downstream device to the Acroname Hub.

Public Members

uint32_t **hubSerialNumber**

Serial number of the Acroname hub where the device was found.

uint8_t **hubPort**

Port of the Acroname hub where the device was found.

uint16_t **idVendor**

Manufactures Vendor ID of the downstream device.

uint16_t **idProduct**

Manufactures Product ID of the downstream device.

PORT_SPEED_t **speed**

The devices downstream device speed.

char **productName**[255]

USB string descriptor

char **serialNumber**[255]

USB string descriptor

char **manufacturer**[255]

USB string descriptor

aErr **getDownstreamDevices** (DeviceNode_t *buffer, uint32_t bufferLength, uint32_t *devicesFound)

Gets downstream device USB information for all Acroname hubs.

Parameters

- **buffer** – Pointer to the start of a list/array to be used by the function.
- **bufferLength** – Size of the list/array in DeviceNode_t's, not bytes.)
- **devicesFound** – The number of DeviceNode_t's that were populated.

Return values

- **aErrNone** – on success
- **aErrParam** – Passed in values are not valid. (NULL, size etc).
- **aErrMemory** – No more room in the list
- **aErrNotFound** – No Acroname devices were found.

3.4.14 `bs_pd_packet.h`

group **BSPDPacket**

BrainStem Power Delivery Packet.

[*bs_pd_packet.h*](#) provides the packet structure which is used with the PDChannelLogger interface.

enum **BS_PD_Packet_Direction**

Packet Direction enumeration

Values:

enumerator **kBS_PD_Packet_Direction_Invalid**

enumerator **kBS_PD_Packet_Direction_Transmit**

enumerator **kBS_PD_Packet_Direction_Receive**

enumerator **kBS_PD_Packet_Direction_Sniff**

enumerator **kBS_PD_Packet_Direction_UNKNOWN**

enumerator **kBS_PD_Packet_Direction_LAST**

enum **BS_PD_Packet_SOP**

Packet SOP type enumeration

Values:

enumerator **kBS_PD_Packet_SOP**

enumerator **kBS_PD_Packet_SOP_P**

enumerator **kBS_PD_Packet_SOP_P_P**

enumerator **kBS_PD_Packet_SOP_P_P_DEBUG**

enumerator **kBS_PD_Packet_SOP_P_P_P_DEBUG**

enumerator **kBS_PD_Packet_SOP_UNKNOWN**

enumerator **kBS_PD_Packet_SOP_LAST**

struct **BS_PD_Packet**

[*BS_PD_Packet*](#) structure - Contains information describing the contained Power Delivery Packet

Public Members

`uint8_t channel`

PD Channel this packet refers too.

`uint32_t seconds`

Seconds since device boot

`uint32_t uSeconds`

Micro seconds since device boot

`BS_PD_Packet_Direction_t direction`

PD Communication direction

`BS_PD_Packet_SOP_t sop`

PD SOP type

`std::vector<uint8_t> payload`

Raw PD packet

`uint32_t event`

PD Event type - defined in aProtocolDefs.h

`BS_PD_Packet_CC_Channel_t ccChannel`

CC Channel type

`uint32_t crc`

CRC of payload

3.5 RESTful API Reference

Welcome to the BrainStem RESTful API reference documentation. This interface provides access to the BrainStem protocol over HTTP as well as additional information about system versions, attached USB devices, and bulk data transfers.

Use of the RESTful API requires either *BrainD*, or one of the applications built on BrainD, such as *ControlRoom*.

Receiving information from the RESTful interface occurs through an HTTP GET request. Sending or setting information occurs through a PUT interface.

The current version of the RESTful interface is v1. Changes that do not break a RESTful API will not constitute a version change. Removing features or making significant changes will result in a version change. Deploying breaking changes may be necessary for legal, performance, or security reasons. When a new version is released, the current version will remain supported for one year and may become unavailable anytime after this period.

3.5.1 API Version v1

- GET `http://<IPADDRESS>:<PORT>/api/v1/<ENDPOINT>/<PARAMETERS>`
- PUT `http://<IPADDRESS>:<PORT>/api/v1/<ENDPOINT>/<PARAMETERS> {BODY}`

Table 19: Endpoint Structure Definition

Parameter	Request Type	Description
IPADDRESS	GET, PUT	IP Address of BrainD server hosting RESTful endpoints.
PORT	GET, PUT	Port that BrainD server is listening for connections. Default is 9005.
ENDPOINT	GET, PUT	RESTful API endpoint to access.
PARAMETERS	GET, PUT	Command parameters for the given ENDPOINT.
BODY	PUT	HTTP PUT payload for the given ENDPOINT and PARAMETERS.

The available options for the `ENDPOINT` parameter are listed below:

Table 20: Available Endpoints

Service	Description
<i>acronameDevices</i>	Enumeration of attached Acroname devices
<i>acronameDevicesState</i>	Bulk Endpoint containing telemetry about all attached Acroname devices
<i>brainstem</i>	Raw access to BrainStem API for a device
<i>devices</i>	List of all USB devices connected to the host computer
<i>version</i>	Software Versions

The response body for all RESTful API calls will be encapsulated in a *transaction envelope*. This envelope will provide error information, request context, and other useful fields about the request and the response payload for the specific API call.

JSON Schema Version 1

acronameDevices Endpoint

The `acronameDevices` endpoint enumerates all of the attached Acroname devices on the host system, and provides information about their connection parameters, software licenses, and module information.

There are no input parameters to this endpoint. It will return a JSON array of devices, or error, encapsulated in a *transaction envelope*.

JSON Schema for Response Object

The following tables list a [JSON Schema](#)¹¹⁸ for the response object.

Download the raw JSON Schema file for this response: `acroname_devices_response.json`

Object Fields of Acroname Devices Response

Name	Type	Mandatory	Description
<code>acronameDevices</code>	Array< <i>Object</i> >	Yes	An array containing all Acroname devices in the system.

Object Fields of `acronameDevices[]`

Name	Type	Mandatory	Description
<code>type</code>	<i>v1-type</i>	Yes	Type of connection to device
<code>serialNumber</code>	<i>v1-serialNumber</i>	Yes	Acroname Serial Number
<code>module</code>	<i>v1-uint32</i>	Yes	Brainstem Module address of this device
<code>router</code>	<i>v1-uint32</i>	No	BrainStem network router address
<code>routerSerialNumber</code>	<i>v1-serialNumber</i>	No	BrainStem network router serial number
<code>model</code>	<i>v1-model</i>	Yes	Model Number of device (see <code>aDefs.c</code> for allowed values)
<code>entitlements</code>	Array< <i>Object</i> >	No	An array containing all entitlements for this specific device

¹¹⁸ <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

Object Fields of `acronameDevices[] .entitlements[]`

Name	Type	Mandatory	Description
tag	<i>v1-tag</i>	Yes	The entitlement tag associated with this entitlement
version	<i>v1-uint32</i>	Yes	Version field of the specified entitlement

v1-type

Name	Value
Type	string
description	Enumeration of all possible USB speeds
Enumerations	INVALID, USB, TCP/IP, SERIAL, NETWORK

v1-uint32

Name	Value
Type	integer
description	Unsigned 32-bit integer
minimum	0
maximum	4294967295

v1-serialNumber

Name	Value
Type	string
description	Representation of an Acroname device serial number
Regular Expression	<code>^(0x)?[A-Fa-f0-9]{8}\$</code>

v1-tag

Name	Value
Type	string
description	Representation of the entitlement tag

v1-model

Name	Value
Type	string
de- scrip- tion	Enumeration of all supported Acroname devices.
Enu- mer- a- tions	USBStem, EtherStem, MTMIOSerial, MTMPM1, MTMEtherStem, MTMUSBStem, USBHub2x4, MTMRelay, USBHub3p, MTMDAQ1, USBCSwitch, USBCSwitchPro, MTMDAQ2, MTMLoad1, USBHub3c, USBExt3c, PD3M, Unknown

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevices
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/acronameDevices')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
1 {
2   "timestamp": "2023-09-20T17:11:54.092Z",
3   "request": {
4     "endpointName": "/api/v1/acronameDevices",
5     "parameters": {}
6   },
7   "response": {
8     "acronameDevices": [
9       {
10        "type": "USB",
11        "serialNumber": "3C43352C",
12        "module": 6,
13        "model": "USBHub3c",
14        "entitlements": [
15          {
16            "tag": "CONTROL",
17            "version": 0
18          }
19        ]
20      },
21      {
22        "type": "USB",
23        "serialNumber": "F7D9AFB6",
24        "module": 6,
```

(continues on next page)

(continued from previous page)

```
25         "model": "USBHub3p",
26         "entitlements": [
27             {
28                 "tag": "CONTROL",
29                 "version": 0
30             }
31         ]
32     }
33 ]
34 }
35 }
```

acronameDevicesState Endpoint

The `acronameDevicesState` endpoint provides telemetry and state information for all of the attached Acroname devices on the host system. It provides USB connection information, custom names, and device version information for each port on the device.

There are no input parameters to this endpoint. It will return a JSON array of devices, or error, encapsulated in a *transaction envelope*.

JSON Schema for Response Object

The following table lists a [JSON Schema](#)¹¹⁹ for the response object.

Download the raw JSON Schema file for this response: `acroname_devices_state_response.json`

Object Fields of Acroname Devices State Response

Name	Type	Mandatory	Description
timestamp	<i>v1-timestamp</i>	Yes	Time and date that this object was created.
sequence	<i>v1-uint32</i>	Yes	Sequence counter that increments on each message sent.
brainstemVersion	<i>v1-version</i>	Yes	Version of the BrainStem library for this object.
acronameDevices	Array< <i>Object</i> >	Yes	Array containing all the available Acroname devices on the system.

¹¹⁹ <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

Object Fields of `acronameDevices []`

Name	Type	Mandatory	Description
<code>serialNumber</code>	<i>v1-serialNumber</i>	Yes	Acroname Serial Number
<code>firmwareVersion</code>	<i>v1-version</i>	Yes	Firmware version that is running on the device.
<code>model</code>	<i>v1-model</i>	Yes	Model Name of the device.
<code>customName</code>	string	No	If set, a user-defined name for the device.
<code>upstreamPort</code>	<i>v1-uint8</i>	Yes	Currently active upstream port index.
<code>upstreamPortSelection-IsAutomatic</code>	boolean	Yes	Is upstream port selected automatically.
<code>ports</code>	Array< <i>Object</i> >	No	Object representing a single port on a device.
<code>powerLimit</code>	<i>v1-portParam</i>	No	Power limit for the device.
<code>powerLimitMax</code>	<i>v1-portParam</i>	No	Maximum power limit for the device.
<code>inputVoltage</code>	<i>v1-portParam</i>	No	Input voltage to the device.
<code>inputCurrent</code>	<i>v1-portParam</i>	No	Input current to the device.
<code>inputVoltageUnregulated</code>	<i>v1-portParam</i>	No	Unregulated input voltage to the device.
<code>inputCurrentUnregulated</code>	<i>v1-portParam</i>	No	Unregulated input current to the device.
<code>inputPowerSource</code>	string	No	Source of input power for the device.
<code>hdbaset</code>	Array< <i>v1-hdbasetInfo</i> >	No	Array of HDBaseT information for the device.
<code>poe</code>	Array< <i>v1-poeInfo</i> >	No	Array of POE information for the device.

Object Fields of `acronameDevices[] .ports[]`

Name	Type	Mandatory	Description
<code>index</code>	<i>v1-uint8</i>	Yes	Physical Port Number on device. This may include upstream ports, control ports, etc.
<code>physicalName</code>	string	Yes	Description of the port type and number.
<code>customName</code>	string	No	If set, a user-defined name for the port.
<code>speed</code>	string	No	Current data speed of the port.
<code>roles</code>	<i>Array<v1-dataRole></i>	Yes	All supported data roles of the port.
<code>currentData-Role</code>	<i>v1-dataRole</i>	Yes	Current data role of the port.
<code>enabled</code>	boolean	No	Whether or not the port is enabled.
<code>voltage</code>	<i>v1-portParam</i>	No	Voltage measurement from this port.
<code>current</code>	<i>v1-portParam</i>	No	Current measurement from this port.
<code>cableInfo</code>	<i>v1-cableInfo</i>	No	If set, the available cable information
<code>allocated-Power</code>	<i>v1-portParam</i>	No	Power allocated to this port.
<code>available-Power</code>	<i>v1-portParam</i>	No	Power available to this port.
<code>powerLimit</code>	<i>v1-portParam</i>	No	Power limit for this port.
<code>vbusAccumulatedPower</code>	<i>v1-portParam</i>	No	Accumulated VBUS power consumption for this port.
<code>vconnAccumulatedPower</code>	<i>v1-portParam</i>	No	Accumulated VCONN power consumption for this port.

v1-timestamp

Name	Value
Type	string
description	String containing a ISO8601 timestamp (YYYY-MM-DDTHH:MM:SS.mmmZ
Regular Expression	<code>^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{3,6}Z\$</code>

v1-serialNumber

Name	Value
Type	string
description	Hexadecimal representation of an Acroname device.
Regular Expression	<code>^(0x)?[A-Fa-f0-9]{8}\$</code>

v1-version

Name	Value
Type	<i>Object</i>
description	Object representing a <major>.<minor>.<patch> semantic version.

Name	Type	Mandatory	Description
major	<i>v1-uint8</i>	Yes	
minor	<i>v1-uint8</i>	Yes	
patch	<i>v1-uint8</i>	Yes	

v1-model

Name	Value
Type	string
de- scrip- tion	Enumeration of all supported Acroname devices.
Enu- mer- a- tions	USBStem, EtherStem, MTMIOSerial, MTMPM1, MTMEtherStem, MTMUSBStem, USBHub2x4, MTMRelay, USBHub3p, MTMDAQ1, USBCSwitch, USBCSwitchPro, MTMDAQ2, MTMLoad1, USBHub3c, USBExt3c, PD3M, Unknown

v1-dataRole

Name	Value
Type	string
description	Enumeration of all supported data roles for a port.
Enumerations	Disabled, Upstream, Downstream, Control, Unknown

v1-portParam

Name	Value
Type	<i>Object</i>
description	Object containing a data parameter with units and scale values.

Name	Type	Mandatory	Description
rawValue	number	Yes	Unformatted return value from the device.
value	number	Yes	If appropriate, a user-friendly formatted value.
units	<i>v1-units</i>	Yes	If appropriate, a user-friendly unit string.

v1-cableInfo

Name	Value
Type	<i>Object</i>
description	Object containing available cable information

Name	Type	Mandatory	Description
orientation	<i>v1-cableOrientation</i>	No	If available, the current cable orientation
maxSpeed	<i>v1-cableMaxSpeed</i>	No	If available, the maximum speed the cable is capable of
type	<i>v1-cableType</i>	No	If available, the type of cable
maxVoltage	<i>v1-cableMaxVoltage</i>	No	If available, the maximum voltage the cable is capable of
maxCurrent	<i>v1-cableMaxCurrent</i>	No	If available, the maximum current the cable is capable of

v1-cableOrientation

Name	Value
Type	string
description	Enumeration of all possible cable orientations
Enumerations	Unknown, CC1, CC2

v1-cableMaxSpeed

Name	Value
Type	string
description	Enumeration of all possible cable speeds
Enumerations	Unknown, USB 2.0, USB 3.2 Gen 1, USB 3.2 / USB 4 Gen 2, USB 4.0 Gen 3, USB 4.0 Gen 4

v1-cableMaxVoltage

Name	Value
Type	string
description	Enumeration of all possible cable voltage limitations
Enumerations	Unknown, 20V, 30V, 40V, 50V

v1-cableMaxCurrent

Name	Value
Type	string
description	Enumeration of all possible cable current limitations
Enumerations	Unknown, 3A, 5A

v1-cableType

Name	Value
Type	string
description	Enumeration of all possible cable types
Enumerations	Unknown, Passive, Active

v1-uint8

Name	Value
Type	integer
description	Unsigned 8-bit integer
minimum	0
maximum	255

v1-uint32

Name	Value
Type	integer
description	Unsigned 32-bit integer
minimum	0
maximum	4294967295

v1-units

Name	Value
Type	string
description	Allowed and recognized units for a value.
Enumerations	volts, millivolts, microvolts, amperes, milliamperes, microamperes, coulombs, millicoulombs, microcoulombs, seconds, milliseconds, microseconds, nanoseconds, kilowatts, watts, milliwatts, microwatts, kilowatt-hours, watt-hours, milliwatt-hours, microwatt-hours

v1-aErr

Nam	Value
Type	string
de- scrip tion	Enumeration of all possible error results
Enu- mer- a- tions	aErrNone, aErrMemory, aErrParam, aErrNotFound, aErrFileNameLength, aErrBusy, aErrIO, aErrMode, aErrWrite, aErrRead, aErrEOF, aErrNotReady, aErrPermission, aErrRange, aErrSize, aErrOverrun, aErrParse, aErrConfiguration, aErrTime-out, aErrInitialization, aErrVersion, aErrUnimplemented, aErrDuplicate, aErrCancel, aErrPacket, aErrConnection, aErrIndexRange, aErrShortCommand, aErrInvalidEntity, aErrInvalidOption, aErrResource, aErrMedia, aErrAsyncReturn, aErrOperation, aErrUnknown

v1-error

Name	Value
Type	<i>Object</i>
description	Error containing an Acroname error

Name	Type	Mandatory	Description
errorCode	<i>v1-aErr</i>	Yes	Acroname error code
errorMes- sage	string	No	A human-readable message to describe the context of the error.

v1-hdbasetInfo

Name	Value
Type	<i>Object</i>
description	Object containing HDBaseT information

Name	Type	Mandatory	Description
index	<i>v1-uint8</i>	Yes	HDBaseT interface index
serialNumber	string	No	HDBaseT device serial number
firmwareVersion	string	No	HDBaseT device firmware version
state	Array< <i>v1-hdbasetState</i> >	No	Array of HDBaseT state flags
cableLength	<i>v1-portParam</i>	No	Cable length in meters
mseA	<i>v1-portParam</i>	No	Mean Square Error for channel A in decibels
mseB	<i>v1-portParam</i>	No	Mean Square Error for channel B in decibels
retransmission-Rate	string	No	Retransmission rate as a string
encodingState	<i>v1-hdbasetEncodingState</i>	No	Current encoding state
linkUtilization	<i>v1-portParam</i>	No	Link utilization as a percentage

v1-hdbasetState

Name	Value
Type	string
description	Enumeration of HDBaseT state flags
Enumerations	Device Present, Link Up

v1-hdbasetEncodingState

Name	Value
Type	string
description	Enumeration of HDBaseT encoding states
Enumerations	Unknown, PAM-16, PAM-8, PAM-4

v1-poelInfo

Name	Value
Type	Object
description	Object containing POE information

Name	Type	Mandatory	Description
voltage	<i>v1-portParam</i>	No	POE voltage
current	<i>v1-portParam</i>	No	POE current
resistance	<i>v1-portParam</i>	No	POE resistance
mode	<i>v1-poePowerMode</i>	No	POE power mode
state	<i>v1-poePowerMode</i>	No	POE power state
pair12DetectionStatus	<i>v1-poeDetectionStatus</i>	No	Detection status for pairs 1/2
pair34DetectionStatus	<i>v1-poeDetectionStatus</i>	No	Detection status for pairs 3/4
pair12SourcingClass	<i>v1-poeClassification</i>	No	Sourcing class for pairs 1/2
pair34SourcingClass	<i>v1-poeClassification</i>	No	Sourcing class for pairs 3/4
pair12DiscoveredClass	<i>v1-poeClassification</i>	No	Discovered class for pairs 1/2
pair34DiscoveredClass	<i>v1-poeClassification</i>	No	Discovered class for pairs 3/4
pair12RequestedClass	<i>v1-poeClassification</i>	No	Requested class for pairs 1/2
pair34RequestedClass	<i>v1-poeClassification</i>	No	Requested class for pairs 3/4

v1-poePowerMode

Name	Value
Type	string
description	Enumeration of POE power modes
Enumerations	Off, PSE, PD, Auto

v1-poeDetectionStatus

Name	Value
Type	string
description	Enumeration of POE detection statuses
Enumerations	Unknown, Short Circuit, Open Circuit, Low Resistance, High Resistance, Valid, Switch Failure

v1-poeClassification

Name	Value
Type	string
description	Enumeration of POE classifications
Enumerations	Unknown, Class 0, Class 1, Class 2, Class 3, Class 4, Class 4+, Class 5 4P SS, Class 5 4P DS, Class 6, Class 7, Class 8, Mismatch, OverCurrent

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevicesState
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/acronameDevicesState')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
1 {
2     "timestamp": "2023-09-20T17:24:24.428Z",
3     "request": {
4         "endpointName": "/api/v1/acronameDevicesState",
5         "parameters": {}
6     },
7     "response": {
8         "timestamp": "2023-09-20T17:24:24.108Z",
9         "sequence": 2563,
10        "brainstemVersion": {
11            "major": 2,
12            "minor": 10,
13            "patch": 0
14        },
15        "acronameDevices": [
16            {
17                "serialNumber": "0x3C43352C",
18                "firmwareVersion": {
19                    "major": 2,
20                    "minor": 10,
21                    "patch": 0
22                },
23                "model": "USBHub3c",
24                "upstreamPort": 0,
25                "upstreamPortSelectionIsAutomatic": true,
26                "ports": [
27                    {
28                        "index": 0,
29                        "physicalName": "Port 0",
30                        "speed": "5 Gbps",
31                        "roles": [
32                            "Upstream",
33                            "Downstream"
34                        ],
35                        "currentDataRole": "Upstream",
36                        "enabled": true,
37                        "voltage": {
38                            "value": 5.207519,
39                            "units": "volts",
40                            "rawValue": 5207519
41                        }
42                    }
43                ]
44            }
45        ]
46    }
47 }
```

(continues on next page)

(continued from previous page)

```

42         "current": {
43             "value": 0.00061,
44             "units": "amperes",
45             "rawValue": 610
46         }
47     },
48     {
49         "index": 1,
50         "physicalName": "Port 1",
51         "speed": "Unknown",
52         "roles": [
53             "Upstream",
54             "Downstream"
55         ],
56         "currentDataRole": "Downstream",
57         "enabled": true,
58         "voltage": {
59             "value": 0.0,
60             "units": "volts",
61             "rawValue": 0
62         },
63         "current": {
64             "value": 0.000305,
65             "units": "amperes",
66             "rawValue": 305
67         }
68     },
69     <additional ports ...>
70 ]
71 }
72 ]
73 }
74 }

```

devices Endpoint

The `devices` endpoint provides a list of all USB devices attached to the host system, including device information, port path, capabilities, and mappings to Acroname devices.

There are no input parameters to this endpoint. It will return a JSON array of devices, encapsulated in a *transaction envelope*.

JSON Schema for Response Object

The following table lists a [JSON Schema](#)¹²⁰ for the response object.

Download the raw JSON Schema file for this response: `devices_response.json`

¹²⁰ <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

Object Fields of Devices Response

Name	Type	Mandatory	Description
usb	Array< <i>Object</i> >	Yes	An array containing all USB devices in the system

Object Fields of `usb []`

Name	Type	Mandatory	Description
vendorId	<i>v1-vidPid</i>	Yes	Vendor Identifier
productId	<i>v1-vidPid</i>	Yes	Product Identifier
speedActual	<i>v1-speed</i>	Yes	Current enumerated speed
maxSpeed	<i>v1-speed</i>	Yes	Maximum supported speed
serialNumber	<i>v1-name</i>	Yes	Vendor-provided serial number
manufacturer	<i>v1-name</i>	Yes	Vendor-provided manufacturer name
productName	<i>v1-name</i>	Yes	Vendor-provided product name
deviceClass	<i>v1-classCode</i>	Yes	Device Class Code
interfaces	Array< <i>Object</i> >	Yes	List of all interfaces provided by device
portPath	Array< <i>v1-uint8</i> >	Yes	List of the logical IDs of each parent node, starting with the controller ID and followed by the port ID for each parent device.
acronameDevice	<i>Object</i>	No	If an Acroname device is detected in this device's parent tree, information about the Acroname device.
bulkCapability	<i>v1-capability</i>	Yes	Capability information for Bulk Transfers
isochronousCapability	<i>v1-capability</i>	Yes	Capability information for Isochronous Transfers
interruptCapability	<i>v1-capability</i>	Yes	Capability information for Interrupt Transfers

Object Fields of `usb[] . interfaces[]`

Name	Type	Mandatory	Description
interfaceIndex	<i>v1-uint8</i>	Yes	Interface Index within this device
entryIndex	<i>v1-uint8</i>	Yes	Alternate Setting Index within this interface
kClass	<i>v1-classCode</i>	Yes	USB-IF class code for this interface
kSubclass	<i>v1-classCode</i>	Yes	USB-IF subclass code for this interface, qualified by the kClass code
kProtocol	<i>v1-uint8</i>	Yes	USB-IF protocol code for this interface, qualified by the kClass and kSubclass codes

Object Fields of `usb[] . acronameDevice`

Name	Type	Mandatory	Description
serialNumber	<i>v1-serialNumber</i>	Yes	Acroname Serial Number
port	<i>v1-uint8</i>	Yes	Physical port number on the Acroname device
distance	<i>v1-uint8</i>	Yes	Number of parent tiers between Acroname device and this device

v1-vidPid

Name	Value
Type	string
description	Representation of a USB Device or Vendor ID
Regular Expression	<code>^(0x)?[A-Fa-f0-9]{4}\$</code>

v1-serialNumber

Name	Value
Type	string
description	Representation of an Acroname device serial number
Regular Expression	<code>^(0x)?[A-Fa-f0-9]{8}\$</code>

v1-speed

Name	Value
Type	string
description	Enumeration of all possible USB speeds
Enumerations	Unknown, 1.5 Mbps, 12 Mbps, 480 Mbps, 5 Gbps, 10 Gbps

v1-name

Name	Value
Type	string
description	USB-IF Compliant Name Field
Maximum # of Glyphs	126

v1-classCode

Name	Value
Type	integer
description	Enumeration of all possible USB device/interface classes (https://vovkos.github.io/doxyrest/samples/libusb/enum_libusb_class_code.html)
Enumerations	0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 220, 224, 254, 255

v1-uint8

Name	Value
Type	integer
description	Unsigned 8-bit integer
minimum	0
maximum	255

v1-capability

Name	Value
Type	<i>Object</i>
description	Description for a capability of a device

Name	Type	Mandatory	Description
isDoing	boolean	Yes	Whether the device is currently using this capability
isCapable	boolean	Yes	Whether the device is capable of using this capability

v1-aErr

Nam	Value
Type	string
de- scrip- tion	Enumeration of all possible error results
Enu- mer- a- tions	aErrNone, aErrMemory, aErrParam, aErrNotFound, aErrFileNameLength, aErrBusy, aErrIO, aErrMode, aErrWrite, aErrRead, aErrEOF, aErrNotReady, aErrPermission, aErrRange, aErrSize, aErrOverrun, aErrParse, aErrConfiguration, aErrTime-out, aErrInitialization, aErrVersion, aErrUnimplemented, aErrDuplicate, aErrCancel, aErrPacket, aErrConnection, aErrIndexRange, aErrShortCommand, aErrInvalidEntity, aErrInvalidOption, aErrResource, aErrMedia, aErrAsyncReturn, aErrOperation, aErrUnknown

v1-error

Name	Value
Type	<i>Object</i>
description	Error containing an Acrname error

Name	Type	Mandatory	Description
errorCode	<i>v1-aErr</i>	Yes	Acrname error code
errorMes- sage	string	No	A human-readable message to describe the context of the error.

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/devices
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/devices')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
1 {
2     "timestamp": "2023-09-20T17:36:15.359Z",
3     "request": {
4         "endpointName": "/api/v1/devices",
5         "parameters": {}
6     },
7     "response": {
8         "usb": [
9             {
10                 "vendorId": "0x067B",
11                 "productId": "0x2303",
12                 "speedActual": "12 Mbps",
13                 "maxSpeed": "12 Mbps",
14                 "productName": "USB-Serial Controller",
15                 "serialNumber": "",
16                 "manufacturer": "Prolific Technology Inc.",
17                 "deviceClass": 0,
18                 "interfaces": [
19                     {
20                         "interfaceIndex": 0,
21                         "entryIndex": 0,
22                         "kClass": 255,
23                         "kSubclass": 0,
24                         "kProtocol": 0
25                     }
26                 ],
27                 "portPath": [
28                     4,
29                     3,
30                     1
31                 ],
32                 "acronameDevice": {
33                     "serialNumber": "0xF7D9AFB6",
34                     "port": 0,
35                     "distance": 0
36                 },
37                 "bulkCapability": {
38                     "isDoing": true,
39                     "isCapable": true
40                 },
41                 "isochronousCapability": {
```

(continues on next page)

(continued from previous page)

```

42         "isDoing": false,
43         "isCapable": false
44     },
45     "interruptCapability": {
46         "isDoing": true,
47         "isCapable": true
48     }
49 },
50 {
51     "vendorId": "0x2188",
52     "productId": "0x0747",
53     "speedActual": "5 Gbps",
54     "maxSpeed": "5 Gbps",
55     "productName": "Card Reader ",
56     "serialNumber": "000000000010",
57     "manufacturer": "CalDigit",
58     "deviceClass": 0,
59     "interfaces": [
60         {
61             "interfaceIndex": 0,
62             "entryIndex": 0,
63             "kClass": 8,
64             "kSubclass": 6,
65             "kProtocol": 80
66         }
67     ],
68     "portPath": [
69         4,
70         8
71     ],
72     "bulkCapability": {
73         "isDoing": true,
74         "isCapable": true
75     },
76     "isochronousCapability": {
77         "isDoing": false,
78         "isCapable": false
79     },
80     "interruptCapability": {
81         "isDoing": false,
82         "isCapable": false
83     }
84 }
85 ]
86 }
87 }

```

brainstem Endpoint

This reference builds on understanding of the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation:

- [BrainStem Overview](#)
- [BrainStem Terminology](#)
- [Getting Started with the BrainStem.](#)

BrainStem GET commands are implemented with HTTP `GET` calls, and SET commands are implemented with HTTP `PUT` calls. The entity and operation fields are mapped directly to BrainStem API calls, minus the `get` and `set` prefixes on the operations.

The parameters for the command are given as slash-separated path entries:

- GET `http://<IPADDRESS>:<PORT>/api/v1/brainstem/<SERIALNUM>/<ENTITY>/<INDEX>/<COMMAND>`
- GET `http://<IPADDRESS>:<PORT>/api/v1/brainstem/<SERIALNUM>/<ENTITY>/<INDEX>/<COMMAND> {BODY}`

Table 21: BrainStem Request Parameters

Parameter	Request Type	Description
IPAD-DRESS	GET, PUT	IP Address of BrainD server hosting RESTful endpoints.
PORT	GET, PUT	Port that BrainD server is listening for connections. Default is 9005.
SERIAL-NUM	GET, PUT	Serial Number of the BrainStem device being accessed
ENTITY	GET, PUT	BrainStem Entity , e.g. system, temperature, digital.
INDEX	GET, PUT	Entity Index
COM-MAND	GET, PUT	Operation to perform on entity without <code>get</code> or <code>set</code> prefix, e.g. inputvoltage, currentlimit, name.
BODY	PUT	HTTP <code>PUT</code> payload, matching PUT Request JSON Schema

Note: The following BrainStem Entities are unsupported at this time: `app`, `i2c`, `powerDelivery`

Analog Endpoint

Interface to analog entities on BrainStem modules.

Analog entities may be configured as a input or output depending on hardware capabilities.

Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value.

The resolution of the voltage or number of useful bits is also hardware dependent.

See the [Analog Entity](#) for generic information.

GET `/api/v1/brainstem/ (serial_num) /analog/
index/value`

Get the raw ADC output value in bits.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/voltage

Get the scaled micro volt value with reference to ground.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/range

Get the analog input range.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/enable

Get the analog output enable status.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /analog/
index/value

Set the value of an analog output (DAC) in bits.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 16-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /analog/
index/voltage

Set the voltage level of an analog output (DAC) in microvolts.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /analog/
index/range

Set the analog input range.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /analog/
index/enable

Set the analog output enable state.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.

May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /analog/
index/configuration

Set the analog configuration.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/configuration

Get the analog configuration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /analog/
index/bulkcapturesamplerate

Set the sample rate for this analog when bulk capturing.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/bulkcapturesamplerate

Get the current sample rate setting for this analog when bulk capturing.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /analog/
index/bulkcapturenumberofsamples

Set the number of samples to capture for this analog when bulk capturing.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/bulkcapturenumberofsamples

Get the current number of samples setting for this analog when bulk capturing.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/bulkcapture

Initiate a BulkCapture on this analog.

Captured measurements are stored in the module’s RAM store (RAM_STORE) slot 0.

Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

GET /api/v1/brainstem/ (*serial_num*) /analog/
index/bulkcapturestate

Get the current bulk capture state for this analog.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

App Endpoint

Used to send a cmdAPP packet to the BrainStem network.

These commands are used for either host-to-stem or stem-to-stem interactions.

BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

See the [App Entity](#) for generic information.

PUT /api/v1/brainstem/ (*serial_num*) /app/
index/execute

Execute the app reflex on the module.

Doesn't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Clock Endpoint

Provides an interface to a real-time clock entity on a BrainStem module.

The clock entity may be used to get and set the real time of the system.

The clock entity has a one second resolution.

See the [Clock Entity](#) for generic information.

GET /api/v1/brainstem/ (*serial_num*) /clock/
index/year

Get the four digit year value (0-4095).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /clock/
index/year

Set the four digit year value (0-4095).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 16-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /clock/
index/month

Get the two digit month value (1-12).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /clock/
index/month

Set the two digit month value (1-12).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /clock/
index/day

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /clock/
index/day

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /clock/
index/hour

Get the two digit hour value (0-23).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /clock/
index/hour

Set the two digit hour value (0-23).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /clock/
index/minute

Get the two digit minute value (0-59).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /clock/
index/minute

Set the two digit minute value (0-59).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /clock/
index/second

Get the two digit second value (0-59).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /clock/
index/second

Set the two digit second value (0-59).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Digital Endpoint

Interface to digital entities on BrainStem modules.

Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ.

Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

See the [Digital Entity](#) for generic information.

PUT /api/v1/brainstem/ (*serial_num*) /digital/
index/configuration

Set the digital configuration to one of the available 5 states.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /digital/
index/configuration

Get the digital configuration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /digital/
index/state

Set the logical state.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /digital/
index/state

Get the state.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /digital/
index/stateall

Sets the logical state of all available digitals based on the bit mapping.

Number of digitals varies across BrainStem modules.

Refer to the datasheet for the capabilities of your module.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /digital/
index/stateall

Gets the logical state of all available digitals in a bit mapped representation.

Number of digitals varies across BrainStem modules.

Refer to the datasheet for the capabilities of your module.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /digital/
index/value

Set the expected value of the digital pin.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /digital/
index/value

Get the expected value of the digital pin.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /digital/
index/linkchannel

Set the link channel (entity index) for linking digital entities.

Two digital entities will link when one is configured as LinkInput, one as LinkOutput, and both have each others linkChannel indexes.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /digital/
index/linkchannel

Get the link channel (entity index) for linking digital entities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Equalizer Endpoint

Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname’s products.

Please see product documentation for further details.

See the [Equalizer Entity](#) for generic information.

PUT /api/v1/brainstem/ (*serial_num*) /equalizer/
index/receiverconfig/channel

Sets the receiver configuration for a given channel.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The equalizer receiver channel.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /equalizer/
index/receiverconfig/channel

Gets the receiver configuration for a given channel.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The equalizer receiver channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /equalizer/
index/transmitterconfig

Sets the transmitter configuration

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /equalizer/
index/transmitterconfig

Gets the transmitter configuration

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Ethernet Endpoint

IP configuration. MAC info. BrainD port.

See the [Ethernet Entity](#) for generic information.

PUT `/api/v1/brainstem/ (serial_num) /ethernet/
index/enabled`

Sets the Ethernet's interface to enabled/disabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as "True", "FALSE", "1", etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /ethernet/
index/enabled`

Gets the current enable value of the Ethernet interface.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /ethernet/
index/networkconfiguration`

Get the method in which IP Address is assigned to this device

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /ethernet/
index/networkconfiguration

Get the method in which IP Address is assigned to this device

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticip4address

Get the expected IPv4 address of this device, when networkConfiguration == STATIC

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticip4address

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 8-bit values to set.
May be specified as: * String (formatted as “Hello”, “x40x41”, etc.) * List of integers *
List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticip4netmask

Get the expected IPv4 netmask of this device, when networkConfiguration == STATIC

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticip4netmask

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 8-bit values to set.
May be specified as: * String (formatted as “Hello”, “x40x41”, etc.) * List of integers *
List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticip4gateway

Get the expected IPv4 gateway of this device, when networkConfiguration == STATIC

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string

- **response.rawValue** (*list (integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticipv4gateway

Set the desired IPv4 gateway of this device, if NetworkConfiguration == STATIC

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 8-bit values to set.

May be specified as: * String (formatted as “Hello”, “x40x41”, etc.) * List of integers *
List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/ipv4address

Get the effective IP address of this device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/ipv4netmask

Get the effective IP netmask of this device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/ipv4gateway

Get the effective IP gateway of this device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticipv4dnsaddress

Set IPv4 DNS Addresses (plural), if NetworkConfiguration == STATIC

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 8-bit values to set.
May be specified as: * String (formatted as “Hello”, “x40x41”, etc.) * List of integers *
List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/staticipv4dnsaddress

Get IPv4 DNS addresses (plural), when NetworkConfiguration == STATIC

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/ipv4dnsaddress

Get effective IPv4 DNS addresses, for the current NetworkConfiguration

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /ethernet/
index/hostname

Set hostname that’s requested when this device sends a DHCP request.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 8-bit values to set.
May be specified as: * String (formatted as “Hello”, “x40x41”, etc.) * List of integers *
List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/hostname

Get hostname that’s requested when this device sends a DHCP request.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/macaddress

Get the MAC address of the Ethernet interface.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /ethernet/
index/interfaceport/service

Set the port of a TCPIP service on the device.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **service** – The index of the service to set the port for.

Request JSON Object

- **value** (*integer*) – 16-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /ethernet/
index/interfaceport/service

Get the port of a TCPIP service on the device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **service** – The index of the service to get the port for.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

HDBaseT Endpoint

This entity is only available on certain modules, and provides information on HDBaseT extenders.

See the [HDBaseT Entity](#) for generic information.

GET `/api/v1/brainstem/ (serial_num) /hdbaset/
index/serialnumber`

Gets the serial number of the HDBaseT device (6 bytes)

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /hdbaset/
index/firmwareversion`

Gets the firmware version of the HDBaseT device

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /hdbaset/
index/state`

Gets the current state of the HDBaseT link

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /hdbaset/
index/cablenght`

Gets the perceived cable length

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/msea

Gets the Mean Squared Error (MSE) for channel A

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/mseb

Gets the Mean Squared Error (MSE) for channel B

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/retransmissionrate

Gets the number of successful messages between retransmission

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/linkutilization

Gets the current link utilization

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/encodingstate

Gets the current encoding state.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/usb2devicetree

Gets the USB2 tree at the HDBaseT device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/usb3devicetree

Gets the USB3 tree at the HDBaseT device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/linkrole

Gets the current link role

In the case of “Auto” the getState API will provide the current role.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /hdbaset/
index/linkrole

Sets the active link role

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

I2C Endpoint

Interface the I2C buses on BrainStem modules.

The class provides a way to send read and write commands to I2C devices on the entities bus.

See the [I2C Entity](#) for generic information.

Mux Endpoint

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs.

An analogy would be the switchboard of a telephone operator.

Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs; some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

See the [Mux Entity](#) for generic information.

GET `/api/v1/brainstem/ (serial_num) /mux/
index/enable`

Get the mux enable/disable status

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /mux/
index/enable`

Enable the mux.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as "True", "FALSE", "1", etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /mux/
index/channel

Get the current selected mux channel.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /mux/
index/channel

Set the current mux channel.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /mux/
index/voltage/channel

Get the voltage of the indicated mux channel.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The channel in which voltage was requested.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /mux/
index/config

Get the configuration of the mux.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /mux/
index/config

Set the configuration of the mux.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /mux/
index/split

Get the current split mode mux configuration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /mux/
index/split

Sets the mux's split mode configuration.

See *JSON Schema for Request Object* and *Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PoE Endpoint

This entity is only available on certain modules, and provides a Power over Ethernet control ability.

See the *PoE Entity* for generic information.

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairenabled/pair

Gets the current enable value of the indicated POE pair.

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /poe/
index/pairenabled/pair

Enables or disables the indicated POE pair.

See *JSON Schema for Request Object* and *Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/powermode

Gets the power mode of the device

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /poe/
index/powermode

Sets the power mode of the device

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/powerstate

Gets the power state of the device

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairsourcingclass/pair

Gets the sourcing class for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /poe/
index/pairsourcingclass/pair

Sets the sourcing class for a given pair.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairrequestedclass/pair

Gets the requested class for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairdiscoveredclass/pair

Gets the discovered class for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairdetectionstatus/pair

Gets detected status of the POE connection for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairvoltage/pair

Gets the Voltage for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/paircurrent/pair

Gets the Current for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairresistance/pair

Gets the Resistance for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/paircapacitance/pair

Gets the Capacitance for a given pair

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairpower/pair

Get the instantaneous power consumption for a given pair

The equivalent of Voltage x Current

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/totalpower

Gets the total instantaneous power consumption

The equivalent of Pair1(Voltage x Current) + Pair2(Voltage x Current)

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/pairaccumulatedpower/pair

Gets the accumulated power for a given pair.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /poe/
index/pairaccumulatedpower/pair

Sets the accumulated power for a given pair.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /poe/
index/totalaccumulatedpower

Gets the total Accumulated Power

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /poe/
index/totalaccumulatedpower

Sets the total accumulated power

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
- May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Pointer Endpoint

Allows access to the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module.

The interface allows the use of the BrainStem scratchpad from the host, and provides a mechanism for allowing the host application and BrainStem relexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file.

The cursor position can be set via `setOffset`. A read of a character short or int can be made from that cursor position.

In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

See the [Pointer Entity](#) for generic information.

GET `/api/v1/brainstem/ (serial_num) /pointer/
index/offset`

Get the offset of the pointer

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /pointer/
index/offset`

Set the offset of the pointer

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 16-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /pointer/
index/mode`

Get the mode of the pointer

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /pointer/
index/mode

Set the mode of the pointer

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /pointer/
index/transferstore

Get the handle to the store.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /pointer/
index/transferstore

Set the handle to the store.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /pointer/
index/transferstore

Transfer data to the store.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /pointer/
index/transferstore

Transfer data from the store.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /pointer/
index/char

Get a char (1 byte) value from the pointer at this object’s index, where elements are 1 byte long.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /pointer/
index/char

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /pointer/
index/short

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /pointer/
index/short

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 16-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /pointer/
index/int

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /pointer/
index/int

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Port Endpoint

The Port Entity provides software control over the most basic items related to a USB Port.

This includes everything from the complete enable and disable of the entire port to the individual control of specific pins.

Voltage and Current measurements are also included for devices which support the Port Entity.

See the [Port Entity](#) for generic information.

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vbusvoltage

Gets the Vbus Voltage

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vbuscurrent

Gets the Vbus Current

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vconnvoltage

Gets the Vconn Voltage

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vconncurrent

Gets the Vconn Current

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/powermode

Gets the Port Power Mode: Convenience Function of get/setPortMode

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/powermode

Sets the Port Power Mode: Convenience Function of get/setPortMode

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/portenabled

Gets the current enable value of the port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/portenabled

Enables or disables the entire port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/dataenabled

Gets the current enable value of the data lines.

Sub-component (Data) of getEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/dataenabled

Enables or disables the data lines.

Sub-component (Data) of setEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datahsenabled

Gets the current enable value of the High Speed (HS) data lines.

Sub-component of getDataEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/datahsenabled

Enables or disables the High Speed (HS) data lines.

Sub-component of setDataEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as "True", "FALSE", "1", etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datahs1enabled

Gets the current enable value of the High Speed A side (HSA) data lines.

Sub-component of getDataHSEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/datahs1enabled

Enables or disables the High Speed A side (HSA) data lines.

Sub-component of setDataHSEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as "True", "FALSE", "1", etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datahs2enabled

Gets the current enable value of the High Speed B side (HSB) data lines.

Sub-component of getDataHSEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/datahs2enabled

Enables or disables the High Speed B side (HSB) data lines.

Sub-component of setDataHSEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datassenabled

Gets the current enable value of the Super Speed (SS) data lines.

Sub-component of getDataEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/datassenabled

Enables or disables the Super Speed (SS) data lines.

Sub-component of setDataEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datass1enabled

Gets the current enable value of the Super Speed A side (SSA) data lines.

Sub-component of getDataSSEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/datass1enabled

Enables or disables the Super Speed A side (SSA) data lines.

Sub-component of setDataEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datass2enabled

Gets the current enable value of the Super Speed B side (SSB) data lines.

Sub-component of getDataSSEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /port/
index/datass2enabled`

Enables or disables the Super Speed B side (SSB) data lines.

Sub-component of setDataSSEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /port/
index/powerenabled`

Gets the current enable value of the power lines. Sub-component (Power) of getEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /port/
index/powerenabled`

Enables or Disables the power lines. Sub-component (Power) of setEnable.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datarole

Gets the Port Data Role.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vconnenabled

Gets the current enable value of the Vconn lines.

Sub-component (Vconn) of getEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/vconnenabled

Enables or disables the Vconn lines.

Sub-component (Vconn) of setEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as "True", "FALSE", "1", etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vconn1enabled

Gets the current enable value of the Vconn1 lines.

Sub-component of getVconnEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/vconn1enabled

Enables or disables the Vconn1 lines.

Sub-component of setVconnEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as "True", "FALSE", "1", etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vconn2enabled

Gets the current enable value of the Vconn2 lines.

Sub-component of getVconnEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/vconn2enabled

Enables or disables the Vconn2 lines.

Sub-component of setVconnEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/ccenabled

Gets the current enable value of the CC lines.

Sub-component (CC) of getEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/ccenabled

Enables or disables the CC lines.

Sub-component (CC) of setEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc1enabled

Gets the current enable value of the CC1 lines.

Sub-component of getCCEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/cc1enabled

Enables or disables the CC1 lines.

Sub-component of setCCEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc2enabled

Gets the current enable value of the CC2 lines.

Sub-component of getCCEnabled.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/cc2enabled

Enables or disables the CC2 lines.

Sub-component of setCCEnabled.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/voltagesetpoint

Gets the current voltage setpoint value for the port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/voltagesetpoint

Sets the current voltage setpoint value for the port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/portstate

A bit mapped representation of the current state of the port.

Reflects what the port IS which may differ from what was requested.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/dataspeed

Gets the speed of the enumerated device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/portmode

Gets current mode of the port

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/portmode

Sets the mode of the port

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/errors

Returns any errors that are present on the port.

Calling this function will clear the current errors. If the error persists it will be set again.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/currentlimit

Gets the current limit of the port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/currentlimit

Sets the current limit of the port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/currentlimitmode

Gets the current limit mode.

The mode determines how the port will react to an over current condition.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/currentlimitmode

Sets the current limit mode.

The mode determines how the port will react to an over current condition.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/availablepower

Gets the current available power.

This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/allocatedpower

Gets the currently allocated power

This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/powerlimit

Gets the user defined power limit for the port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/powerlimit

Sets a user defined power limit for the port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/powerlimitmode

Gets the power limit mode.

The mode determines how the port will react to an over power condition.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/powerlimitmode

Sets the power limit mode.

The mode determines how the port will react to an over power condition.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/name

Gets a user defined name of the port.

Helpful for identifying ports/devices in a static environment.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/name

Sets a user defined name of the port.

Helpful for identifying ports/devices in a static environment.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 8-bit values to set.
May be specified as: * String (formatted as “Hello”, “x40x41”, etc.) * List of integers *
List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list (integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cccurrentlimit

Gets the CC Current Limit Resistance

The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/cccurrentlimit

Sets the CC Current Limit Resistance

The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datahsroutingbehavior

Gets the HighSpeed Data Routing Behavior.

The mode determines how the port will route the data lines.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/datahsroutingbehavior

Sets the HighSpeed Data Routing Behavior.

The mode determines how the port will route the data lines.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/datassroutingbehavior

Gets the SuperSpeed Data Routing Behavior.

The mode determines how the port will route the data lines.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/datassroutingbehavior

Sets the SuperSpeed Data Routing Behavior.

The mode determines how the port will route the data lines.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vbusaccumulatedpower

Gets the Vbus Accumulated Power

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/vbusaccumulatedpower

Sets the Vbus Accumulated Power

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/resetvbusaccumulatedpower

Resets the Vbus Accumulated Power to zero.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

GET /api/v1/brainstem/ (*serial_num*) /port/
index/vconnaccumulatedpower

Gets the Vconn Accumulated Power

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/vconnaccumulatedpower

Sets the Vconn Accumulated Power

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/resetvconnaccumulatedpower

Resets the Vconn Accumulated Power to zero.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/hsboost

Sets the ports USB 2.0 High Speed Boost Settings

The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/hsboost

Gets the ports USB 2.0 High Speed Boost Settings

The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc1state

Gets the current CC1 Strapping on local and remote

The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc2state

Gets the current CC2 Strapping on local and remote

The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/sbu1voltage

Get the voltage of SBU1 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/sbu2voltage

Get the voltage of SBU2 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc1voltage

Get the voltage of CC1 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc2voltage

Get the voltage of CC2 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc1current

Get the current through the CC1 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc2current

Get the current through the CC2 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc1accumulatedpower

Gets the CC1 Accumulated Power

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/cc1accumulatedpower

Sets the CC1 Accumulated Power

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /port/
index/cc2accumulatedpower

Gets the CC2 Accumulated Power

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /port/
index/cc2accumulatedpower

Sets the CC2 Accumulated Power

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PowerDelivery Endpoint

Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface.

This Entity will allow you to directly access the vast landscape of PD.

See the [PowerDelivery Entity](#) for generic information.

GET `/api/v1/brainstem/ (serial_num) /powerdelivery/
index/connectionstate`

Gets the current state of the connection in the form of an enumeration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /powerdelivery/
index/numberofpowerdataobjects/partner/powerRole`

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerdataobject/partner/powerRole/ruleIndex

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerdataobject/powerRole/ruleIndex

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/resetpowerdataobjecttodefault/powerRole/ruleIndex

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerdataobjectlist

Gets all Power Data Objects (PDOs).

Equivalent to calling PowerDeliveryClass::getPowerDataObject() on all partners, power roles, and index's.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /powerdelivery/
index/powerdataobjectenabled/powerRole/ruleIndex`

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index.

Enabled refers to whether the PDO will be advertised when a PD connection is made.

This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /powerdelivery/
index/powerdataobjectenabled/powerRole/ruleIndex`

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index.

Enabled refers to whether the PDO will be advertised when a PD connection is made.

This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerdataobjectenabledlist/powerRole

Gets all Power Data Object enables for a given power role.

Equivalent of calling PowerDeliveryClass::getPowerDataObjectEnabled() for all indexes.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/requestdataobject/partner

Gets the current Request Data Object (RDO) for a given partner.

RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero).

Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/requestdataobject

Sets the current Request Data Object (RDO) for a given partner.

Only the local partner can be changed.

RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero).

Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/linkstate

Gets the current state of the connection in the form of a bitmask.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/attachtimeelapsed

Gets the length of time that the port has been in the attached state.

Returned as a list of two unsigned integers, first seconds, then microseconds.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerrolecapabilities

Gets the power roles that may be advertised by the local partner. (CC Strapping).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerrole

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerrole

Set the current power role to be advertised by the Local partner. (CC Strapping).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerrolepreferred

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/powerrolepreferred

Set the preferred power role to be advertised by the Local partner (CC Strapping).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/datarolecapabilities

Gets the data roles that may be advertised by the local partner.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/cablevoltagegmax

Gets the maximum voltage capability reported by the e-mark of the attached cable.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/cablecurrentmax

Gets the maximum current capability report by the e-mark of the attached cable.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/cablespeedmax

Gets the maximum data rate capability reported by the e-mark of the attached cable.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/cabletype

Gets the cable type reported by the e-mark of the attached cable.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/cableorientation

Gets the current orientation being used for PD communication

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/requestcommand

Requests an action of the Remote partner.

Actions are not guaranteed to occur.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/requeststatus

Gets the status of the last request command sent.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/override

Gets the current enabled overrides

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/override

Sets the current enabled overrides

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/protocolrevisionmax

Gets the configured maximum USB Power Delivery protocol revision used as a negotiation ceiling (pdProtocolRevision2 = 2, pdProtocolRevision3 = 3). This is unrelated to hardware or product revision.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/protocolrevisionmax

Sets the maximum USB Power Delivery protocol revision for negotiation. Only pdProtocolRevision2 (2) and pdProtocolRevision3 (3) are valid API values.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/flagmode/flag

Gets the current mode of the local partner flag/advertisement.

These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices.

This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **flag** – Flag/Advertisement to be modified

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/flagmode/flag

Sets how the local partner flag/advertisement is managed.

These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices.

This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **flag** – Flag/Advertisement to be modified

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/peakcurrentconfiguration

Gets the Peak Current Configuration for the Local Source.

The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current.

This tolerance includes a maximum value and a time unit.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/peakcurrentconfiguration

Sets the Peak Current Configuration for the Local Source.

The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current.

This tolerance includes a maximum value and a time unit.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/fastroleswapcurrent

Gets the Fast Role Swap Current

The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /powerdelivery/
index/fastroleswapcurrent

Sets the Fast Role Swap Current

The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

RCServo Endpoint

Interface to servo entities on BrainStem modules.

Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs.

Please see the product datasheet on the configuration limitations.

See the [RCServo Entity](#) for generic information.

PUT /api/v1/brainstem/ (*serial_num*) /servo/
index/enable

Enable the servo channel

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.

May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /servo/
index/enable

Get the enable status of the servo channel.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /servo/
index/position

Set the position of the servo channel

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
- May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /servo/
index/position

Get the position of the servo channel

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /servo/
index/reverse

Set the output to be reversed on the servo channel

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /servo/
index/reverse

Get the reverse status of the servo channel

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

Rail Endpoint

Provides power rail functionality on certain modules.

The RailClass can be used to control power to downstream devices.

It has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

See the [Rail Entity](#) for generic information.

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/current

Get the rail current.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/currentsetpoint

Set the rail supply current. Rail current control capabilities vary between modules.

Refer to the module datasheet for definition of the rail current capabilities.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/currentsetpoint

Get the rail setpoint current. Rail current control capabilities vary between modules.

Refer to the module datasheet for definition of the rail current capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/currentlimit

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/currentlimit

Get the rail current limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/temperature

Get the rail temperature.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/enable

Get the state of the external rail switch. Not all rails can be switched on and off.

Refer to the module datasheet for capability specification of the rails.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/enable

Set the state of the external rail switch.

Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/voltage

Get the rail supply voltage.

Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/voltagesetpoint

Set the rail supply voltage.

Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/voltage/point

Get the rail setpoint voltage.

Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/voltage/limit

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/voltage~~min~~limit

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/voltage~~max~~limit

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/voltage~~max~~limit

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/power

Get the rail supply power.

Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/powersetpoint

Set the rail supply power.

Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/powersetpoint

Get the rail setpoint power.

Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/powerlimit

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/powerlimit

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/resistance

Get the rail load resistance.

Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/resistancesetpoint

Set the rail load resistance.

Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/resistancesetpoint

Get the rail setpoint resistance.

Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/kelvinsensingenable

Enable or Disable kelvin sensing on the module.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.

May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/kelvinsensingenable

Determine whether kelvin sensing is enabled or disabled.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/kelvinsensingstate

Determine whether kelvin sensing has been disabled by the system.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /rail/
index/operationalmode

Set the operational mode of the rail.

Refer to the module datasheet for definition of the rail operational capabilities.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/operationalmode

Determine the current operational mode of the system.

Refer to the module datasheet for definition of the rail operational mode capabilities.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/operationalstate

Determine the current operational state of the system.

Refer to the module datasheet for definition of the rail operational states.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /rail/
index/clearfaults

Clears the current fault state of the rail.

Refer to the module datasheet for definition of the rail faults.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Relay Endpoint

Interface to relay entities on BrainStem modules.

Relay entities can be set, and the voltage read.

Other capabilities may be available, please see the product datasheet.

See the [Relay Entity](#) for generic information.

PUT `/api/v1/brainstem/ (serial_num) /relay/
index/enable`

Set the enable/disable state.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as "True", "FALSE", "1", etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /relay/
index/enable`

Get the state.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /relay/
index/voltage`

Get the scaled micro volt value with reference to ground.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Signal Endpoint

Interface to digital pins configured to produce square wave signals.

This class is designed to allow for square waves at various frequencies and duty cycles.

Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time).

See the entity overview section of the reference for more detail regarding the timing.

See the [Signal Entity](#) for generic information.

PUT /api/v1/brainstem/ (*serial_num*) /signal/
index/enable

Enable/Disable the signal output.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /signal/
index/enable

Get the Enable/Disable of the signal.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean

- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /signal/
index/invert

Invert the signal output.

Normal mode is High on t0 then low at t2.

Inverted mode is Low at t0 on period start and high at t2.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /signal/
index/invert

Get the invert status the signal output.

Normal mode is High on t0 then low at t2.

Inverted mode is Low at t0 on period start and high at t2.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /signal/
index/t3time

Set the signal period or T3 in nanoseconds.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /signal/
index/t3time

Get the signal period or T3 in nanoseconds.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /signal/
index/t2time

Set the signal active period or T2 in nanoseconds.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /signal/
index/t2time

Get the signal active period or T2 in nanoseconds.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Store Endpoint

The store provides a flat file system on modules that have storage capacity.

Files are referred to as slots and they have simple zero-based numbers for access.

Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system.

Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure.

Stores have fixed indices based on type.

Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

See the [Store Entity](#) for generic information.

GET /api/v1/brainstem/ (*serial_num*) /store/
index/slotstate/slot

Get slot state.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **slot** – The slot number.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /store/
index/slotenable

Enable slot.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /store/
index/slotdisable

Disable slot.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /store/
index/lock/slot

Gets the current lock state of the slot

Allows for write protection on a slot.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **slot** – The slot number

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /store/
index/lock/slot

Sets the locked state of the slot

Allows for write protection on a slot.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **slot** – The slot number

Request JSON Object

- **value** (*boolean*) – Boolean value to set.

May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

System Endpoint

The System class provides access to the core settings, configuration and system information of the BrainStem module.

The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc.

See the [System Entity](#) for generic information.

GET /api/v1/brainstem/ (*serial_num*) /system/
index/module

Get the current address the module uses on the BrainStem network.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/modulebaseaddress

Get the base address of the module.

Software offsets and hardware offsets are added to this base address to produce the effective module address.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/router

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

This setting must be saved and the board reset before the setting becomes active.

Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module.

Please review the BrainStem network fundamentals before modifying the router address.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/router

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/hbinterval

Set the delay between heartbeat packets which are sent from the module.

For link modules, these these heartbeat are sent to the host.

For non-link modules, these heartbeats are sent to the router address.

Interval values are in 25.6 millisecond increments

Valid values are 1-255; default is 10 (256 milliseconds).

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/hbinterval

Get the delay between heartbeat packets which are sent from the module.

For link modules, these these heartbeat are sent to the host.

For non-link modules, these heartbeats are sent to the router address.

Interval values are in 25.6 millisecond increments.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/led

Set the system LED state. Most modules have a blue system LED.

Refer to the module datasheet for details on the system LED location and color.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean

- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/led

Get the system LED state. Most modules have a blue system LED.

Refer to the module datasheet for details on the system LED location and color.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/ledmaxbrightness

Sets the scaling factor for the brightness of all LEDs on the system.

The brightness is set to the ratio of this value compared to 255 (maximum).

The colors of each LED may be inconsistent at low brightness levels.

Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on.

When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/ledmaxbrightness

Gets the scaling factor for the brightness of all LEDs on the system.

The brightness is set to the ratio of this value compared to 255 (maximum).

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/bootslot

Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command.

Set the slot to 255 to disable mapping on boot.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/bootslot

Get the store slot which is mapped when the module boots.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/version

Get the modules firmware version number.

The version number is packed into the return value.

Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/build

Get the modules firmware build number

The build number is a unique hash assigned to a specific firmware.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/model

Get the module’s model enumeration.

A subset of the possible model enumerations is defined in BrainStem.h under “BrainStem model codes”.

Other codes are be used by Acroname for proprietary module types.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/hardwareversion

Get the module’s hardware revision information.

The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions.

The codes are not well defined and may change at any time.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/serialnumber

Get the module’s serial number.

The serial number is a unique 32 bit integer which is usually communicated in hexadecimal format.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/save

Save the system operating parameters to the persistent module flash memory.

Operating parameters stored in the system flash will be loaded after the module reboots.

Operating parameters include: heartbeat interval, module address, module router address.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

GET /api/v1/brainstem/ (*serial_num*) /system/
index/reset

Reset the system.

A return value of aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

GET /api/v1/brainstem/ (*serial_num*) /system/
index/logevents

Saves system log events to a slot defined by the module (usually ram slot 0).

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

GET /api/v1/brainstem/ (*serial_num*) /system/
index/uptime

Get the module’s accumulated uptime in minutes

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/temperature

Get the module’s current temperature in micro-C

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/mintemperature

Get the module’s minimum temperature ever recorded in micro-C (uC).

This value will persists through a power cycle.

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/maxtemperature

Get the module’s maximum temperature ever recorded in micro-C (uC).

This value will persists through a power cycle.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/inputvoltage

Get the module’s input voltage.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/inputcurrent

Get the module’s input current.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/modulehardwareoffset

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware.

Not all modules support the hardware module address offset. Refer to the module datasheet.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/modulesoftwareoffset

Set the software address offset.

This software offset is added to the module base address, and potentially a module hardware address to produce the final module address.

You must save the system settings and restart for this to take effect.

Please review the BrainStem network fundamentals before modifying the module address.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/modulesoftwareoffset

Get the software address offset.

This software offset is added to the module base address, and potentially a module hardware address to produce the final module address.

You must save the system settings and restart for this to take effect.

Please review the BrainStem network fundamentals before modifying the module address.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/routeraddresssetting

Get the router address system setting.

This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred.

Please review the BrainStem network fundamentals before modifying the module address.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/routetome

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules.

Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device.

Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/powerlimit

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/powerlimitmax

Gets the user defined maximum power limit for the system.

Provides mechanism for defining an unregulated power supplies capability.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/powerlimitmax

Sets a user defined maximum power limit for the system.

Provides mechanism for defining an unregulated power supplies capability.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/powerlimitstate

Gets a bit mapped representation of the factors contributing to the power limit.

Active limit can be found through PowerDeliverClass::getPowerLimit().

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/unregulatedvoltage

Gets the voltage present at the unregulated port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/unregulatedcurrent

Gets the current passing through the unregulated port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/inputpowersource

Provides the source of the current power source in use.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/inputpowerbehavior

Gets the systems input power behavior.

This behavior refers to where the device sources its power from and what happens if that power source goes away.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/inputpowerbehavior

Sets the systems input power behavior.

This behavior refers to where the device sources its power from and what happens if that power source goes away.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/inputpowerbehaviorconfig

Gets the input power behavior configuration

Certain behaviors use a list of ports to determine priority when budgeting power.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /system/
index/inputpowerbehaviorconfig

Sets the input power behavior configuration

Certain behaviors use a list of ports to determine priority when budgeting power.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 32-bit values to set.
May be specified as: * List of integers * List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/name

Gets a user defined name of the device.

Helpful for identifying ports/devices in a static environment.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /system/
index/name`

Sets a user defined name for the device.

Helpful for identification when multiple devices of the same type are present in a system.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 8-bit values to set.

May be specified as: * String (formatted as “Hello”, “x40x41”, etc.) * List of integers * List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a string
- **response.rawValue** (*list(integer)*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /system/
index/resetdevicetofactorydefaults`

Resets the device to it factory default configuration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

GET `/api/v1/brainstem/ (serial_num) /system/
index/linkinterface`

Gets the link interface configuration.

This refers to which interface is being used for control by the device.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /system/
index/linkinterface`

Sets the link interface configuration.

This refers to which interface is being used for control by the device.

See *JSON Schema for Request Object* and *Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/errors

Gets any system level errors.

Calling this function will clear the current errors. If the error persists it will be set again.

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /system/
index/protocolfeatures

Gets the firmware protocol features

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Temperature Endpoint

This entity is only available on certain modules, and provides a temperature reading in microcelsius.

See the [Temperature Entity](#) for generic information.

GET `/api/v1/brainstem/ (serial_num) /temperature/
index/microcelsius`

Get the modules temperature in micro-C

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /temperature/
index/minimummicrocelsius`

Get the module’s minimum temperature in micro-C since the last power cycle.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /temperature/
index/maximummicrocelsius`

Get the module’s maximum temperature in micro-C since the last power cycle.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

Timer Endpoint

The Timer Class provides access to a simple scheduler.

The timer can set to fire only once, or to repeat at a certain interval.

Additionally, a timer entity can execute custom Reflex routines upon firing.

See the [Timer Entity](#) for generic information.

GET `/api/v1/brainstem/ (serial_num) /timer/
index/expiration`

Get the currently set expiration time in microseconds.

This is not a “live” timer. That is, it shows the expiration time originally set with `setExpiration`; it does not “tick down” to show the time remaining before expiration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /timer/
index/expiration`

Set the expiration time for the timer entity.

When the timer expires, it will fire the associated `timer[index]()` reflex.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /timer/
index/mode`

Get the mode of the timer which is either single or repeat mode.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /timer/
index/mode

Set the mode of the timer which is either single or repeat mode.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

UART Endpoint

A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines.

See the [UART Entity](#) for generic information.

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/enable

Enable the UART channel.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/enable

Get the enabled state of the uart.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/baudrate

Set the UART baud rate.

If zero, automatic baud rate selection is used.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/baudrate

Get the UART baud rate.

If zero, automatic baud rate selection is used.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/protocol

Set the UART protocol.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/protocol

Get the UART protocol.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/linkchannel

Set the index of another UART Entity that should be linked to this UART.

If set to the index of this entity, the channel will not be linked.

If set to the index of another UART entity, data will be sent between the two UART entities with no additional processing.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/linkchannel

Gets the index of the UART Entity that this entity is linked to.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/stopbits

Set the UART stop bit configuration

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/stopbits

Set the UART stop bit configuration

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/parity

Set the UART parity.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/parity

Get the UART parity.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/databits

Set the number of bits per character

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/databits

Get the number of bits per character

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /uart/
index/flowcontrol

Set the UART flow control configuration

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/flowcontrol

Set the UART flow control configuration

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/capableprotocols

Returns a bitmask containing a list of protocols that this UART entity is allowed to select.

This does not guarantee that selecting a protocol with “setProtocol” will have an available resource.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /uart/
index/availableprotocols

Returns a bitmask containing a list of protocols that this UART entity is capable of selecting, and has an available protocol resource to assign.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

USB Endpoint

The USB class provides methods to interact with a USB hub and USB switches.

Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

See the [USB Entity](#) for generic information.

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/portenable

Enable both power and data lines for a port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/portdisable

Disable both power and data lines for a port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/dataenable

Enable the only the data lines for a port without changing the state of the power line.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/datadisable

Disable only the data lines for a port without changing the state of the power line.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/hispeeddataenable

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/hispeeddatadisable

Disable only the data lines for a port without changing the state of the power line, Hi- Speed (2.0) only.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/superspeeddataenable

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/superspeeddatadisable

Disable only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/powerenable

Enable only the power line for a port without changing the state of the data lines.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/powerdisable

Disable only the power line for a port without changing the state of the data lines.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/portcurrent/channel

Get the current through the power line for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/portvoltage/channel

Get the voltage on the power line for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/hubmode

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/hubmode

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/portclearerrorstatus

Clear the error status for the given port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/upstreammode

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/upstreammode

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0, usbUpstreamModePort1, and usbUpstreamModeNone.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/upstreamstate

Get the upstream switch state for the USB upstream ports.

Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/hubenumerationdelay

Set the inter-port enumeration delay in milliseconds.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/hubenumerationdelay

Get the inter-port enumeration delay in milliseconds.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/portcurrentlimit/channel

Set the current limit for the port.

If the set limit is not achievable, devices will round down to the nearest available current limit setting.

This setting can be saved with a `stem.system.save()` call.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB downstream channel to limit.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/portcurrentlimit/channel

Get the current limit for the port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB downstream channel to limit.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/portmode/channel

Set the mode for the Port.

The mode is a bitmapped representation of the capabilities of the usb port.

These capabilities change for each of the BrainStem devices which implement the usb entity.

See your device reference page for a complete list of capabilities.

Some devices use a common bit mapping for port mode, as sub-definitions of usbPortMode.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB downstream channel to set the mode on.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/portmode/channel

Get the current mode for the Port.

The mode is a bitmapped representation of the capabilities of the usb port.

These capabilities change for each of the BrainStem devices which implement the usb entity.

See your device reference page for a complete list of capabilities.

Some devices use a common bit mapping for port mode, as sub-definitions of usbPortMode.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB downstream channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/portstate/channel

Get the current State for the Port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB downstream channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/porterror/channel

Get the current error for the Port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB downstream channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer

- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/upstreamboostmode

Set the upstream boost mode.

Boost mode increases the drive strength of the USB data signals (power signals are not changed).

Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins.

Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

This setting is not applied until a stem.system.save() call and power cycle of the hub.

Setting is then persistent until changed or the hub is reset.

After reset, default value of 0% boost is restored.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/downstreamboostmode

Set the downstream boost mode.

Boost mode increases the drive strength of the USB data signals (power signals are not changed).

Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins.

Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

This setting is not applied until a stem.system.save() call and power cycle of the hub.

Setting is then persistent until changed or the hub is reset.

After reset, default value of 0% boost is restored.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/upstreamboostmode

Get the upstream boost mode.

Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/downstreamboostmode

Get the downstream boost mode.

Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/downstreamdataspeed/channel

Get the current data transfer speed for the downstream port.

The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

- **channel** – USB downstream channel to check.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/connectmode/channel

Sets the connect mode of the switch.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/connectmode/channel

Gets the connect mode of the switch.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/cc1enable/channel

Set Enable/Disable on the CC1 line.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

- **channel** – USB channel.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.

May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/cc1enable/channel

Get Enable/Disable on the CC1 line.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB channel.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/cc2enable/channel

Set Enable/Disable on the CC2 line.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB channel.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.

May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/cc2enable/channel

Get Enable/Disable on the CC2 line.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – USB channel.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/cc1current/channel

Get the current through the CC1 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/cc2current/channel

Get the current through the CC2 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/cc1voltage/channel

Get the voltage of CC1 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/cc2voltage/channel

Get the voltage of CC2 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/sbuenable/channel

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/sbuenable/channel

Get the Enable/Disable status of the SBU

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usb/
index/cableflip/channel

Set Cable flip. This will flip SBU, CC and SS data lines.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Request JSON Object

- **value** (*boolean*) – Boolean value to set.
May be specified as a boolean, integer, or a string (formatted as “True”, “FALSE”, “1”, etc.)

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/cableflip/channel

Get Cable flip setting.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*boolean*) – Value formatted as a boolean
- **response.rawValue** (*integer*) – Unformatted value

PUT `/api/v1/brainstem/ (serial_num) /usb/
index/altmode/channel`

Set USB Alt Mode Configuration.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /usb/
index/altmode/channel`

Get USB Alt Mode Configuration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET `/api/v1/brainstem/ (serial_num) /usb/
index/sbu1voltage/channel`

Get the voltage of SBU1 for a port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usb/
index/sbu2voltage/channel

Get the voltage of SBU2 for a port.

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **channel** – The USB sub channel.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

USBSystem Endpoint

The USBSystem class provides high level control of the lower level Port Class.

See the *USBSystem Entity* for generic information.

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/upstreamport

Gets the upstream port.

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/upstreamport

Sets the upstream port.

See *JSON Schema for Request Object* and *Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/enumerationdelay

Gets the inter-port enumeration delay in milliseconds.

Delay is applied upon hub enumeration.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/enumerationdelay

Sets the inter-port enumeration delay in milliseconds.

Delay is applied upon hub enumeration.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/datarolelist

Gets the data role of all ports with a single call

Equivalent to calling PortClass::getDataRole() on each individual port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/enabledlist

Gets the current enabled status of all ports with a single call.

Equivalent to calling PortClass::setEnabled() on each port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/enabledlist

Sets the enabled status of all ports with a single call.

Equivalent to calling PortClass::setEnabled() on each port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.
May be specified as an integer or a string (formatted as "123", "0x12", etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/modelist

Gets the current mode of all ports with a single call.

Equivalent to calling PortClass::getMode() on each port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/modelist

Sets the mode of all ports with a single call.

Equivalent to calling PortClass::setMode() on each port

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 32-bit values to set.
May be specified as: * List of integers * List of strings (formatted as "0x40", "40", etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/statelist

Gets the state for all ports with a single call.

Equivalent to calling PortClass::getState() on each port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/powerbehavior

Gets the behavior of the power manager.

The power manager is responsible for budgeting the power of the system, i.e. What happens when requested power greater than available power.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/powerbehavior

Sets the behavior of how available power is managed, i.e. What happens when requested power is greater than available power.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/powerbehaviorconfig

Gets the current power behavior configuration.

Certain power behaviors use a list of ports to determine priority when budgeting power.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/powerbehaviorconfig

Sets the current power behavior configuration.

Certain power behaviors use a list of ports to determine priority when budgeting power.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 32-bit values to set.

May be specified as: * List of integers * List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list(integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/databehavior

Gets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/databehavior

Sets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/databehaviorconfig

Gets the current data role behavior configuration.

Certain data role behaviors use a list of ports to determine priority host priority.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list (integer)*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/databehaviorconfig

Sets the current data role behavior configuration.

Certain data role behaviors use a list of ports to determine host priority.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*list*) – List of 32-bit values to set.
May be specified as: * List of integers * List of strings (formatted as “0x40”, “40”, etc.)

Response JSON Object

- **response.value** (*string*) – Value formatted as a list of integers
- **response.rawValue** (*list (integer)*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/selectormode

Gets the current mode of the selector input.

This mode determines what happens and in what order when the external selector input is used.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/selectormode

Sets the current mode of the selector input.

This mode determines what happens and in what order when the external selector input is used.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/upstreamhsport

Gets the USB HighSpeed upstream port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/upstreamhsport

Sets the USB HighSpeed upstream port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/upstreamssport

Gets the USB SuperSpeed upstream port.

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/upstreamssport

Sets the USB SuperSpeed upstream port.

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 8-bit Integer value to set.
May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/override

Gets the current enabled overrides

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/override

Sets the current enabled overrides

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/dataahsmaxdatarate

Sets the USB HighSpeed Max datarate

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/dataahsmaxdatarate

Gets the USB HighSpeed Max datarate

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

PUT /api/v1/brainstem/ (*serial_num*) /usbssystem/
index/datassmaxdatarate

Sets the USB SuperSpeed Max datarate

See [JSON Schema for Request Object](#) and [Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.

Request JSON Object

- **value** (*integer*) – 32-bit Integer value to set.

May be specified as an integer or a string (formatted as “123”, “0x12”, etc.)

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

GET /api/v1/brainstem/ (*serial_num*) /usbsystem/
index/datassmaxdatarate

Gets the USB SuperSpeed Max datarate

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.

Response JSON Object

- **response.value** (*integer*) – Value formatted as an integer
- **response.rawValue** (*integer*) – Unformatted value

UserConfig Endpoint

Interface for saving individual entity instances to non-volatile storage.

GET /api/v1/brainstem/ (*serial_num*) /userconfig/
index/saveentitytostore/command/index

Saves the given entity to the Store

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **command** – Protocol Definition for Entity Command (i.e. value of cmdSOMETHING)
- **index** – Index of Entity to access

GET /api/v1/brainstem/ (*serial_num*) /userconfig/
index/resetentitytofactorydefaults/command/index

Resets the given entity to factory defaults

See [JSON Schema for Response Object](#)

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. “0x1234ABCD”). Optional on embedded REST server.

- **index** – Index of entity to access. Default is 0.
- **command** – Protocol Definition for Entity Command (i.e. value of cmdSOMETHING)
- **index** – Index of Entity to access

GET `/api/v1/brainstem/ (serial_num) /userconfig/
index/loadentityfromstore/command/index`

Restores the saved configuration of the given entity

See *JSON Schema for Response Object*

Parameters

- **serial_num** – Serial Number of device to access. Formatted as a hex string (e.g. "0x1234ABCD"). Optional on embedded REST server.
- **index** – Index of entity to access. Default is 0.
- **command** – Protocol Definition for Entity Command (i.e. value of cmdSOMETHING)
- **index** – Index of Entity to access

Examples

GET Command

Bash

```
curl http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/inputvoltage
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/  
↪inputvoltage')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
1 {
2   "timestamp": "2023-09-20T17:53:32.480Z",
3   "request": {
4     "endpointName": "/api/v1/brainstem/3C43352C/system/0/inputvoltage",
5     "parameters": {}
6   },
7   "response": {
8     "value": 22974139,
9     "rawValue": 22974139
10  }
11 }
```

SET Command

Bash

```
curl -X PUT -d '{"value": "New Port Name"}' http://127.0.0.1:9005/api/v1/brainstem/
↪ 3C43352C/port/2/name
```

Python

```
import requests
import json

payload = {}
payload['value'] = 'New Port Name'

response = requests.put('http://127.0.0.1:9005/api/v1/brainstem/3C43352C/port/2/name',
↪ json=payload)
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
1 {
2     "timestamp": "2023-09-20T17:55:15.134Z",
3     "request": {
4         "endpointName": "/api/v1/brainstem/3C43352C/port/2/name",
5         "parameters": {
6             "value": "New Port Name"
7         }
8     },
9     "response": {}
10 }
```

GET Command with no response

Bash

```
curl http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/save
```

Python

```
import requests
import json

response = requests.get('http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/save
↪ ')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
1 {
2     "timestamp": "2023-09-20T17:57:05.245Z",
3     "request": {
4         "endpointName": "/api/v1/brainstem/3C43352C/system/0/save",
5         "parameters": {}
```

(continues on next page)

(continued from previous page)

```

6   },
7   "response": {}
8 }

```

JSON Schema for GET Response Object

The following table lists a [JSON Schema](#)¹²¹ for the response object.

Download the raw JSON Schema file for this response: `get_brainstem_response.json`

Object Fields of Get Brainstem Response

Name	Type	Mandatory	Description
rawValue	<i>v1-rawValue</i>	Yes	
value	string <i>or</i> number	No	If appropriate, a user-friendly formatted value.
units	<i>v1-units</i>	No	

v1-aErr

Nam	Value
Type	string
de- scrip tion	Enumeration of all possible error results
Enu- mer- a- tions	aErrNone, aErrMemory, aErrParam, aErrNotFound, aErrFileNameLength, aErrBusy, aErrIO, aErrMode, aErrWrite, aErrRead, aErrEOF, aErrNotReady, aErrPermission, aErrRange, aErrSize, aErrOverrun, aErrParse, aErrConfiguration, aErrTime-out, aErrInitialization, aErrVersion, aErrUnimplemented, aErrDuplicate, aErrCancel, aErrPacket, aErrConnection, aErrIndexRange, aErrShortCommand, aErrInvalidEntity, aErrInvalidOption, aErrResource, aErrMedia, aErrAsyncReturn, aErrOperation, aErrUnknown

¹²¹ <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

v1-units

Name	Value
Type	string
description	Allowed and recognized units for a value.
Enumeration	volts, millivolts, microvolts, amperes, milliamperes, microamperes, coulombs, millicoulombs, microcoulombs, seconds, milliseconds, microseconds, nanoseconds, kilowatts, watts, milliwatts, microwatts, kilowatt-hours, watt-hours, milliwatt-hours, microwatt-hours

v1-uint32

Name	Value
Type	integer
description	Unsigned 32-bit integer
minimum	0
maximum	4294967295

v1-rawValue

Name	Value
Type	v1-uint32 or Array< v1-uint32 >
description	Unformatted return value from the device.

v1-error

Name	Value
Type	<i>Object</i>
description	Error containing an acronym error

Name	Type	Mandatory	Description
errorCode	v1-acronaErr	Yes	Acroname error code
errorMessage	string	No	A human-readable message to describe the context of the error.

JSON Schema for PUT Request Object

The following table lists a [JSON Schema¹²²](#) for the response object.

Download the raw JSON Schema file for this response: `put_brainstem_payload_v1.json`

Object Fields of Put BrainStem Payload (v1)

Name	Type	Mandatory	Description
value	integer or string or boolean or Array<integer or string>	Yes	Value to set on the Device.

JSON Schema for PUT Response Object

The following table lists a [JSON Schema¹²³](#) for the response object.

Download the raw JSON Schema file for this response: `put_brainstem_response.json`

Object Fields of Put BrainStem Response

Name	Type	Mandatory	Description
errorCode	v1-aErr	Yes	Acroname error code
errorMessage	string	No	A human-readable message to describe the context of the error.
- or -			
thisMessageLeftIntentionallyBlank	null	No	

¹²² <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

¹²³ <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

v1-aErr

Nam	Value
Type	string
de- scrip tion	Enumeration of all possible error results
Enu- mer- a- tions	aErrNone, aErrMemory, aErrParam, aErrNotFound, aErrFileNameLength, aErrBusy, aErrIO, aErrMode, aErrWrite, aErrRead, aErrEOF, aErrNotReady, aErrPermission, aErrRange, aErrSize, aErrOverrun, aErrParse, aErrConfiguration, aErrTime-out, aErrInitialization, aErrVersion, aErrUnimplemented, aErrDuplicate, aErrCancel, aErrPacket, aErrConnection, aErrIndexRange, aErrShortCommand, aErrInvalidEntity, aErrInvalidOption, aErrResource, aErrMedia, aErrAsyncReturn, aErrOperation, aErrUnknown

v1-error

Name	Value
Type	<i>Object</i>
description	Error containing an Acroname error

Name	Type	Mandatory	Description
errorCode	<i>v1-aErr</i>	Yes	Acroname error code
errorMes- sage	string	No	A human-readable message to describe the context of the error.

version Endpoint

The `version` endpoint shows version information for various software components, including BrainStem version, BrainD version, and build information.

There are no input parameters to this endpoint. It will return a JSON object as described below. This endpoint does not use a transaction envelope. Contents are returned exactly as described in the schema below.

JSON Schema for Response Object

The following table lists a [JSON Schema](#)¹²⁴ for the response object.

Download the raw JSON Schema file for this response: `get_version_response.json`

Object Fields of Get Version Response

Name	Type	Mandatory	Description
braind	<i>v1-buildInfo</i>	Yes	Version and Build Information for the braind daemon.
brainstem	<i>v1-buildInfo</i>	Yes	Version and Build Information for the BrainStem library.
control-Room	<i>v1-buildInfo</i>	No	Version and Build Information for the Control Room application.

v1-version

Name	Value
Type	<i>Object</i>
description	Object representing a <major>.<minor>.<patch> semantic version.

Name	Type	Mandatory	Description
major	<i>v1-uint8</i>	Yes	
minor	<i>v1-uint8</i>	Yes	
patch	<i>v1-uint8</i>	Yes	

v1-buildInfo

Name	Value
Type	<i>Object</i>
description	Version and Build Information for an arbitrary software component.

¹²⁴ <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

Name	Type	Mandatory	Description
version	<i>v1-version</i>	Yes	
buildDate	string	No	Date and Time that this software component was built.
buildHash	string	No	Hexadecimal Git commit ID that was used for this build.

v1-uint8

Name	Value
Type	integer
description	Unsigned 8-bit integer
minimum	0
maximum	255

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/version
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/version')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
1 {
2   "braind": {
3     "version": {
4       "major": 1,
5       "minor": 0,
6       "patch": 1
7     },
8     "buildDate": "Wed Aug 23 13:35:12 2023",
9     "buildHash": "b1ba5c6153b7fbbb65c868b65c8ac70bb3d7b7dd"
10  },
11  "brainstem": {
12    "version": {
13      "major": 2,
14      "minor": 10,
15      "patch": 0
16    },
17    "buildDate": "2023-09-11T17:14:25Z",
18    "buildHash": "7af9c07e44601d6987fe3dab0ea201a13609683e"
```

(continues on next page)

(continued from previous page)

```

19   }
20 }
```

Transaction Envelope

Most RESTful Endpoints respond with this JSON object, where `response` is populated by an endpoint-specific schema. The Transaction Envelope schema enables providing error information, request context, and other useful fields about the request and response payload for the specific API call.

The *Acroname Devices*, *Acroname Devices State endpoint*, *Devices*, and *BrainStem* endpoints use this schema as an envelope.

The *Version* endpoint does not use this envelope.

JSON Schema for Transaction Envelope

The following table lists a [JSON Schema](#)¹²⁵.

Download the raw JSON Schema file: `transaction_envelope.json`

Object Fields of BrainD V1 Transaction Envelope

Name	Type	Mandatory	Description
request	<i>Object</i>	Yes	
response	<i>v1-error</i> or boolean or object or array or number or string	Yes	
developer-Notes	Array<null>	No	
timestamp	<i>v1-timestamp</i>	Yes	When the call completed, in ISO8601 time.

Object Fields of `request`

Name	Type	Mandatory	Description
endpointName	string	Yes	Path-section of the URL
params	<i>Object</i>	No	Parameters decoded from the HTTP body.

¹²⁵ <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01>

Object Fields of `request.params`

Name	Type	Mandatory	Description
------	------	-----------	-------------

v1-aErr

Nam	Value
Type	string
de- scrip- tion	Enumeration of all possible error results
Enu- mer- a- tions	aErrNone, aErrMemory, aErrParam, aErrNotFound, aErrFileNameLength, aErrBusy, aErrIO, aErrMode, aErrWrite, aErrRead, aErrEOF, aErrNotReady, aErrPermission, aErrRange, aErrSize, aErrOverrun, aErrParse, aErrConfiguration, aErrTime-out, aErrInitialization, aErrVersion, aErrUnimplemented, aErrDuplicate, aErrCancel, aErrPacket, aErrConnection, aErrIndexRange, aErrShortCommand, aErrInvalidEntity, aErrInvalidOption, aErrResource, aErrMedia, aErrAsyncReturn, aErrOperation, aErrUnknown

v1-error

Name	Value
Type	<i>Object</i>
description	Error containing an Acroname error

Name	Type	Mandatory	Description
errorCode	<i>v1-aErr</i>	Yes	Acroname error code
errorMes- sage	string	No	A human-readable message to describe the context of the error.

v1-timestamp

Name		Value
Type		string
description		String containing a ISO8601 timestamp (YYYY-MM-DDTHH:MM:SS.mmmZ
Regular Expression		^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{3,6}Z\$

Examples

These are examples of payloads for endpoints that use a transaction envelope. These examples closely follow the examples on every endpoint's reference page, but include the envelope for verbosity.

Acroname Devices

The success-case of `response` is detailed in *Acroname Devices endpoint* page.

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevices
```

A reasonable output in the expected case:

```
1 {
2   "timestamp": "2023-09-20T17:11:54.092Z",
3   "request": {
4     "endpointName": "/api/v1/acronameDevices",
5     "parameters": {}
6   },
7   "response": {
8     "acronameDevices": [
9       {
10        "type": "USB",
11        "serialNumber": "3C43352C",
12        "module": 6,
13        "model": "USBHub3c",
14        "entitlements": [
15          {
16            "tag": "CONTROL",
17            "version": 0
18          }
19        ]
20      },
21      {
22        "type": "USB",
23        "serialNumber": "F7D9AFB6",
24        "module": 6,
25        "model": "USBHub3p",
26        "entitlements": [
27          {
28            "tag": "CONTROL",
```

(continues on next page)

(continued from previous page)

```

29         "version": 0
30     }
31 ]
32 }
33 ]
34 }
35 }

```

The Acroname Device endpoint does not emit errors. But if it did, it might look something like:

```

1 {
2     "timestamp": "2023-10-05T18:34:30.200Z",
3     "request": {
4         "endpointName": "/api/v1/acronameDevices",
5         "parameters": {}
6     },
7     "response": {
8         "errorCode": "aErrUnknown",
9         "errorMessage": "Unhandled exception while gathering device names."
10    }
11 }

```

Acroname Devices State

The success-case of `response` is detailed in [Acroname Devices State endpoint](#) page.

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevicesState
```

A reasonable output in the expected case:

```

1 {
2     "timestamp": "2023-09-20T17:24:24.428Z",
3     "request": {
4         "endpointName": "/api/v1/acronameDevicesState",
5         "parameters": {}
6     },
7     "response": {
8         "timestamp": "2023-09-20T17:24:24.108Z",
9         "sequence": 2563,
10        "brainstemVersion": {
11            "major": 2,
12            "minor": 10,
13            "patch": 0
14        },
15        "acronameDevices": [
16            {
17                "serialNumber": "0x3C43352C",
18                "firmwareVersion": {
19                    "major": 2,
20                    "minor": 10,
21                    "patch": 0
22                },
23                "model": "USBHub3c",

```

(continues on next page)

(continued from previous page)

```

24     "upstreamPort": 0,
25     "upstreamPortSelectionIsAutomatic": true,
26     "ports": [
27         {
28             "index": 0,
29             "physicalName": "Port 0",
30             "speed": "5 Gbps",
31             "roles": [
32                 "Upstream",
33                 "Downstream"
34             ],
35             "currentDataRole": "Upstream",
36             "enabled": true,
37             "voltage": {
38                 "value": 5.207519,
39                 "units": "volts",
40                 "rawValue": 5207519
41             },
42             "current": {
43                 "value": 0.00061,
44                 "units": "amperes",
45                 "rawValue": 610
46             }
47         },
48         {
49             "index": 1,
50             "physicalName": "Port 1",
51             "speed": "Unknown",
52             "roles": [
53                 "Upstream",
54                 "Downstream"
55             ],
56             "currentDataRole": "Downstream",
57             "enabled": true,
58             "voltage": {
59                 "value": 0.0,
60                 "units": "volts",
61                 "rawValue": 0
62             },
63             "current": {
64                 "value": 0.000305,
65                 "units": "amperes",
66                 "rawValue": 305
67             }
68         },
69         <additional ports ...>
70     ]
71 }
72 ]
73 }
74 }

```

The *Acroname Devices State endpoint* emits an error in the case where licenses were not found.

```

1 {
2     "timestamp": "2023-10-05T18:57:49.317Z",

```

(continues on next page)

(continued from previous page)

```

3   "request": {
4       "endpointName": "/api/v1/acronameDevicesState",
5       "parameters": {}
6   },
7   "response": {
8       "errorCode": "aErrPermission",
9       "errorMessage": "The call is not allowed, check permissions and licenses.
10      ↪associated with this call"
11   }
12 }

```

Devices

The success-case of response is detailed in the [Devices endpoint](#) page.

Bash

```
curl http://127.0.0.1:9005/api/v1/devices
```

A reasonable output in the expected case:

```

1  {
2      "timestamp": "2023-09-20T17:36:15.359Z",
3      "request": {
4          "endpointName": "/api/v1/devices",
5          "parameters": {}
6      },
7      "response": {
8          "usb": [
9              {
10                 "vendorId": "0x067B",
11                 "productId": "0x2303",
12                 "speedActual": "12 Mbps",
13                 "maxSpeed": "12 Mbps",
14                 "productName": "USB-Serial Controller",
15                 "serialNumber": "",
16                 "manufacturer": "Prolific Technology Inc.",
17                 "deviceClass": 0,
18                 "interfaces": [
19                     {
20                         "interfaceIndex": 0,
21                         "entryIndex": 0,
22                         "kClass": 255,
23                         "kSubclass": 0,
24                         "kProtocol": 0
25                     }
26                 ],
27                 "portPath": [
28                     4,
29                     3,
30                     1
31                 ],
32                 "acronameDevice": {
33                     "serialNumber": "0xF7D9AFB6",
34                     "port": 0,

```

(continues on next page)

(continued from previous page)

```

35         "distance": 0
36     },
37     "bulkCapability": {
38         "isDoing": true,
39         "isCapable": true
40     },
41     "isochronousCapability": {
42         "isDoing": false,
43         "isCapable": false
44     },
45     "interruptCapability": {
46         "isDoing": true,
47         "isCapable": true
48     }
49 },
50 {
51     "vendorId": "0x2188",
52     "productId": "0x0747",
53     "speedActual": "5 Gbps",
54     "maxSpeed": "5 Gbps",
55     "productName": "Card Reader ",
56     "serialNumber": "000000000010",
57     "manufacturer": "CalDigit",
58     "deviceClass": 0,
59     "interfaces": [
60         {
61             "interfaceIndex": 0,
62             "entryIndex": 0,
63             "kClass": 8,
64             "kSubclass": 6,
65             "kProtocol": 80
66         }
67     ],
68     "portPath": [
69         4,
70         8
71     ],
72     "bulkCapability": {
73         "isDoing": true,
74         "isCapable": true
75     },
76     "isochronousCapability": {
77         "isDoing": false,
78         "isCapable": false
79     },
80     "interruptCapability": {
81         "isDoing": false,
82         "isCapable": false
83     }
84 }
85 ]
86 }
87 }

```

The *Devices endpoint* emits an error in the case where licenses were not found.

```

1 {
2   "timestamp": "2023-10-05T19:11:51.113Z",
3   "request": {
4     "endpointName": "/api/v1/devices",
5     "parameters": {}
6   },
7   "response": {
8     "errorCode": "aErrPermission",
9     "errorMessage": "The call is not allowed, check permissions and licenses_
10    ↪associated with this call"
11   }
12 }

```

BrainStem

The success-case of response is detailed in *BrainStem Endpoint* page.

Bash

```
curl http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/inputvoltage
```

A reasonable output in the expected case:

```

1 {
2   "timestamp": "2023-09-20T17:53:32.480Z",
3   "request": {
4     "endpointName": "/api/v1/brainstem/3C43352C/system/0/inputvoltage",
5     "parameters": {}
6   },
7   "response": {
8     "value": 22974139,
9     "rawValue": 22974139
10  }
11 }

```

The *BrainStem endpoint* will emit errors specific to malformed requests, or hardware exceptions.

Example: Invalid value field supplied

```

1 {
2   "timestamp": "2023-10-05T20:11:55.218Z",
3   "request": {
4     "endpointName": "/api/v1/brainstem/F2C3B5AC/port/5/Enabled/",
5     "parameters": {
6       "value": "9"
7     }
8   },
9   "response": {
10    "errorCode": "aErrParse",
11    "errorMessage": "Value can not be converted to an integer value."
12  }
13 }

```

Example: 8-port hardware does not support index 9

```
1 {
2   "timestamp": "2023-10-05T20:13:16.954Z",
3   "request": {
4     "endpointName": "/api/v1/brainstem/F2C3B5AC/port/9/Enabled/",
5     "parameters": {
6       "value": "1"
7     }
8   },
9   "response": {
10    "errorCode": "aErrUnimplemented",
11    "errorMessage": "Functionality unimplemented"
12  }
13 }
```

3.6 .NET API Reference

3.6.1 BrainStem2 CLI Types

```
enum class Acroname::BrainStem2CLI::aErr
```

Values:

enumerator **aErrNone**

0 - Success, no error.

enumerator **aErrMemory**

1 - Memory allocation.

enumerator **aErrParam**

2 - Invalid parameter.

enumerator **aErrNotFound**

3 - Not found.

enumerator **aErrFileNameLength**

4 - File name too long.

enumerator **aErrBusy**

5 - Resource busy.

enumerator **aErrIO**

6 - Input/Output error.

enumerator **aErrMode**

7 - Invalid Mode.

enumerator **aErrWrite**

8 - Write error.

enumerator **aErrRead**

9 - Read error.

enumerator **aErrEOF**

10 - End of file.

enumerator **aErrNotReady**

11 - Not ready, no bytes available.

enumerator **aErrPermission**

12 - Insufficient permissions.

- enumerator **aErrRange**
13 - Value out of range.
- enumerator **aErrSize**
14 - Invalid Size.
- enumerator **aErrOverrun**
15 - Buffer/queue overrun.
- enumerator **aErrParse**
16 - Parse error.
- enumerator **aErrConfiguration**
17 - Configuration error.
- enumerator **aErrTimeout**
18 - Timeout occurred.
- enumerator **aErrInitialization**
19 - Initialization error.
- enumerator **aErrVersion**
20 - Invalid version.
- enumerator **aErrUnimplemented**
21 - Functionality unimplemented.
- enumerator **aErrDuplicate**
22 - Duplicate request.
- enumerator **aErrCancel**
23 - Cancelation occurred, or did not complete.
- enumerator **aErrPacket**
24 - Packet unsigned char invalid.
- enumerator **aErrConnection**
25 - Connection error.
- enumerator **aErrIndexRange**
26 - Index out of range.
- enumerator **aErrShortCommand**
27 - BrainStem command too short.

enumerator **aErrInvalidEntity**

28 - Invalid entity error.

enumerator **aErrInvalidOption**

29 - Invalid option code.

enumerator **aErrResource**

30 - Resource unavailable.

enumerator **aErrMedia**

31 - Media error.

enumerator **aErrAsyncReturn**

32 - Asynchronous return.

enumerator **aErrStreamStale**

33 - Stream value is stale.

enumerator **aErrUnknown**

34 - Unknown error.

3.6.2 Port Mapping

enum class **Acroname::BrainStem2CLI::PORT_SPEED**

Port speed enumeration

Values:

enumerator **kPORT_SPEED_UNKNOWN**

kPORT_SPEED_UNKNOWN (0)

enumerator **kPORT_SPEED_LOW**

kPORT_SPEED_LOW (1)

enumerator **kPORT_SPEED_FULL**

kPORT_SPEED_FULL (2)

enumerator **kPORT_SPEED_HIGH**

kPORT_SPEED_HIGH (3)

enumerator **kPORT_SPEED_SUPER**

kPORT_SPEED_SUPER (4)

enumerator **kPORT_SPEED_SUPER_PLUS**

kPORT_SPEED_SUPER_PLUS (5)

class DeviceNode

Device Node Structure - Contains information linking the downstream device to the Acroname Hub.

Public Functions

DeviceNode (DeviceNode_t&)

Constructor.

~DeviceNode ()

Destructor.

!DeviceNode ()

Finalizer.

Public Members

const unsigned int **hubSerialNumber**

Acroname Device Information.

Serial number of the Acroname hub where the device was found.

const unsigned int **hubPort**

Port of the Acroname hub where the device was found.

const unsigned short **idVendor**

Downstream device information.

Manufactures Vendor ID of the downstream device.

const unsigned short **idProduct**

Manufactures Product ID of the downstream device.

const *PORT_SPEED* **speed**

The devices downstream device speed.

const String ^ **productName**

USB string descriptor

const String ^ **serialNumber**

USB string descriptor

const String ^ **manufacturer**

USB string descriptor

class PortMapping

The *PortMapping* Gets downstream device USB information for all Acroname hubs.

Public Functions

PortMapping()

Constructor. Calls “update” on construction. The error can be accessed through the “lastError” member.

~PortMapping()

Destructor.

!PortMapping()

Finalizer.

aErr **update**(void)

Updates the current device list. Calls the underlying C function getDownstreamDevices

Public Members

cli::array< DeviceNode^> ^ deviceList

List of all devices found downstream of an Acraname hub.

aErr **lastError**

Provides access to error code of update on construction.

3.6.3 Module Class

class ModuleClass

ModuleClass. Provides a generic interface to a BrainStem hardware module. The Module class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

Public Functions

ModuleClass(const unsigned char address, const unsigned char model)

Constructors.

~ModuleClass()

Destructor.

!ModuleClass()

Finalizer.

BrainStem2CLI::*aErr* **discoverAndConnect**(m_linkType transport)

A discovery-based connect. This member function will attempt to connect to the first BrainStem module found. If this module does not match the object type the connection will fail.

Parameters

transport -- Transport on which to search for available BrainStem link modules. See the m_linkType “transport” enum for supported transports.

Returns

aErr::aErrBusy - if the module is already in use.

Returns

aErr::aErrParam - if the transport type is undefined.

Returns

aErr::aErrNotFound - if the module cannot be found.

Returns

aErr::aErrNone If the connect was successful.

BrainStem2CLI::aErr discoverAndConnect (m_linkType transport, unsigned int serialNum)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will

Parameters

- **transport** - - Transport on which to search for available BrainStem link modules. See the m_linkType “transport” enum for supported transports.
- **serialNum** - - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Returns

aErr::aErrBusy - if the module is already in use.

Returns

aErr::aErrParam - if the transport type is undefined.

Returns

aErr::aErrNotFound - if the module cannot be found.

Returns

aErr::aErrNone If the connect was successful.

BrainStem2CLI::aErr connectFromSpec (m_linkSpec spec)

Connect to a link with a fully defined specifier.

Parameters

spec - - Connect to module with specifier.

Returns

aErr::aErrInitialization - If there is currently no link object.

Returns

aErr::aErrBusy - If the link is currently connected.

Returns

aErr::aErrParam - if the specifier is incorrect.

Returns

aErr::aErrNotFound - if the module cannot be found.

Returns

aErr::aErrNone If the connect was successful.

BrainStem2CLI::aErr connect (m_linkType transport)

Connect using the current link specifier.

Parameters

transport - - Transport on which to search for available BrainStem link modules. See the m_linkType “transport” enum for supported transports.

Returns

aErr::aErrBusy - if the module is already in use.

Returns

aErr::aErrParam - if the type is incorrect or serialNum is not specified

Returns

aErr::aErrNotFound - if the module cannot be found.

Returns

aErr::aErrNone If the connect was successful.

BrainStem2CLI::aErr connect (m_linkType transport, unsigned int serialNum)

Connect using the current link specifier.

Parameters

- **transport** - - Transport on which to search for available BrainStem link modules. See the m_linkType "transport" enum for supported transports.
- **serialNum** - - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Returns

aErr::aErrBusy - if the module is already in use.

Returns

aErr::aErrParam - if the type is incorrect or serialNum is not specified

Returns

aErr::aErrNotFound - if the module cannot be found.

Returns

aErr::aErrNone If the connect was successful.

BrainStem2CLI::aErr connectThroughLinkModule (BrainStem2CLI::ModuleClass
^module)

Connect using link from another Module. This member function will connect to the same BrainStem used by given Module. If a link module is found on the specified transport, a connection will

Parameters

module - - Pointer to a valid Module class object.

Returns

aErr::aErrParam - if the module is undefined.

Returns

aErr::aErrNone - if the connect was successful.

BrainStem2CLI::aErr disconnect ()

Disconnect from the BrainStem module.

Returns

aErr::aErrResource - If the there is no valid connection.

Returns

aErr::aErrConnection - If the disconnect failed, due to a communication issue.

Returns

aErr::aErrNone If the disconnect was successful.

BrainStem2CLI::aErr reconnect ()

Reconnect using the current link specifier.

Returns

aErr::aErrBusy - if the module is already in use.

Returns

aErr::aErrParam - if the specifier is incorrect.

Returns

aErr::aErrNotFound - if the module cannot be found.

Returns

aErr::aErrNone If the connect was successful.

bool isConnected ()

Is the link connected to the BrainStem Module.

BrainStem2CLI::linkStatus **getStatus** ()

Check the status of the module connection.

Returns

linkStatus (see aLink.h for status values)

BrainStem2CLI::aErr **getConfig** (BrainStem2CLI::aEtherConfig ^config)

Gets the links current aEther configuration

Parameters

config – Variable to be filled with the config

Returns

aErrNone on success. aErrNotReady if the module does not have a link aErrParam if config is NULL

cli::array<unsigned int> **getIPv4Interfaces** (void)

Populates a list with all of the available IPv4 Interfaces.

Returns

An array of network interfaces.

BrainStem2CLI::aErr **setConfig** (BrainStem2CLI::aEtherConfig ^config)

Sets the links aEther configuration. Configuration must be applied BEFORE connecting

Parameters

config – Configuration to be applied

Returns

aErrNone on success. aErrPermission if the module is currently connected. aErrNotReady if the module does not have a link

void **setModuleAddress** (const unsigned char address)

Accessor to set the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

Parameters

address – The module address.

unsigned char **getModuleAddress** ()

Accessor to get the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

Returns

The module address.

BrainStem2CLI::aErr **getBuild** (unsigned int% build)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Parameters

build – Variable the build will be placed in.

BrainStem2CLI::aErr **hasUEI** (const unsigned char command, const unsigned char option, const unsigned char index, const unsigned char flags)

Queries the module to determine if it implements a UEI. Each UEI has a command, option or variant, index and flag. The hasUEI method queries for a fully specified UEI. Returns aErrNone if the variation is supported and an appropriate error if not. This call is blocking for up to the nMSTimeout period.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **option** – The option or variant of the command.

- **index** – The entity index.
- **flags** – The flags (ueiOPTION_SET or ueiOPTION_GET).

BrainStem2CLI::aErr classQuantity (const unsigned char command, unsigned char% count)

Queries the module to determine how many entities of the specified class are implemented by the module. Zero is a valid return value. For example, calling classQuantity with the command parameter of cmdANALOG would return the number of analog entities implemented by the module.

Parameters

- **command** – One of UEI commands (cmdXXX).
- **count** – When the request is successful count is updated with the number of entities found.

BrainStem2CLI::aErr subClassQuantity (const unsigned char command, const unsigned char index, unsigned char% count)

Queries the module to determine how many subclass entities of the specified class are implemented by the module for a given entity index. This is used for entities which may be 2-dimensional. E.g. cmdMUX subclasses are the number of channels supported by a particular mux type (index); as a specific example, a module may support 4 UART channels, so subClassQuantity(cmdMUX, aMUX_UART...) could return 4. Zero is a valid return value.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **count** – The number of subclasses found.

BrainStem2CLI::aErr entityGroup (const unsigned char command, const unsigned char index, unsigned char% group)

Queries the module the group assigned to an entity and index. Entities groups are used to specify when certain hardware features are fundamentally related. E.g. certain hardware modules may have some digital pins associated with an adjustable voltage rail; these digitals would be in the same group as the rail. Zero is the default group.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **group** – Upon success, group is filled with the entities group value.

void setNetworkingMode (const bool mode)

Sets the networking mode of the module object. By default the module object is configured to automatically adjust its address based on the device's current module address. So that, if the device has a software or hardware offset it will still be able to communicate with the device. If advanced networking is required the auto networking mode can be turned off.

Parameters

mode – True/1 for Auto Networking, False/0 for manual networking

Public Static Functions

```
static BrainStem2CLI::m_linkSpec findFirstModule (Brain-  
Stem2CLI::m_linkType type, [Optional] Nullable< un-  
signed int > networkInterface)
```

Finds the first module found on the given transport

Parameters

type – The transport type on which to search for devices. Valid m_linkType “linktypes” are accepted.

Returns

If found, the linkspec will be populated with the devices information. Values will be all zeros otherwise.

```
static BrainStem2CLI::m_linkSpec findModule (Brain-  
Stem2CLI::m_linkType type, unsigned int serialNum, [Optional] Nul-  
lable< unsigned int > networkInterface)
```

Finds the module with the given serial number on the given transport type.

Parameters

- **type** – The transport type on which search for devices. Valid m_linkType “linktypes” are accepted
- **serialNum** – Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Returns

If found the linkspec will be populated with the devices information. Values will be all zeros otherwise.

```
static BrainStem2CLI::aErr sDiscover (Brain-  
Stem2CLI::m_linkType type, List< m_linkSpec > **specList, [Op-  
tional] Nullable< unsigned int > networkInterface)
```

Discover is called with a specified transport to search for link modules on that transport. The devices list is filled with device specifiers. sDiscover returns aErrNone if the discovery process is successful, regardless of whether any links are found. An error is only returned if the link discovery process fails. Discovery can take some time.

Parameters

- **type** – Transport to search for available BrainStem link modules on. See the m_linkType “transport” enum for supported transports.
- **specList** – an empty list of specifiers that will be filled in.

Returns

aErr::aErrNotFound if no devices were found.

Returns

aErr::aErrNone on success.

3.6.4 Analog Class

See the [Analog Entity](#) for generic information.

class **AnalogClass** : public Acroname::BrainStem2CLI::EntityClass

AnalogClass: Interface to analog entities on BrainStem modules. Analog entities may be configured as an input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while others simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

Public Functions

AnalogClass ()

Constructors.

~AnalogClass ()

Destructor.

!AnalogClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdANALOG index to be addressed.

aErr **getValue** (unsigned short %value)

Get the raw ADC output value in bits.

Note: Not all modules provide 16 useful bits; this value's least significant bits are zero-padded to 16 bits. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Parameters

value – 16 bit analog reading with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

Returns

Returns *common entity* return values

aErr **getVoltage** (int %microvolts)

Get the scaled micro volt value with reference to ground.

Note: Not all modules provide 32 bits of accuracy. Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

microvolts – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns *common entity* return values

aErr **getRange** (unsigned char %range)

Get the analog input range.

Parameters

range – 8 bit value corresponding to a discrete range option

Returns

Returns *common entity* return values

aErr **getEnable** (unsigned char %enable)

Get the analog output enable status.

Parameters

enable – 0 if disabled 1 if enabled.

Returns

Returns *common entity* return values

aErr **setValue** (const unsigned short value)

Set the value of an analog output (DAC) in bits.

Note: Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Parameters

value – 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

Returns

Returns *common entity* return values

aErr **setVoltage** (const int microvolts)

Set the voltage level of an analog output (DAC) in microvolts.

Note: Voltage range is dependent on the specific DAC channel range.

Parameters

microvolts – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns *common entity* return values

aErr **setRange** (const unsigned char range)

Set the analog input range.

Parameters

range – 8 bit value corresponding to a discrete range option

Returns

Returns *common entity* return values

aErr **setEnabled** (const unsigned char enable)

Set the analog output enable state.

Parameters

enable – set 1 to enable or 0 to disable.

Returns

Returns *common entity* return values

aErr **setConfiguration** (const unsigned char configuration)

Set the analog configuration.

Parameters

configuration – bitAnalogConfigurationOutput configures the analog entity as an output.

Returns

Returns *common entity* return values

aErr **getConfiguration** (unsigned char %configuration)

Get the analog configuration.

Parameters

configuration – Current configuration of the analog entity.

Returns

Returns *common entity* return values

aErr **setBulkCaptureSampleRate** (const unsigned int value)

Set the sample rate for this analog when bulk capturing.

Parameters

value – sample rate in samples per second (Hertz).

- Minimum rate: 7,000 Hz
- Maximum rate: 200,000 Hz

Returns

Returns *common entity* return values

aErr **getBulkCaptureSampleRate** (unsigned int %value)

Get the current sample rate setting for this analog when bulk capturing.

Parameters

value – upon success filled with current sample rate in samples per second (Hertz).

Returns

Returns *common entity* return values

aErr **setBulkCaptureNumberOfSamples** (const unsigned int value)

Set the number of samples to capture for this analog when bulk capturing.

Parameters

value – number of samples.

- Minimum # of Samples: 0

- Maximum # of Samples: $(\text{BRAINSTEM_RAM_SLOT_SIZE} / 2) = (3\text{FFF} / 2) = 1\text{FFF} = 8191$

Returns

Returns *common entity* return values

aErr **getBulkCaptureNumberOfSamples** (unsigned int %value)

Get the current number of samples setting for this analog when bulk capturing.

Parameters

value – number of samples.

Returns

Returns *common entity* return values

aErr **initiateBulkCapture** ()

Initiate a BulkCapture on this analog. Captured measurements are stored in the module's RAM store (RAM_STORE) slot 0. Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

Note: When the bulk capture is complete *getBulkCaptureState()* will return either bulkCaptureFinished or bulkCaptureError.

Returns

Returns *common entity* return values

aErr **getBulkCaptureState** (unsigned char %state)

Get the current bulk capture state for this analog.

Parameters

state – the state of bulk capture.

- Idle: bulkCaptureIdle = 0
- Pending: bulkCapturePending = 1
- Finished: bulkCaptureFinished = 2
- Error: bulkCaptureError = 3

Returns

Returns *common entity* return values

3.6.5 App Class

See the *App Entity* for generic information.

```
class AppClass : public Acroname::BrainStem2CLI::EntityClass
```

AppClass: Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

Public Functions

AppClass ()

Constructors.

~AppClass ()

Destructor.

!AppClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdAPP index to be addressed.

aErr execute (const unsigned int appParam)

Execute the app reflex on the module. Doesn't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

Parameters

appParam – The app parameter handed to the reflex.

Return values

- **aErrNone** – success.
- **aErrTimeout** – The request timed out waiting to start execution.
- **aErrConnection** – No active link connection.
- **aErrNotFound** – the app reflex was not found or not enabled on the module.

aErr execute (const unsigned int appParam, unsigned int %returnVal)

Execute the app reflex on the module. Waits for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

Parameters

- **appParam** – The app parameter handed to the reflex.
- **returnVal** – The return value filled in from the result of executing the reflex routine.

Return values

- **aErrNone** – - success.
- **aErrTimeout** – - The request timed out waiting for a response.
- **aErrConnection** – - No active link connection.
- **aErrNotFound** – - the app reflex was not found or not enabled on the module.

aErr execute (const unsigned int appParam, unsigned int %returnVal, const unsigned int msTimeout)

Execute the app reflex on the module. Waits for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

Parameters

- **appParam** – The app parameter handed to the reflex.
- **returnVal** – The return value filled in from the result of executing the reflex routine.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

Return values

- **aErrNone** – - success.
- **aErrTimeout** – - The request timed out waiting for a response.
- **aErrConnection** – - No active link connection.
- **aErrNotFound** – - the app reflex was not found or not enabled on the module.

3.6.6 Clock Class

See the [Clock Entity](#) for generic information.

class **ClockClass** : public Acroname::BrainStem2CLI::EntityClass

ClockClass: Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

Note: Clock time must be reset if power to the BrainStem module is lost.

Public Functions

ClockClass ()

Constructors.

~ClockClass ()

Destructor.

!ClockClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdCLOCK index to be addressed.

aErr **getYear** (unsigned short %year)

Get the four digit year value (0-4095).

Parameters

year – Get the year portion of the real-time clock value.

Returns

Returns [common entity](#) return values

aErr **setYear** (const unsigned short year)

Set the four digit year value (0-4095).

Parameters

year – Set the year portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getMonth** (unsigned char %month)

Get the two digit month value (1-12).

Parameters

month – The two digit month portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setMonth** (const unsigned char month)

Set the two digit month value (1-12).

Parameters

month – The two digit month portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getDay** (unsigned char %day)

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Parameters

day – The two digit day portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setDay** (const unsigned char day)

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Parameters

day – The two digit day portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getHour** (unsigned char %hour)

Get the two digit hour value (0-23).

Parameters

hour – The two digit hour portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setHour** (const unsigned char hour)

Set the two digit hour value (0-23).

Parameters

hour – The two digit hour portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getMinute** (unsigned char %minute)

Get the two digit minute value (0-59).

Parameters

minute – The two digit minute portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setMinute** (const unsigned char minute)

Set the two digit minute value (0-59).

Parameters

minute – The two digit minute portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **getSecond** (unsigned char %second)

Get the two digit second value (0-59).

Parameters

second – The two digit second portion of the real-time clock value.

Returns

Returns *common entity* return values

aErr **setSecond** (const unsigned char second)

Set the two digit second value (0-59).

Parameters

second – The two digit second portion of the real-time clock value.

Returns

Returns *common entity* return values

3.6.7 Digital Class

See the *Digital Entity* for generic information.

```
class DigitalClass : public Acroname::BrainStem2CLI::EntityClass
```

DigitalClass: Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

Public Functions

```
DigitalClass ()
```

Constructors.

```
~DigitalClass ()
```

Destructor.

```
!DigitalClass ()
```

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdDIGITAL index to be addressed.

aErr setConfiguration (const unsigned char configuration)

Set the digital configuration to one of the available 5 states.

Note: Some configurations are only supported on specific pins.

Parameters

configuration – The configuration to be applied

- Digital Input: digitalConfigurationInput = 0
- Digital Output: digitalConfigurationOutput = 1
- RCServo Input: digitalConfigurationRCServoInput = 2
- RCServo Output: digitalConfigurationRCServoOutput = 3
- High Z State: digitalConfigurationHiZ = 4
- Digital Input: digitalConfigurationInputPullUp = 0
- Digital Input: digitalConfigurationInputNoPull = 4
- Digital Input: digitalConfigurationInputPullDown = 5

Returns

Returns *common entity* return values

aErr getConfiguration (unsigned char %configuration)

Get the digital configuration.

Parameters

configuration – Current configuration of the digital entity.

Returns

Returns *common entity* return values

aErr setState (const unsigned char state)

Set the logical state.

Parameters

state – The state to be set. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr getState (unsigned char %state)

Get the state.

Note: If in high Z state an error will be returned.

Parameters

state – The current state of the digital entity. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **setStateAll** (const unsigned int state)

Sets the logical state of all available digitals based on the bit mapping. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

Parameters

state – The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc.

Returns

Returns *common entity* return values

aErr **getStateAll** (unsigned int %state)

Gets the logical state of all available digitals in a bit mapped representation. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

Parameters

state – The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **setValue** (const unsigned char value)

Set the expected value of the digital pin.

Parameters

value – The expected value of the digital pin. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **getValue** (unsigned char %value)

Get the expected value of the digital pin.

Parameters

value – The expected value of the digital pin. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **setLinkChannel** (const unsigned char linkChannel)

Set the link channel (entity index) for linking digital entities. Two digital entities will link when one is configured as LinkInput, one as LinkOutput, and both have each others linkChannel indexes.

Parameters

linkChannel – The link channel (entity index) value. Entities with matching linkChannel values can be linked.

Returns

Returns *common entity* return values

aErr **getLinkChannel** (unsigned char %linkChannel)

Get the link channel (entity index) for linking digital entities.

Parameters

linkChannel – The current link channel (entity index) value.

Returns

Returns *common entity* return values

3.6.8 Equalizer Class

See the *Equalizer Entity* for generic information.

class **EqualizerClass** : public Acroname::BrainStem2CLI::EntityClass

EqualizerClass: Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

Public Functions

EqualizerClass()

Constructors.

~EqualizerClass()

Destructor.

!EqualizerClass()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdEQUALIZER index to be addressed.

aErr **setReceiverConfig** (const unsigned char channel, const unsigned char config)

Sets the receiver configuration for a given channel.

Parameters

- **channel** – The equalizer receiver channel.
- **config** – Configuration to be applied to the receiver.

Returns

Returns *common entity* return values

aErr **getReceiverConfig** (const unsigned char channel, unsigned char %config)

Gets the receiver configuration for a given channel.

Parameters

- **channel** – The equalizer receiver channel.
- **config** – Configuration of the receiver.

Returns

Returns *common entity* return values

aErr **setTransmitterConfig** (const unsigned char config)

Sets the transmitter configuration

Parameters

config – Configuration to be applied to the transmitter.

Returns

Returns *common entity* return values

aErr **getTransmitterConfig** (unsigned char %config)

Gets the transmitter configuration

Parameters

config – Configuration of the Transmitter.

Returns

Returns *common entity* return values

3.6.9 Ethernet Class

See the *Ethernet Entity* for generic information.

class **EthernetClass** : public Acroname::BrainStem2CLI::EntityClass

EthernetClass: IP configuration. MAC info. BrainD port.

Public Functions

EthernetClass ()

Constructors.

~EthernetClass ()

Destructor.

!EthernetClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdETHERNET index to be addressed.

aErr **setEnabled** (const unsigned char enabled)

Sets the Ethernet's interface to enabled/disabled.

Parameters

enabled – 1 = enabled; 0 = disabled

Returns

Returns *common entity* return values

aErr **getEnabled** (unsigned char %enabled)

Gets the current enable value of the Ethernet interface.

Parameters

enabled – 1 = Fully enabled network connectivity; 0 = Ethernet MAC is disabled.

Returns

Returns *common entity* return values

aErr **getNetworkConfiguration** (unsigned char %addressStyle)

Get the method in which IP Address is assigned to this device

Parameters

addressStyle – Method used. Current methods

- NONE = 0
- STATIC = 1
- DHCP = 2

Returns

Returns *common entity* return values

aErr **setNetworkConfiguration** (const unsigned char addressStyle)

Get the method in which IP Address is assigned to this device

Parameters

addressStyle – Method to use. See getNetworkConfiguration for addressStyle enumerations.

Returns

Returns *common entity* return values

aErr **getStaticIPv4Address** (unsigned char %buffer, const unsigned int bufferSize, unsigned int %unloadedLength)

Get the expected IPv4 address of this device, when networkConfiguration == STATIC

Note: The functional IPv4 address of The Module will differ if NetworkConfiguration != STATIC.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4Address** (const unsigned char %buffer, const unsigned int bufferSize)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC.

Parameters

- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4.

Returns

Returns *common entity* return values

aErr **getStaticIPv4Netmask** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get the expected IPv4 netmask of this device, when networkConfiguration == STATIC

Note: The functional IPv4 netmask of The Module will differ if NetworkConfiguration != STATIC.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4Netmask** (const unsigned char %buffer, const unsigned int bufferLength)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4.

Returns

Returns *common entity* return values

aErr **getStaticIPv4Gateway** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get the expected IPv4 gateway of this device, when networkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4Gateway** (const unsigned char %buffer, const unsigned int bufferLength)

Set the desired IPv4 gateway of this device, if NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4. setStaticIPv4Gateway([192, 168, 1, 1], 4) would equate with address “192.168.1.1”

Returns

Returns *common entity* return values

aErr **getIPv4Address** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get the effective IP address of this device.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **getIPv4Netmask** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get the effective IP netmask of this device.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **getIPv4Gateway** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get the effective IP gateway of this device.

Parameters

- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.
- **unloadedLength** – occupied bytes in buffer, Should be 4 post-call.

Returns

Returns *common entity* return values

aErr **setStaticIPv4DNSAddress** (const unsigned char %buffer, const unsigned int bufferLength)

Set IPv4 DNS Addresses (plural), if NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Total array length in bytes. Must be a multiple of 4.

Returns

Returns *common entity* return values

aErr **getStaticIPv4DNSAddress** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get IPv4 DNS addresses (plural), when NetworkConfiguration == STATIC

Parameters

- **buffer** – alias to an array of uint8_t[N][4]

- **bufferLength** – Maximum length of array, in bytes.
- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **getIPv4DNSAddress** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get effective IPv4 DNS addresses, for the current NetworkConfiguration

Parameters

- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Maximum length of array, in bytes.
- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **setHostname** (const unsigned char %buffer, const unsigned int bufferLength)

Set hostname that's requested when this device sends a DHCP request.

Parameters

- **buffer** – alias to an array of uint8_t[N]
- **bufferLength** – N, for N bytes.

Returns

Returns *common entity* return values

aErr **getHostname** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get hostname that's requested when this device sends a DHCP request.

Parameters

- **buffer** – alias to an array of uint8_t[N]
- **bufferLength** – N, for N bytes.
- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **getMACAddress** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Get the MAC address of the Ethernet interface.

Parameters

- **buffer** – alias to an array of uint8_t[6]
- **bufferLength** – length of buffer that's writeable, should be > 6.
- **unloadedLength** – Length of occupied bytes of buffer, after the call.

Returns

Returns *common entity* return values

aErr **setInterfacePort** (const unsigned char service, const unsigned short port)

Set the port of a TCPIP service on the device.

Parameters

- **service** – The index of the service to set the port for.
- **port** – The port to be used for the TCPIP server.

Returns

Returns *common entity* return values

aErr **getInterfacePort** (const unsigned char service, unsigned short %port)

Get the port of a TCPIP service on the device.

Parameters

- **service** – The index of the service to get the port for.
- **port** – The port of the TCPIP server.

Returns

Returns *common entity* return values

3.6.10 HDBaseT Class

See the *HDBaseT Entity* for generic information.

class **HDBaseTClass** : public Acroname::BrainStem2CLI::EntityClass

HDBaseTClass: This entity is only available on certain modules, and provides information on HDBaseT extenders.

Public Functions

HDBaseTClass ()

Constructors.

~HDBaseTClass ()

Destructor.

!HDBaseTClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdHDBASET index to be addressed.

aErr **getSerialNumber** (unsigned char %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Gets the serial number of the HDBaseT device (6 bytes)

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getFirmwareVersion** (unsigned int %firmwareVersion)

Gets the firmware version of the HDBaseT device

Parameters

firmwareVersion – A bit packet representation of the firmware version Major: Bits 24-31; Minor: Bits 16-23; Patch: Bits 8-15; Build: Bits 0-7

Returns

Returns *common entity* return values

aErr **getState** (unsigned int %state)

Gets the current state of the HDBaseT link

Parameters

state – Bit packeted representation of the state.

Returns

Returns *common entity* return values

aErr **getCableLength** (unsigned int %cableLength)

Gets the perceived cable length

Parameters

cableLength – Cable length in micro-meters

Returns

Returns *common entity* return values

aErr **getMSEA** (int %mseA)

Gets the Mean Squared Error (MSE) for channel A

Parameters

mseA – The current MSE for channel A in micro-dB

Returns

Returns *common entity* return values

aErr **getMSEB** (int %mseB)

Gets the Mean Squared Error (MSE) for channel B

Parameters

mseB – The current MSE for channel B in micro-dB

Returns

Returns *common entity* return values

aErr **getRetransmissionRate** (unsigned int %retransmissionRate)

Gets the number of successful messages between retransmission

Parameters

retransmissionRate – Instantaneous number of successful messages between retransmission. To be interpreted as: 1 / retransmissionRate for rate interpretation. If the value is 0, there have been no retransmissions, otherwise higher is better..

Returns

Returns *common entity* return values

aErr **getLinkUtilization** (unsigned int %linkUtilization)

Gets the current link utilization

Parameters

linkUtilization – Utilization in milli-percent

Returns

Returns *common entity* return values

aErr **getEncodingState** (unsigned char %encodingState)

Gets the current encoding state.

Parameters

encodingState – Signal modulation encoding type.

Returns

Returns *common entity* return values

aErr **getUSB2DeviceTree** (unsigned char %buffer, const unsigned int bufferSize, unsigned int %unloadedLength)

Gets the USB2 tree at the HDBaseT device.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getUSB3DeviceTree** (unsigned char %buffer, const unsigned int bufferSize, unsigned int %unloadedLength)

Gets the USB3 tree at the HDBaseT device.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getLinkRole** (unsigned char %role)

Gets the current link role In the case of “Auto” the getState API will provide the current role.

Parameters

role – Link role

Returns

Returns *common entity* return values

aErr **setLinkRole** (const unsigned char role)

Sets the active link role

Parameters

role – The role to be set.

Returns

Returns *common entity* return values

3.6.11 I2C Class

See the *I2C Entity* for generic information.

class **I2CClass** : public Acroname::BrainStem2CLI::EntityClass

I2CClass: Interface the I2C buses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entities bus.

Public Functions

I2CClass ()

Constructors.

~I2CClass ()

Destructor.

!I2CClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdI2C index to be addressed.

aErr **read** (const unsigned char address, const unsigned char readLength, unsigned char %buffer)

Read from a device on this I2C bus.

Parameters

- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **readLength** – The length of the data to read in bytes.
- **buffer** – The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAINSTEM_MAXPACKETBYTES - 5

Returns

Returns *common entity* return values

aErr **write** (const unsigned char address, const unsigned char bufferLength, const unsigned char %buffer)

Write to a device on this I2C bus.

Parameters

- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to write.

- **bufferLength** – The length of the data to write in bytes.
- **buffer** – The data to send to the device This array should be no larger than aBRAINSTEM_MAXPACKETBYTES - 5

Returns

Returns *common entity* return values

aErr **setPullup** (const unsigned char enable)

Set bus pull-up state. This call only works with stems that have software controlled pull-ups. Check the datasheet for more information. This parameter is saved when system.save is called.

Parameters

enable – true enables pull-ups false disables them.

Returns

Returns *common entity* return values

aErr **setSpeed** (const unsigned char speed)

Set I2C bus speed.

This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values: 1 - 100Khz 2 - 400Khz 3 - 1MHz

Parameters

speed – The speed setting value.

Returns

Returns *common entity* return values

aErr **getSpeed** (unsigned char %speed)

Get I2C bus speed.

This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values: 1 - 100Khz 2 - 400Khz 3 - 1MHz

Parameters

speed – The speed setting value.

Returns

Returns *common entity* return values

3.6.12 Mux Class

See the *Mux Entity* for generic information.

class **MuxClass** : public Acroname::BrainStem2CLI::EntityClass

MuxClass: A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to on or more outputs. One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs; some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

Public Functions

MuxClass ()

Constructors.

~MuxClass ()

Destructor.

!MuxClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdMUX index to be addressed.

aErr getEnable (unsigned char %enabled)

Get the mux enable/disable status

Parameters

enabled – true: mux is enabled, false: the mux is disabled.

Returns

Returns *common entity* return values

aErr setEnable (const unsigned char enable)

Enable the mux.

Parameters

enable – true: enables the mux for the selected channel.

Returns

Returns *common entity* return values

aErr getChannel (unsigned char %channel)

Get the current selected mux channel.

Parameters

channel – Indicates which channel is selected.

Returns

Returns *common entity* return values

aErr setChannel (const unsigned char channel)

Set the current mux channel.

Parameters

channel – mux channel to select.

Returns

Returns *common entity* return values

aErr **getChannelVoltage** (const unsigned char channel, int %microvolts)

Get the voltage of the indicated mux channel.

Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

- **channel** – The channel in which voltage was requested.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns *common entity* return values

aErr **getConfiguration** (int %config)

Get the configuration of the mux.

Parameters

config – integer representing the mux configuration either default, or split-mode.

Returns

Returns *common entity* return values

aErr **setConfiguration** (const int config)

Set the configuration of the mux.

Parameters

config – integer representing the mux configuration either muxConfig_default, or mux-Config_splitMode.

Returns

Returns *common entity* return values

aErr **getSplitMode** (int %splitMode)

Get the current split mode mux configuration.

Parameters

splitMode – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Returns

Returns *common entity* return values

aErr **setSplitMode** (const int splitMode)

Sets the mux's split mode configuration.

Parameters

splitMode – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Returns

Returns *common entity* return values

3.6.13 PoE Class

See the *PoE Entity* for generic information.

class **PoEClass** : public Acroname::BrainStem2CLI::EntityClass

PoEClass: This entity is only available on certain modules, and provides a Power over Ethernet control ability.

Public Functions

PoEClass ()

Constructors.

~PoEClass ()

Destructor.

!PoEClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdPOE index to be addressed.

aErr **getPairEnabled** (const unsigned char pair, unsigned char %enable)

Gets the current enable value of the indicated POE pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **enable** – 1 = Enabled; 0 = Disabled;

Returns

Returns *common entity* return values

aErr **setPairEnabled** (const unsigned char pair, const unsigned char enable)

Enables or disables the indicated POE pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **enable** – 1 = Enable port; 0 = Disable port.

Returns

Returns *common entity* return values

aErr **getPowerMode** (unsigned char %value)

Gets the power mode of the device

Parameters

value – The power mode (PD, PSE, Auto, Off).

Returns

Returns *common entity* return values

aErr **setPowerMode** (const unsigned char value)

Sets the power mode of the device

Parameters

value – The power mode (PD, PSE, Auto, Off).

Returns

Returns *common entity* return values

aErr **getPowerState** (unsigned char %value)

Gets the power state of the device

Parameters

value – The power state (PD, PSE, Off).

Returns

Returns *common entity* return values

aErr **getPairSourcingClass** (const unsigned char pair, unsigned char %value)

Gets the sourcing class for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The POE class being offered by the device (PSE).

Returns

Returns *common entity* return values

aErr **setPairSourcingClass** (const unsigned char pair, const unsigned char value)

Sets the sourcing class for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The POE class being offered by the device (PSE).

Returns

Returns *common entity* return values

aErr **getPairRequestedClass** (const unsigned char pair, unsigned char %value)

Gets the requested class for a given pair.

Parameters

- **pair** – Selects PoE pair to access

- 0 = Pair 1/2
- 1 = Pair 3/4
- **value** – The requested POE class by the device (PD).

Returns

Returns *common entity* return values

aErr **getPairDiscoveredClass** (const unsigned char pair, unsigned char %value)

Gets the discovered class for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The negotiated POE class by the device (PSE/PD).

Returns

Returns *common entity* return values

aErr **getPairDetectionStatus** (const unsigned char pair, unsigned char %value)

Gets detected status of the POE connection for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The current detected status of the pairs.

Returns

Returns *common entity* return values

aErr **getPairVoltage** (const unsigned char pair, int %microvolts)

Gets the Voltage for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **microvolts** – The voltage in microvolts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getPairCurrent** (const unsigned char pair, int %microamps)

Gets the Current for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

- **microamps** – The current in microamps (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getPairResistance** (const unsigned char pair, int %milliohms)

Gets the Resistance for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **milliohms** – The resistance in milliohms (1 == 1e-3Z).

Returns

Returns *common entity* return values

aErr **getPairCapacitance** (const unsigned char pair, int %nanofarads)

Gets the Capacitance for a given pair

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **nanofarads** – The capacitance in nanofarads (1 == 1e-9F).

Returns

Returns *common entity* return values

aErr **getPairPower** (const unsigned char pair, int %power)

Get the instantaneous power consumption for a given pair The equivalent of Voltage x Current

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **power** – Variable to be filled with the pairs power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getTotalPower** (int %power)

Gets the total instantaneous power consumption The equivalent of Pair1(Voltage x Current) + Pair2(Voltage x Current)

Parameters

power – Variable to be filled with the total POE power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPairAccumulatedPower** (const unsigned char pair, int %power)

Gets the accumulated power for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **power** – Variable to be filled with the total accumulated POE power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **setPairAccumulatedPower** (const unsigned char pair, const int power)

Sets the accumulated power for a given pair.

Parameters

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **power** – The power accumulator value to be set in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getTotalAccumulatedPower** (int %power)

Gets the total Accumulated Power

Parameters

power – Variable to be filled with the total accumulated POE power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **setTotalAccumulatedPower** (const int power)

Sets the total accumulated power

Parameters

power – The power accumulator value to be set in milli-watts (mW).

Returns

Returns *common entity* return values

3.6.14 Pointer Class

See the *Pointer Entity* for generic information.

```
class PointerClass : public Acroname::BrainStem2CLI::EntityClass
```

PointerClass: Allows access to the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the BrainStem scratchpad from the host, and provides a mechanism for allowing the host application and BrainStem relexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via `setOffset`. A read of a character short or int can be made from that cursor position.

In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

Public Functions

PointerClass ()

Constructors.

~PointerClass ()

Destructor.

!PointerClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdPOINTER index to be addressed.

aErr getOffset (unsigned short %offset)

Get the offset of the pointer

Parameters

offset – The value of the offset.

Returns

Returns *common entity* return values

aErr setOffset (const unsigned short offset)

Set the offset of the pointer

Parameters

offset – The value of the offset.

Returns

Returns *common entity* return values

aErr getMode (unsigned char %mode)

Get the mode of the pointer

Parameters

mode – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Returns

Returns *common entity* return values

aErr **setMode** (const unsigned char mode)

Set the mode of the pointer

Parameters

mode – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Returns

Returns *common entity* return values

aErr **getTransferStore** (unsigned char %handle)

Get the handle to the store.

Parameters

handle – The handle of the store.

Returns

Returns *common entity* return values

aErr **setTransferStore** (const unsigned char handle)

Set the handle to the store.

Parameters

handle – The handle of the store.

Returns

Returns *common entity* return values

aErr **initiateTransferToStore** (const unsigned char transferLength)

Transfer data to the store.

Parameters

transferLength – The length of the data transfer.

Returns

Returns *common entity* return values

aErr **initiateTransferFromStore** (const unsigned char transferLength)

Transfer data from the store.

Parameters

transferLength – The length of the data transfer.

Returns

Returns *common entity* return values

aErr **getChar** (unsigned char %value)

Get a char (1 byte) value from the pointer at this object's index, where elements are 1 byte long.

Parameters

value – The value of a single character (1 byte) stored in the pointer.

Returns

Returns *common entity* return values

aErr **setChar** (const unsigned char value)

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

Parameters

value – The single char (1 byte) value to be stored in the pointer.

ReturnsReturns *common entity* return values*aErr* **getShort** (unsigned short %value)

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long

Parameters**value** – The value of a single short (2 byte) stored in the pointer.**Returns**Returns *common entity* return values*aErr* **setShort** (const unsigned short value)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

Parameters**value** – The single short (2 byte) value to be set in the pointer.**Returns**Returns *common entity* return values*aErr* **getInt** (unsigned int %value)

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

Parameters**value** – The value of a single int (4 byte) stored in the pointer.**Returns**Returns *common entity* return values*aErr* **setInt** (const unsigned int value)

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

Parameters**value** – The single int (4 byte) value to be stored in the pointer.**Returns**Returns *common entity* return values

3.6.15 Port Class

See the *Port Entity* for generic information.

```
class PortClass : public Acroname::BrainStem2CLI::EntityClass
```

PortClass: The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

Public Functions

PortClass ()

Constructors.

~PortClass ()

Destructor.

!PortClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdPORT index to be addressed.

aErr getVbusVoltage (int %microvolts)

Gets the Vbus Voltage

Parameters

microvolts – The voltage in microvolts (1 == 1e-6V) currently present on Vbus.

Returns

Returns *common entity* return values

aErr getVbusCurrent (int %microamps)

Gets the Vbus Current

Parameters

microamps – The current in microamps (1 == 1e-6A) currently present on Vbus.

Returns

Returns *common entity* return values

aErr getVconnVoltage (int %microvolts)

Gets the Vconn Voltage

Parameters

microvolts – The voltage in microvolts (1 == 1e-6V) currently present on Vconn.

Returns

Returns *common entity* return values

aErr getVconnCurrent (int %microamps)

Gets the Vconn Current

Parameters

microamps – The current in microamps (1 == 1e-6A) currently present on Vconn.

Returns

Returns *common entity* return values

aErr getPowerMode (unsigned char %powerMode)

Gets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

powerMode – The current power mode.

Returns

Returns *common entity* return values

aErr **setPowerMode** (const unsigned char powerMode)

Sets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

powerMode – The power mode to be set.

Returns

Returns *common entity* return values

aErr **getEnabled** (unsigned char %enable)

Gets the current enable value of the port.

Parameters

enable – 1 = Fully enabled port; 0 = One or more disabled components.

Returns

Returns *common entity* return values

aErr **setEnabled** (const unsigned char enable)

Enables or disables the entire port.

Parameters

enable – 1 = Fully enable port; 0 = Fully disable port.

Returns

Returns *common entity* return values

aErr **getDataEnabled** (unsigned char %enable)

Gets the current enable value of the data lines. Sub-component (Data) of getEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataEnabled** (const unsigned char enable)

Enables or disables the data lines. Sub-component (Data) of setEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataHSEnabled** (unsigned char %enable)

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataHSEnabled** (const unsigned char enable)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataHS1Enabled** (unsigned char %enable)

Gets the current enable value of the High Speed A side (HSA) data lines. Sub-component of getDataHSEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataHS1Enabled** (const unsigned char enable)

Enables or disables the High Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataHS2Enabled** (unsigned char %enable)

Gets the current enable value of the High Speed B side (HSB) data lines. Sub-component of getDataHSEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataHS2Enabled** (const unsigned char enable)

Enables or disables the High Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSSEnabled** (unsigned char %enable)

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSSEnabled** (const unsigned char enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSS1Enabled** (unsigned char %enable)

Gets the current enable value of the Super Speed A side (SSA) data lines. Sub-component of getDataSSEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSS1Enabled** (const unsigned char enable)

Enables or disables the Super Speed A side (SSA) data lines. Sub-component of setDataEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSS2Enabled** (unsigned char %enable)

Gets the current enable value of the Super Speed B side (SSB) data lines. Sub-component of getDataSSEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSS2Enabled** (const unsigned char enable)

Enables or disables the Super Speed B side (SSB) data lines. Sub-component of setDataSSEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getPowerEnabled** (unsigned char %enable)

Gets the current enable value of the power lines. Sub-component (Power) of getEnabled.

Parameters

enable – 1 = Power enabled; 0 = Power disabled.

Returns

Returns *common entity* return values

aErr **setPowerEnabled** (const unsigned char enable)

Enables or Disables the power lines. Sub-component (Power) of setEnable.

Parameters

enable – 1 = Enable power; 0 = Disable disable.

Returns

Returns *common entity* return values

aErr **getDataRole** (unsigned char %dataRole)

Gets the Port Data Role.

Parameters

dataRole – The data role to be set. See datasheet for details.

Returns

Returns *common entity* return values

aErr **getVconnEnabled** (unsigned char %enable)

Gets the current enable value of the Vconn lines. Sub-component (Vconn) of getEnabled.

Parameters

enable – 1 = Vconn enabled; 0 = Vconn disabled.

Returns

Returns *common entity* return values

aErr **setVconnEnabled** (const unsigned char enable)

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

Parameters

enable – 1 = Enable Vconn lines; 0 = Disable Vconn lines.

Returns

Returns *common entity* return values

aErr **getVconn1Enabled** (unsigned char %enable)

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled.

Parameters

enable – 1 = Vconn1 enabled; 0 = Vconn1 disabled.

Returns

Returns *common entity* return values

aErr **setVconn1Enabled** (const unsigned char enable)

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

Parameters

enable – 1 = Enable Vconn1 lines; 0 = Disable Vconn1 lines.

Returns

Returns *common entity* return values

aErr **getVconn2Enabled** (unsigned char %enable)

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled.

Parameters

enable – 1 = Vconn2 enabled; 0 = Vconn2 disabled.

Returns

Returns *common entity* return values

aErr **setVconn2Enabled** (const unsigned char enable)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

Parameters

enable – 1 = Enable Vconn2 lines; 0 = Disable Vconn2 lines.

Returns

Returns *common entity* return values

aErr **getCCEnabled** (unsigned char %enable)

Gets the current enable value of the CC lines. Sub-component (CC) of getEnabled.

Parameters

enable – 1 = CC enabled; 0 = CC disabled.

Returns

Returns *common entity* return values

aErr **setCCEnabled** (const unsigned char enable)

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

Parameters

enable – 1 = Enable CC lines; 0 = Disable CC lines.

Returns

Returns *common entity* return values

aErr **getCC1Enabled** (unsigned char %enable)

Gets the current enable value of the CC1 lines. Sub-component of getCCEnabled.

Parameters

enable – 1 = CC1 enabled; 0 = CC1 disabled.

Returns

Returns *common entity* return values

aErr **setCC1Enabled** (const unsigned char enable)

Enables or disables the CC1 lines. Sub-component of setCCEnabled.

Parameters

enable – 1 = Enable CC1 lines; 0 = Disable CC1 lines.

Returns

Returns *common entity* return values

aErr **getCC2Enabled** (unsigned char %enable)

Gets the current enable value of the CC2 lines. Sub-component of getCCEnabled.

Parameters

enable – 1 = CC2 enabled; 0 = CC2 disabled.

Returns

Returns *common entity* return values

aErr **setCC2Enabled** (const unsigned char enable)

Enables or disables the CC2 lines. Sub-component of setCCEnabled.

Parameters

enable – 1 = Enable CC2 lines; 0 = Disable CC2 lines.

Returns

Returns *common entity* return values

aErr **getVoltageSetpoint** (unsigned int %value)

Gets the current voltage setpoint value for the port.

Parameters

value – the voltage setpoint of the port in uV.

Returns

Returns *common entity* return values

aErr **setVoltageSetpoint** (const unsigned int value)

Sets the current voltage setpoint value for the port.

Parameters

value – the voltage setpoint of the port in uV.

Returns

Returns *common entity* return values

aErr **getState** (unsigned int %state)

A bit mapped representation of the current state of the port. Reflects what the port IS which may differ from what was requested.

Parameters

state – Variable to be filled with the current state.

Returns

Returns *common entity* return values

aErr **getDataSpeed** (unsigned char %speed)

Gets the speed of the enumerated device.

Parameters

speed – Bit mapped value representing the devices speed. See “Devices” reference for details.

Returns

Returns *common entity* return values

aErr **getMode** (unsigned int %mode)

Gets current mode of the port

Parameters

mode – Bit mapped value representing the ports mode. See “Devices” reference for details.

Returns

Returns *common entity* return values

aErr **setMode** (const unsigned int mode)

Sets the mode of the port

Parameters

mode – Port mode to be set. See “Devices” documentation for details.

Returns

Returns *common entity* return values

aErr **getErrors** (unsigned int %errors)

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

Parameters

errors – Bit mapped field representing the current errors of the ports

Returns

Returns *common entity* return values

aErr **getCurrentLimit** (unsigned int %limit)

Gets the current limit of the port.

Parameters

limit – Variable to be filled with the limit in microAmps (uA).

Returns

Returns *common entity* return values

aErr **setCurrentLimit** (const unsigned int limit)

Sets the current limit of the port.

Parameters

limit – Current limit to be applied in microAmps (uA).

Returns

Returns *common entity* return values

aErr **getCurrentLimitMode** (unsigned char %mode)

Gets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters

mode – Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setCurrentLimitMode** (const unsigned char mode)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters

mode – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getAvailablePower** (unsigned int %power)

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Parameters

power – Variable to be filled with the available power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getAllocatedPower** (int %power)

Gets the currently allocated power This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Parameters

power – Variable to be filled with the allocated power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPowerLimit** (unsigned int %limit)

Gets the user defined power limit for the port.

Parameters

limit – Variable to be filled with the power limit in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **setPowerLimit** (const unsigned int limit)

Sets a user defined power limit for the port.

Parameters

limit – Power limit to be applied in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPowerLimitMode** (unsigned char %mode)

Gets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

mode – Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setPowerLimitMode** (const unsigned char mode)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

mode – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getName** (unsigned char %buffer, const unsigned int bufferSize, unsigned int %unloadedLength)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setName** (const unsigned char %buffer, const unsigned int bufferSize)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getCCCurrentLimit** (unsigned char %value)

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

Parameters

value – Variable to be filled with an enumerated representation of the CC Current limit.

- 0 = None
- 1 = Default (500/900mA)
- 2 = 1.5A
- 3 = 3.0A

Returns

Returns *common entity* return values

aErr **setCCCurrentLimit** (const unsigned char value)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

Parameters

value – Variable to be filled with an enumerated representation of the CC Current limit.

- 0 = None
- 1 = Default (500/900mA)
- 2 = 1.5A
- 3 = 3.0A

Returns

Returns *common entity* return values

aErr **getDataHSRoutingBehavior** (unsigned char %mode)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataHSRoutingBehavior** (const unsigned char mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getDataSSRoutingBehavior** (unsigned char %mode)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataSSRoutingBehavior** (const unsigned char mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getVbusAccumulatedPower** (int %milliwatthours)

Gets the Vbus Accumulated Power

Parameters

milliwatthours – The accumulated power on Vbus in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setVbusAccumulatedPower** (const int milliwatthours)

Sets the Vbus Accumulated Power

Parameters

milliwatthours – The accumulated power on Vbus in milliwatt-hours to be set.

Returns

Returns *common entity* return values

aErr **resetVbusAccumulatedPower** ()

Resets the Vbus Accumulated Power to zero.

Deprecated:

Replace with setVbusAccumulatedPower(0)

Returns

Returns *common entity* return values

aErr **getVconnAccumulatedPower** (int %milliwatthours)

Gets the Vconn Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setVconnAccumulatedPower** (const int milliwatthours)

Sets the Vconn Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn to be set.

Returns

Returns *common entity* return values

***aErr* resetVconnAccumulatedPower ()**

Resets the Vconn Accumulated Power to zero.

Deprecated:

Replace with setVconnAccumulatedPower(0)

Returns

Returns *common entity* return values

***aErr* setHSBoost (const unsigned char boost)**

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Parameters

boost – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

Returns

Returns *common entity* return values

***aErr* getHSBoost (unsigned char %boost)**

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Parameters

boost – An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

***aErr* getCC1State (unsigned short %value)**

Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Parameters

value – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:

- None = 0 = portCC1State_None
- Invalid = 1 = portCC1State_Invalid
- Rp (default) = 2 = portCC1State_RpDefault
- Rp (1.5A) = 3 = portCC1State_Rp1p5
- Rp (3A) = 4 = portCC1State_Rp3p0
- Rd = 5 = portCC1State_Rd
- Ra = 6 = portCC1State_Ra
- Managed by controller = 7 = portCC1State_Managed
- Unknown = 8 = portCC1State_Unknown

Returns

Returns *common entity* return values

aErr **getCC2State** (unsigned short %value)

Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Parameters

value – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:

- None = 0 = portCC2State_None
- Invalid = 1 = portCC2State_Invalid
- Rp (default) = 2 = portCC2State_RpDefault
- Rp (1.5A) = 3 = portCC2State_Rp1p5
- Rp (3A) = 4 = portCC2State_Rp3p0
- Rd = 5 = portCC2State_Rd
- Ra = 6 = portCC2State_Ra
- Managed by controller = 7 = portCC2State_Managed
- Unknown = 8 = portCC2State_Unknown

Returns

Returns *common entity* return values

aErr **getSBU1Voltage** (int %microvolts)

Get the voltage of SBU1 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getSBU2Voltage** (int %microvolts)

Get the voltage of SBU2 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC1Voltage** (int %microvolts)

Get the voltage of CC1 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC2Voltage** (int %microvolts)

Get the voltage of CC2 for a port.

Parameters

microvolts – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC1Current** (int %microamps)

Get the current through the CC1 for a port.

Parameters

microamps – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC2Current** (int %microamps)

Get the current through the CC2 for a port.

Parameters

microamps – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC1AccumulatedPower** (int %milliwatthours)

Gets the CC1 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setCC1AccumulatedPower** (const int milliwatthours)

Sets the CC1 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn to be set.

Returns

Returns *common entity* return values

aErr **getCC2AccumulatedPower** (int %milliwatthours)

Gets the CC2 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn in milliwatt-hours.

Returns

Returns *common entity* return values

aErr **setCC2AccumulatedPower** (const int milliwatthours)

Sets the CC2 Accumulated Power

Parameters

milliwatthours – The accumulated power on Vconn to be set.

Returns

Returns *common entity* return values

3.6.16 PowerDelivery Class

See the [PowerDelivery Entity](#) for generic information.

class **PowerDeliveryClass** : public Acroname::BrainStem2CLI::EntityClass

[PowerDeliveryClass](#): Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface. This Entity will allow you to directly access the vast landscape of PD.

Public Functions

PowerDeliveryClass ()

Constructors.

~PowerDeliveryClass ()

Destructor.

!PowerDeliveryClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdPOWERDELIVERY index to be addressed.

[aErr](#) **getConnectionState** (unsigned char %state)

Gets the current state of the connection in the form of an enumeration.

Parameters

state – Pointer to be filled with the current connection state.

Returns

Returns [common entity](#) return values

[aErr](#) **getNumberOfPowerDataObjects** (const unsigned char partner, const unsigned char powerRole, unsigned char %numRules)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

Parameters

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **numRules** – Variable to be filled with the number of PDOs.

Returns

Returns *common entity* return values

aErr **getPowerDataObject** (const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex, unsigned int %pdo)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

Parameters

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Variable to be filled with the requested power rule.

Returns

Returns *common entity* return values

aErr **setPowerDataObject** (const unsigned char powerRole, const unsigned char ruleIndex, const unsigned int pdo)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Power Data Object to be set.

Returns

Returns *common entity* return values

aErr **resetPowerDataObjectToDefault** (const unsigned char powerRole, const unsigned char ruleIndex)

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Returns

Returns *common entity* return values

aErr **getPowerDataObjectList** (unsigned int %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Gets all Power Data Objects (PDOs). Equivalent to calling *PowerDeliveryClass::getPowerDataObject()* on all partners, power roles, and index's.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled The order of which is:
 - Rules 1-7 Local Source
 - Rules 1-7 Local Sink
 - Rules 1-7 Partner Source
 - Rules 1-7 Partner Sink.
- **bufferLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled. On success this value should be 28 (7 rules * 2 partners * 2 power roles)

Returns

Returns *common entity* return values

aErr **getPowerDataObjectEnabled** (const unsigned char powerRole, const unsigned char ruleIndex, unsigned char %enabled)

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – Variable to be filled with enabled state.

Returns

Returns *common entity* return values

aErr **setPowerDataObjectEnabled** (const unsigned char powerRole, const unsigned char ruleIndex, const unsigned char enabled)

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – The state to be set.

Returns

Returns *common entity* return values

aErr **getPowerDataObjectEnabledList** (const unsigned char powerRole, unsigned char %enabledList)

Gets all Power Data Object enables for a given power role. Equivalent of calling *PowerDeliveryClass::getPowerDataObjectEnabled()* for all indexes.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **enabledList** – Variable to be filled with a mapped representation of the enabled PDOs for a given power role. Values align with a given rule index (bits 1-7, bit 0 is invalid)

Returns

Returns *common entity* return values

aErr **getRequestDataObject** (const unsigned char partner, unsigned int %rdo)

Gets the current Request Data Object (RDO) for a given partner. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

Parameters

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **rdo** – Variable to be filled with the current RDO. Zero indicates the RDO is not active.

Returns

Returns *common entity* return values

aErr **setRequestDataObject** (const unsigned int rdo)

Sets the current Request Data Object (RDO) for a given partner. Only the local partner can be changed. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

Parameters

rdo – Request Data Object to be set.

Returns

Returns *common entity* return values

aErr **getLinkState** (unsigned int %state)

Gets the current state of the connection in the form of a bitmask.

Parameters

state – Pointer to be filled with the current connection state bits.

Returns

Returns *common entity* return values

aErr **getAttachTimeElapsed** (unsigned int %secondsAndMicroseconds, const unsigned int
bufferLength, unsigned int %unloadedLength)

Gets the length of time that the port has been in the attached state. Returned as a list of two unsigned integers, first seconds, then microseconds.

Parameters

- **secondsAndMicroseconds** – Pointer to list of unsigned integers to fill with attach time elapsed
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getPowerRoleCapabilities** (unsigned char %powerRole)

Gets the power roles that may be advertised by the local partner. (CC Strapping).

Parameters

powerRole – Variable to be filled with the power role

- None = 0 = pdPowerRoleCapabilities_None
- Source = 1 = pdPowerRoleCapabilities_Source
- Sink = 2 = pdPowerRoleCapabilities_Sink
- Source/Sink = 3 = pdPowerRoleCapabilities_DualRole (Dual Role Port)

Returns

Returns *common entity* return values

aErr **getPowerRole** (unsigned char %powerRole)

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

Parameters

powerRole – Variable to be filled with the power role

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Returns

Returns *common entity* return values

aErr **setPowerRole** (const unsigned char powerRole)

Set the current power role to be advertised by the Local partner. (CC Strapping).

Parameters

powerRole – Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Returns

Returns *common entity* return values

aErr **getPowerRolePreferred** (unsigned char %powerRole)

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

Parameters

powerRole – Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Returns

Returns *common entity* return values

aErr **setPowerRolePreferred** (const unsigned char powerRole)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

Parameters

powerRole – Value to be applied.

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Returns

Returns *common entity* return values

aErr **getDataRoleCapabilities** (unsigned char %dataRole)

Gets the data roles that may be advertised by the local partner.

Parameters

dataRole – Variable to be filled with the data role

- None = 0 = pdDataRoleCapabilities_None
- DFP = 1 = pdDataRoleCapabilities_DFP
- UFP = 2 = pdDataRoleCapabilities_UFP
- DFP/UFP = 3 = pdDataRoleCapabilities_DualRole (Dual Role Port)

Returns

Returns *common entity* return values

aErr **getCableVoltageMax** (unsigned char %maxVoltage)

Gets the maximum voltage capability reported by the e-mark of the attached cable.

Parameters

maxVoltage – Variable to be filled with an enumerated representation of voltage.

- Unknown/Unattached (0)
- 20 Volts DC (1)
- 30 Volts DC (2)
- 40 Volts DC (3)
- 50 Volts DC (4)

Returns

Returns *common entity* return values

aErr **getCableCurrentMax** (unsigned char %maxCurrent)

Gets the maximum current capability report by the e-mark of the attached cable.

Parameters

maxCurrent – Variable to be filled with an enumerated representation of current.

- Unknown/Unattached (0)
- 3 Amps (1)
- 5 Amps (2)

Returns

Returns *common entity* return values

aErr **getCableSpeedMax** (unsigned char %maxSpeed)

Gets the maximum data rate capability reported by the e-mark of the attached cable.

Parameters

maxSpeed – Variable to be filled with an enumerated representation of data speed.

- Unknown/Unattached (0)
- USB 2.0 (1)
- USB 3.2 gen 1 (2)
- USB 3.2 / USB 4 gen 2 (3)
- USB 4 gen 3 (4)

Returns

Returns *common entity* return values

aErr **getCableType** (unsigned char %type)

Gets the cable type reported by the e-mark of the attached cable.

Parameters

type – Variable to be filled with an enumerated representation of the cable type.

- Invalid, no e-mark and not Vconn powered (0)
- Passive cable with e-mark (1)
- Active cable (2)

Returns

Returns *common entity* return values

aErr **getCableOrientation** (unsigned char %orientation)

Gets the current orientation being used for PD communication

Parameters

orientation – Variable filled with an enumeration of the orientation.

- Unconnected (0)
- CC1 (1)
- CC2 (2)

Returns

Returns *common entity* return values

aErr **request** (const unsigned char request)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

Parameters

request – Request to be issued to the remote partner

- pdRequestHardReset (0)
- pdRequestSoftReset (1)
- pdRequestDataReset (2)
- pdRequestPowerRoleSwap (3)
- pdRequestPowerFastRoleSwap (4)
- pdRequestDataRoleSwap (5)
- pdRequestVconnSwap (6)
- pdRequestSinkGoToMinimum (7)
- pdRequestRemoteSourcePowerDataObjects (8)
- pdRequestRemoteSinkPowerDataObjects (9)
- pdRequestRemoteSourceExtendedCapabilities (10)
- pdRequestRemoteSinkExtendedCapabilities (11)
- pdRequestStatus (12)
- pdRequestPPSStatus (13)
- pdRequestBatteryCapabilities (14)
- pdRequestBatteryStatus (15)
- pdRequestManufacturerInfoSop (16)
- pdRequestManufacturerInfoSopp (17)
- pdRequestManufacturerInfoSoppp (18)
- pdRequestDiscoverIdentitySop (19)
- pdRequestDiscoverIdentitySopp (20)
- pdRequestDiscoverIdentitySoppp (21)
- pdRequestRevision (22)
- pdRequestSourceInfo (23)
- pdRequestCountryCode (24)
- pdRequestCountryInfo (25)
- pdRequestRemoteSourceEPRCapabilities (26)
- pdRequestRemoteSinkEPRCapabilities (27)

Returns

Returns *common entity* return values

aErr **requestStatus** (unsigned int %status)

Gets the status of the last request command sent.

Parameters

status – Variable to be filled with the status

Returns

Returns *common entity* return values

aErr **getOverride** (unsigned int %overrides)

Gets the current enabled overrides

Parameters

overrides – Bit mapped representation of the current override configuration.

Returns

Returns *common entity* return values

aErr **setOverride** (const unsigned int overrides)

Sets the current enabled overrides

Parameters

overrides – Overrides to be set in a bit mapped representation.

Returns

Returns *common entity* return values

aErr **getProtocolRevisionMax** (unsigned char %maxProtocolRevision)

Gets the configured maximum USB Power Delivery protocol revision used as a negotiation ceiling (pdProtocolRevision2 = 2, pdProtocolRevision3 = 3). This is unrelated to hardware or product revision.

Parameters

maxProtocolRevision – Filled with pdProtocolRevision2 (2) or pdProtocolRevision3 (3).

Returns

Returns *common entity* return values

aErr **setProtocolRevisionMax** (const unsigned char maxProtocolRevision)

Sets the maximum USB Power Delivery protocol revision for negotiation. Only pdProtocolRevision2 (2) and pdProtocolRevision3 (3) are valid API values.

Parameters

maxProtocolRevision – pdProtocolRevision2 (2) or pdProtocolRevision3 (3).

Returns

Returns *common entity* return values

aErr **getFlagMode** (const unsigned char flag, unsigned char %mode)

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Parameters

- **flag** – Flag/Advertisement to be modified
- **mode** – Variable to be filled with the current mode.
 - Disabled (0)
 - Enabled (1)
 - Auto (2) default

Returns

Returns *common entity* return values

aErr **setFlagMode** (const unsigned char flag, const unsigned char mode)

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Parameters

- **flag** – Flag/Advertisement to be modified
- **mode** – Value to be applied.
 - Disabled (0)
 - Enabled (1)
 - Auto (2) default

Returns

Returns *common entity* return values

aErr **getPeakCurrentConfiguration** (unsigned char %configuration)

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Parameters

configuration – An enumerated value referring to the current configuration.

- Allowable values are 0 - 4

Returns

Returns *common entity* return values

aErr **setPeakCurrentConfiguration** (const unsigned char configuration)

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Parameters

configuration – An enumerated value referring to the configuration to be set

- Allowable values are 0 - 4

Returns

Returns *common entity* return values

aErr **getFastRoleSwapCurrent** (unsigned char %swapCurrent)

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

Parameters

swapCurrent – An enumerated value referring to current swap value.

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

Returns

Returns *common entity* return values

aErr **setFastRoleSwapCurrent** (const unsigned char swapCurrent)

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

Parameters

swapCurrent – An enumerated value referring to value to be set.

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

Returns

Returns *common entity* return values

Public Static Functions

static *aErr* **packDataObjectAttributes** (unsigned char %attributes, const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex)

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

Parameters

- **attributes** – Variable to be filled with packed values.
- **partner** – Indicates which side of the PD connection.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

Returns

Returns *common entity* return values

static *aErr* **unpackDataObjectAttributes** (const unsigned char attributes, unsigned char %partner, unsigned char %powerRole, unsigned char %ruleIndex)

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

Parameters

- **attributes** – Variable to be filled with packed values.
- **partner** – Indicates which side of the PD connection.
 - Local = 0 = powerdeliveryPartnerLocal

- Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

Returns

Returns *common entity* return values

3.6.17 RCServo Class

See the *RCServo Entity* for generic information.

class **RCServoClass** : public Acroname::BrainStem2CLI::EntityClass

RCServoClass: Interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

Public Functions

RCServoClass ()

Constructors.

~RCServoClass ()

Destructor.

!RCServoClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdSERVO index to be addressed.

aErr **setEnabled** (const unsigned char enable)

Enable the servo channel

Parameters

enable – The state to be set. 0 is disabled, 1 is enabled.

Returns

Returns *common entity* return values

aErr **getEnabled** (unsigned char %enable)

Get the enable status of the servo channel.

Parameters

enable – The current enable status of the servo entity. 0 is disabled, 1 is enabled.

Returns

Returns *common entity* return values

aErr **setPosition** (const unsigned char position)

Set the position of the servo channel

Parameters

position – The position to be set. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns

Returns *common entity* return values

aErr **getPosition** (unsigned char %position)

Get the position of the servo channel

Parameters

position – The current position of the servo channel. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns

Returns *common entity* return values

aErr **setReverse** (const unsigned char reverse)

Set the output to be reversed on the servo channel

Parameters

reverse – Reverses the value set by “setPosition”. For example, if the position is set to 64 (1ms pulse) the output will now be 192 (2ms pulse), however “getPosition” will return the set value of 64. 0 = not reversed, 1 = reversed.

Returns

Returns *common entity* return values

aErr **getReverse** (unsigned char %reverse)

Get the reverse status of the servo channel

Parameters

reverse – The current reverse status of the servo entity. 0 = not reversed, 1 = reversed.

Returns

Returns *common entity* return values

3.6.18 Rail Class

See the *Rail Entity* for generic information.

```
class RailClass : public Acroname::BrainStem2CLI::EntityClass
```

RailClass: Provides power rail functionality on certain modules. The *RailClass* can be used to control power to downstream devices. It has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

Public Functions

RailClass()

Constructors.

~RailClass()

Destructor.

!RailClass()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdRAIL index to be addressed.

aErr getCurrent (int %microamps)

Get the rail current.

Parameters

microamps – The current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr setCurrentSetpoint (const int microamps)

Set the rail supply current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Parameters

microamps – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

Returns

Returns *common entity* return values

aErr getCurrentSetpoint (int %microamps)

Get the rail setpoint current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Parameters

microamps – The current in micro-amps (1 == 1e-6A) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setCurrent interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr setCurrentLimit (const int microamps)

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

Parameters

microamps – The current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCurrentLimit** (int %microamps)

Get the rail current limit setting. (Check product datasheet to see if this feature is available)

Parameters

microamps – The current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getTemperature** (int %microcelsius)

Get the rail temperature.

Parameters

microcelsius – The measured temperature associated with the rail in micro-Celsius (1 == 1e-6°C). The temperature may be associated with the module's internal rail circuitry or an externally connected temperature sensors. Refer to the module datasheet for definition of the temperature measurement location and specific capabilities.

Returns

Returns *common entity* return values

aErr **getEnable** (unsigned char %enable)

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Parameters

enable – true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage

Returns

Returns *common entity* return values

aErr **setEnable** (const unsigned char enable)

Set the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Parameters

enable – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

Returns

Returns *common entity* return values

aErr **getVoltage** (int %microvolts)

Get the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setVoltageSetpoint** (const int microvolts)

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

Returns

Returns *common entity* return values

aErr **getVoltageSetpoint** (int %microvolts)

Get the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setVoltageMinLimit** (const int microvolts)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getVoltageMinLimit** (int %microvolts)

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **setVoltageMaxLimit** (const int microvolts)

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getVoltageMaxLimit** (int %microvolts)

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microvolts – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getPower** (int %microwatts)

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

microwatts – The power in micro-watts (1 == 1e-6W) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the `setPower` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setPowerSetpoint** (const int microwatts)

Set the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

microwatts – The power in micro-watts (1 == 1e-6W) to be supplied by the rail.

Returns

Returns *common entity* return values

aErr **getPowerSetpoint** (int %microwatts)

Get the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

microwatts – The power in micro-watts (1 == 1e-6W) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the `setPower` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setPowerLimit** (const int microwatts)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microwatts – The power in micro-watts (1 == 1e-6W).

Returns

Returns *common entity* return values

aErr **getPowerLimit** (int %microwatts)

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

microwatts – The power in micro-watts (1 == 1e-6W).

Returns

Returns *common entity* return values

aErr **getResistance** (int %microohms)

Get the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

microohms – The resistance in micro-ohms (1 == 1e-6Ohms) currently drawn by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the `setResistance` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setResistanceSetpoint** (const int microohms)

Set the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

microohms – The resistance in micro-ohms (1 == 1e-6Ohms) to be drawn by the rail.

Returns

Returns *common entity* return values

aErr **getResistanceSetpoint** (int %microohms)

Get the rail setpoint resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

microohms – The resistance in micro-ohms (1 == 1e-6Ohms) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setResistance interface. Refer to the module datasheet to determine if this is a measured or stored value.

Returns

Returns *common entity* return values

aErr **setKelvinSensingEnable** (const unsigned char enable)

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

enable – enable or disable kelvin sensing.

Returns

Returns *common entity* return values

aErr **getKelvinSensingEnable** (unsigned char %enable)

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

enable – Kelvin sensing is enabled or disabled.

Returns

Returns *common entity* return values

aErr **getKelvinSensingState** (unsigned char %state)

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

state – Kelvin sensing is enabled or disabled.

Returns

Returns *common entity* return values

aErr **setOperationalMode** (const unsigned char mode)

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

Parameters

mode – The operational mode to employ.

Returns

Returns *common entity* return values

aErr **getOperationalMode** (unsigned char %mode)

Determine the current operational mode of the system. Refer to the module datasheet for definition of the rail operational mode capabilities.

Parameters

mode – The current operational mode setting.

Returns

Returns *common entity* return values

aErr **getOperationalState** (unsigned int %state)

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

Parameters

state – The current operational state, hardware configuration, faults, and operating mode.

Returns

Returns *common entity* return values

aErr **clearFaults** ()

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

Returns

Returns *common entity* return values

3.6.19 Relay Class

See the *Relay Entity* for generic information.

class **RelayClass** : public Acroname::BrainStem2CLI::EntityClass

RelayClass: Interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

Public Functions

RelayClass ()

Constructors.

~RelayClass ()

Destructor.

!RelayClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdRELAY index to be addressed.

aErr **setEnabled** (const unsigned char enable)

Set the enable/disable state.

Parameters

enable – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns *common entity* return values

aErr **getEnabled** (unsigned char %enabled)

Get the state.

Parameters

enabled – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns *common entity* return values

aErr **getVoltage** (int %microvolts)

Get the scaled micro volt value with reference to ground.

Note: Not all modules provide 32 bits of accuracy. Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

microvolts – 32 bit signed integer (in micro Volts) based on the boards ground and reference voltages.

Returns

Returns *common entity* return values

3.6.20 Signal Class

See the *Signal Entity* for generic information.

```
class SignalClass : public Acroname::BrainStem2CLI::EntityClass
```

SignalClass: Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Public Functions

SignalClass()

Constructors.

~SignalClass()

Destructor.

!SignalClass()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdSIGNAL index to be addressed.

aErr setEnable (const unsigned char enable)

Enable/Disable the signal output.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr getEnable (unsigned char %enable)

Get the Enable/Disable of the signal.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr setInvert (const unsigned char invert)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert – True to invert, false for normal mode.

Returns

Returns *common entity* return values

aErr getInvert (unsigned char %invert)

Get the invert status the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert – True to invert, false for normal mode.

Returns

Returns *common entity* return values

aErr **setT3Time** (const unsigned int t3_nsec)

Set the signal period or T3 in nanoseconds.

Parameters

t3_nsec – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Returns

Returns *common entity* return values

aErr **getT3Time** (unsigned int %t3_nsec)

Get the signal period or T3 in nanoseconds.

Parameters

t3_nsec – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Returns

Returns *common entity* return values

aErr **setT2Time** (const unsigned int t2_nsec)

Set the signal active period or T2 in nanoseconds.

Parameters

t2_nsec – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Returns

Returns *common entity* return values

aErr **getT2Time** (unsigned int %t2_nsec)

Get the signal active period or T2 in nanoseconds.

Parameters

t2_nsec – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Returns

Returns *common entity* return values

3.6.21 Store Class

See the *Store Entity* for generic information.

```
class StoreClass : public Acroname::BrainStem2CLI::EntityClass
```

StoreClass: The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

Public Functions

StoreClass ()

Constructors.

~StoreClass ()

Destructor.

!StoreClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdSTORE index to be addressed.

aErr getSlotState (const unsigned char slot, unsigned char %state)

Get slot state.

Parameters

- **slot** – The slot number.
- **state** – true: enabled, false: disabled.

Returns

Returns *common entity* return values

aErr loadSlot (const unsigned char slot, const unsigned char %buffer, const unsigned short
bufferLength)

Load the slot.

Parameters

- **slot** – The slot number.
- **buffer** – The data.
- **bufferLength** – The data length.

Returns

Returns *common entity* return values

aErr unloadSlot (const unsigned char slot, const unsigned int bufferLength, unsigned char %buffer,
unsigned int %unloadedLength)

Unload the slot data.

Parameters

- **slot** – The slot number.
- **bufferLength** – The length of buffer buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **buffer** – Byte array that the unloaded data will be placed into.
- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than dataLength.

Returns

Returns *common entity* return values

aErr **slotEnable** (const unsigned char slot)

Enable slot.

Parameters

slot – The slot number.

Returns

Returns *common entity* return values

aErr **slotDisable** (const unsigned char slot)

Disable slot.

Parameters

slot – The slot number.

Returns

Returns *common entity* return values

aErr **getSlotCapacity** (const unsigned char slot, unsigned int %capacity)

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold.

Parameters

- **slot** – The slot number.
- **capacity** – The slot capacity.

Returns

Returns *common entity* return values

aErr **getSlotSize** (const unsigned char slot, unsigned int %size)

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes.

Parameters

- **slot** – The slot number.
- **size** – The slot size.

Returns

Returns *common entity* return values

aErr **getSlotLocked** (const unsigned char slot, unsigned char %lock)

Gets the current lock state of the slot Allows for write protection on a slot.

Parameters

- **slot** – The slot number
- **lock** – Variable to be filled with the locked state.

Returns

Returns *common entity* return values

aErr **setSlotLocked** (const unsigned char slot, const unsigned char lock)

Sets the locked state of the slot Allows for write protection on a slot.

Parameters

- **slot** – The slot number
- **lock** – state to be set.

Returns

Returns *common entity* return values

3.6.22 System Class

See the *System Entity* for generic information.

class **SystemClass** : public Acroname::BrainStem2CLI::EntityClass

SystemClass: The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc.

Public Functions

SystemClass ()

Constructors.

~SystemClass ()

Destructor.

!SystemClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdSYSTEM index to be addressed.

aErr **getModule** (unsigned char %address)

Get the current address the module uses on the BrainStem network.

Parameters

address – The address the module is using on the BrainStem network.

Returns

Returns *common entity* return values

aErr **getModuleBaseAddress** (unsigned char %address)

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

Parameters

address – The address the module is using on the BrainStem network.

Returns

Returns *common entity* return values

aErr **setRouter** (const unsigned char address)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

Parameters

address – The router address to be used.

Returns

Returns *common entity* return values

aErr **getRouter** (unsigned char %address)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

Parameters

address – The address.

Returns

Returns *common entity* return values

aErr **setHBInterval** (const unsigned char interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

Parameters

interval – The desired heartbeat delay.

Returns

Returns *common entity* return values

aErr **getHBInterval** (unsigned char %interval)

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

Parameters

interval – The current heartbeat delay.

Returns

Returns *common entity* return values

aErr **setLED** (const unsigned char ledOn)

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

Parameters

ledOn – true: turn the LED on, false: turn LED off.

Returns

Returns *common entity* return values

aErr **getLED** (unsigned char %ledOn)

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

Parameters

ledOn – true: LED on, false: LED off.

Returns

Returns *common entity* return values

aErr **setLEDMaxBrightness** (const unsigned char brightness)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Parameters

brightness – Brightness value relative to 255

Returns

Returns *common entity* return values

aErr **getLEDMaxBrightness** (unsigned char %brightness)

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum).

Parameters

brightness – Brightness value relative to 255

Returns

Returns *common entity* return values

aErr **setBootSlot** (const unsigned char slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

Parameters

slot – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

Returns

Returns *common entity* return values

aErr **getBootSlot** (unsigned char %slot)

Get the store slot which is mapped when the module boots.

Parameters

slot – The slot number in aSTORE_INTERNAL that is mapped after the module boots.

Returns

Returns *common entity* return values

aErr **getVersion** (unsigned int %build)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

Parameters

build – The build version date code.

Returns

Returns *common entity* return values

aErr **getBuild** (unsigned int %build)

Get the modules firmware build number. The build number is a unique hash assigned to a specific firmware.

Parameters

build – Variable to be filled with build.

Returns

Returns *common entity* return values

aErr **getModel** (unsigned char %model)

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem

model codes". Other codes are be used by Acroname for proprietary module types.

Parameters

model – The module's model enumeration.

Returns

Returns *common entity* return values

aErr **getHardwareVersion** (unsigned int %hardwareVersion)

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

Parameters

hardwareVersion – The module's hardware version information.

Returns

Returns *common entity* return values

aErr **getSerialNumber** (unsigned int %serialNumber)

Get the module's serial number. The serial number is a unique 32 bit integer which is usually communicated in hexadecimal format.

Parameters

serialNumber – The module's serial number.

Returns

Returns *common entity* return values

aErr **save** ()

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address.

Returns

Returns *common entity* return values

aErr **reset** ()

Reset the system. A return value of aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

Returns

Returns *common entity* return values

***aErr* logEvents ()**

Saves system log events to a slot defined by the module (usually ram slot 0).

Returns

Returns *common entity* return values

***aErr* getUptime (unsigned int %uptimeCounter)**

Get the module's accumulated uptime in minutes

Parameters

uptimeCounter – The module's accumulated uptime in minutes.

Returns

Returns *common entity* return values

***aErr* getTemperature (int %temperature)**

Get the module's current temperature in micro-C

Parameters

temperature – The module's system temperature in micro-C

Returns

Returns *common entity* return values

***aErr* getMinimumTemperature (int %minTemperature)**

Get the module's minimum temperature ever recorded in micro-C (uC). This value will persists through a power cycle.

Parameters

minTemperature – The module's minimum system temperature in micro-C

Returns

Returns *common entity* return values

***aErr* getMaximumTemperature (int %maxTemperature)**

Get the module's maximum temperature ever recorded in micro-C (uC). This value will persists through a power cycle.

Parameters

maxTemperature – The module's maximum system temperature in micro-C

Returns

Returns *common entity* return values

***aErr* getInputVoltage (unsigned int %inputVoltage)**

Get the module's input voltage.

Parameters

inputVoltage – The module's input voltage reported in microvolts.

Returns

Returns *common entity* return values

***aErr* getInputCurrent (unsigned int %inputCurrent)**

Get the module's input current.

Parameters

inputCurrent – The module's input current reported in microamps.

Returns

Returns *common entity* return values

***aErr* getModuleHardwareOffset** (unsigned char %offset)

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

Parameters

offset – The module address offset.

Returns

Returns *common entity* return values

***aErr* setModuleSoftwareOffset** (const unsigned char address)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

***aErr* getModuleSoftwareOffset** (unsigned char %address)

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

***aErr* getRouterAddressSetting** (unsigned char %address)

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

***aErr* routeToMe** (const unsigned char enable)

Enables/Disables the route to me function. This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

Parameters

enable – Enable or disable of the route to me function 1 = enable.

Returns

Returns *common entity* return values

aErr **getPowerLimit** (unsigned int %power)

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

Parameters

power – The available power in milli-Watts (mW, 1 t)

Returns

Returns *common entity* return values

aErr **getPowerLimitMax** (unsigned int %power)

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters

power – Variable to be filled with the power limit in milli-Watts (mW)

Returns

Returns *common entity* return values

aErr **setPowerLimitMax** (const unsigned int power)

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters

power – Limit in milli-Watts (mW) to be set.

Returns

Returns *common entity* return values

aErr **getPowerLimitState** (unsigned int %state)

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit().

Parameters

state – Variable to be filled with the state.

Returns

Returns *common entity* return values

aErr **getUnregulatedVoltage** (int %voltage)

Gets the voltage present at the unregulated port.

Parameters

voltage – Variable to be filled with the voltage in micro-Volts (uV).

Returns

Returns *common entity* return values

aErr **getUnregulatedCurrent** (int %current)

Gets the current passing through the unregulated port.

Parameters

current – Variable to be filled with the current in micro-Amps (uA).

Returns

Returns *common entity* return values

aErr **getInputPowerSource** (unsigned char %source)

Provides the source of the current power source in use.

Parameters

source – Variable to be filled with enumerated representation of the source.

Returns

Returns *common entity* return values

aErr **getInputPowerBehavior** (unsigned char %behavior)

Gets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Parameters

behavior – Variable to be filled with an enumerated value representing behavior.

Returns

Returns *common entity* return values

aErr **setInputPowerBehavior** (const unsigned char behavior)

Sets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Parameters

behavior – An enumerated representation of behavior to be set.

Returns

Returns *common entity* return values

aErr **getInputPowerBehaviorConfig** (unsigned int %buffer, const unsigned int bufferSize, unsigned int %unloadedLength)

Gets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setInputPowerBehaviorConfig** (const unsigned int %buffer, const unsigned int bufferSize)

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getName** (unsigned char %buffer, const unsigned int bufferSize, unsigned int %unloadedLength)

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled

- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setName** (const unsigned char %buffer, const unsigned int bufferLength)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **resetDeviceToFactoryDefaults** ()

Resets the device to it factory default configuration.

Returns

Returns *common entity* return values

aErr **getLinkInterface** (unsigned char %linkInterface)

Gets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters

linkInterface – Variable to be filled with an enumerated value representing interface.

- 0 = Auto= systemLinkAuto
- 1 = Control Port = systemLinkUSBControl
- 2 = Hub Upstream Port = systemLinkUSBHub

Returns

Returns *common entity* return values

aErr **setLinkInterface** (const unsigned char linkInterface)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters

linkInterface – An enumerated representation of interface to be set.

- 0 = Auto= systemLinkAuto
- 1 = Control Port = systemLinkUSBControl
- 2 = Hub Upstream Port = systemLinkUSBHub

Returns

Returns *common entity* return values

aErr **getErrors** (unsigned int %errors)

Gets any system level errors. Calling this function will clear the current errors. If the error persists it will be set again.

Parameters

errors – Bit mapped field representing the devices errors

Returns

Returns *common entity* return values

aErr **getProtocolFeatures** (unsigned int %features)

Gets the firmware protocol features

Parameters

features – Value representing the firmware protocol features

Returns

Returns *common entity* return values

3.6.23 Temperature Class

See the *Temperature Entity* for generic information.

class **TemperatureClass** : public Acroname::BrainStem2CLI::EntityClass

TemperatureClass: This entity is only available on certain modules, and provides a temperature reading in microcelsius.

Public Functions

TemperatureClass ()

Constructors.

~TemperatureClass ()

Destructor.

!TemperatureClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdTEMPERATURE index to be addressed.

aErr **getValue** (int %temp)

Get the modules temperature in micro-C

Parameters

temp – The temperature in micro-Celsius (1 == 1e-6C).

Returns

Returns *common entity* return values

aErr **getValueMin** (int %minTemp)

Get the module's minimum temperature in micro-C since the last power cycle.

Parameters

minTemp – The module's minimum temperature in micro-C

Returns

Returns *common entity* return values

aErr **getValueMax** (int %maxTemp)

Get the module's maximum temperature in micro-C since the last power cycle.

Parameters

maxTemp – The module's maximum temperature in micro-C

Returns

Returns *common entity* return values

3.6.24 Timer Class

See the *Timer Entity* for generic information.

class **TimerClass** : public Acroname::BrainStem2CLI::EntityClass

TimerClass: The Timer Class provides access to a simple scheduler. The timer can set to fire only once, or to repeat at a certain interval. Additionally, a timer entity can execute custom Reflex routines upon firing.

Public Functions

TimerClass ()

Constructors.

~TimerClass ()

Destructor.

!TimerClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdTIMER index to be addressed.

aErr **getExpiration** (unsigned int %usecDuration)

Get the currently set expiration time in microseconds. This is not a "live" timer. That is, it shows the expiration time originally set with setExpiration; it does not "tick down" to show the time remaining before expiration.

Parameters

usecDuration – The timer expiration duration in microseconds.

Returns

Returns *common entity* return values

aErr **setExpiration** (const unsigned int usecDuration)

Set the expiration time for the timer entity. When the timer expires, it will fire the associated timer[index]() reflex.

Parameters

usecDuration – The duration before timer expiration in microseconds.

Returns

Returns *common entity* return values

aErr **getMode** (unsigned char %mode)

Get the mode of the timer which is either single or repeat mode.

Parameters

mode – The mode of the time. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Returns

Returns *common entity* return values

aErr **setMode** (const unsigned char mode)

Set the mode of the timer which is either single or repeat mode.

Parameters

mode – The mode of the timer. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Returns

Returns *common entity* return values

3.6.25 UART Class

See the *UART Entity* for generic information.

class **UARTClass** : public Acroname::BrainStem2CLI::EntityClass

UARTClass: A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port. The UART Class allows the enabling and disabling of the UART data lines.

Public Functions

UARTClass ()

Constructors.

~UARTClass ()

Destructor.

!UARTClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdUART index to be addressed.

aErr **setEnable** (const unsigned char enabled)

Enable the UART channel.

Parameters

enabled – true: enabled, false: disabled.

Returns

Returns *common entity* return values

aErr **getEnable** (unsigned char %enabled)

Get the enabled state of the uart.

Parameters

enabled – true: enabled, false: disabled.

Returns

Returns *common entity* return values

aErr **setBaudRate** (const unsigned int rate)

Set the UART baud rate. If zero, automatic baud rate selection is used.

Parameters

rate – baud rate.

Returns

Returns *common entity* return values

aErr **getBaudRate** (unsigned int %rate)

Get the UART baud rate. If zero, automatic baud rate selection is used.

Parameters

rate – Pointer variable to be filled with baud rate.

Returns

Returns *common entity* return values

aErr **setProtocol** (const unsigned char protocol)

Set the UART protocol.

Parameters

protocol – An enumeration of serial protocols.

Returns

Returns *common entity* return values

aErr **getProtocol** (unsigned char %protocol)

Get the UART protocol.

Parameters

protocol – Pointer to where result is placed.

Returns

Returns *common entity* return values

aErr **setLinkChannel** (const unsigned char channel)

Set the index of another UART Entity that should be linked to this UART.

If set to the index of this entity, the channel will not be linked. If set to the index of another UART entity, data will be sent between the two UART entities with no additional processing.

Parameters

channel – Index of the UART Entity to link

Returns

Returns *common entity* return values

aErr **getLinkChannel** (unsigned char %channel)

Gets the index of the UART Entity that this entity is linked to.

Parameters

channel – Pointer to where result is placed.

Returns

Returns *common entity* return values

aErr **setStopBits** (const unsigned char stopBits)

Set the UART stop bit configuration

Parameters

stopBits – Stop Bits of UART Channel. Allowed options:

- uartStopBits_1_Value
- uartStopBits_1p5_Value
- uartStopBits_2_Value

Returns

Returns *common entity* return values

aErr **getStopBits** (unsigned char %stopBits)

Set the UART stop bit configuration

Parameters

stopBits – Pointer to where result is placed. Possible values:

- uartStopBits_1_Value
- uartStopBits_1p5_Value
- uartStopBits_2_Value

Returns

Returns *common entity* return values

aErr **setParity** (const unsigned char parity)

Set the UART parity.

Parameters

parity – Parity of UART Channel. Allowed options:

- uartParity_None_Value
- uartParity_Odd_Value
- uartParity_Even_Value
- uartParity_Mark_Value
- uartParity_Space_Value

Returns

Returns *common entity* return values

aErr **getParity** (unsigned char %parity)

Get the UART parity.

Parameters

parity – Pointer variable to be filled with value. Possible values:

- uartParity_None_Value
- uartParity_Odd_Value
- uartParity_Even_Value
- uartParity_Mark_Value
- uartParity_Space_Value

Returns

Returns *common entity* return values

aErr **setDataBits** (const unsigned char dataBits)

Set the number of bits per character

Parameters

dataBits – Data Bits of UART Channel.

Returns

Returns *common entity* return values

aErr **getDataBits** (unsigned char %dataBits)

Get the number of bits per character

Parameters

dataBits – Pointer to where result is placed.

Returns

Returns *common entity* return values

aErr **setFlowControl** (const unsigned char flowControl)

Set the UART flow control configuration

Parameters

flowControl – Flow Control of UART Channel as a bitmask. Allowed bits:

- uartFlowControl_RTS_CTS_Bit
- uartFlowControl_DSR_DTR_Bit
- uartFlowControl_XON_XOFF_Bit

Returns

Returns *common entity* return values

aErr **getFlowControl** (unsigned char %flowControl)

Set the UART flow control configuration

Parameters

flowControl – Pointer to bitmask where result is placed. Possible bits:

- uartFlowControl_RTS_CTS_Bit
- uartFlowControl_DSR_DTR_Bit
- uartFlowControl_XON_XOFF_Bit

Returns

Returns *common entity* return values

aErr **getCapableProtocols** (unsigned int %protocols)

Returns a bitmask containing a list of protocols that this UART entity is allowed to select. This does not guarantee that selecting a protocol with “setProtocol” will have an available resource.

Parameters

protocols – Bitmask containing list of protocols that may be selected. The value of the uartProtocol is mapped to the bit index (e.g. uartProtocol_Undefined is bit 0, uartProtocol_ExtronResponder_Value is bit 1, etc.)

Returns

Returns *common entity* return values

aErr **getAvailableProtocols** (unsigned int %protocols)

Returns a bitmask containing a list of protocols that this UART entity is capable of selecting, and has an available protocol resource to assign.

Parameters

protocols – Bitmask containing list of protocols that are available to select. The value of the uartProtocol is mapped to the bit index (e.g. uartProtocol_Undefined is bit 0, uartProtocol_ExtronResponder_Value is bit 1, etc.)

Returns

Returns *common entity* return values

3.6.26 USB Class

See the *USB Entity* for generic information.

```
class USBClass : public Acroname::BrainStem2CLI::EntityClass
```

USBClass: The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

Public Functions

```
USBClass ()
```

Constructors.

```
~USBClass ()
```

Destructor.

```
!USBClass ()
```

Finalizer.

```
void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)
```

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdUSB index to be addressed.

aErr **setPortEnable** (const unsigned char channel)

Enable both power and data lines for a port.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setPortDisable** (const unsigned char channel)

Disable both power and data lines for a port.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setHiSpeedDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setHiSpeedDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setSuperSpeedDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setSuperSpeedDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setPowerEnable** (const unsigned char channel)

Enable only the power line for a port without changing the state of the data lines.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **setPowerDisable** (const unsigned char channel)

Disable only the power line for a port without changing the state of the data lines.

Parameters

channel – The USB sub channel.

Returns

Returns *common entity* return values

aErr **getPortCurrent** (const unsigned char channel, int %microamps)

Get the current through the power line for a port.

Parameters

- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getPortVoltage** (const unsigned char channel, int %microvolts)

Get the voltage on the power line for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in microvolts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getHubMode** (unsigned int %mode)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

Parameters

mode – The USB hub mode.

Returns

Returns *common entity* return values

aErr **setHubMode** (const unsigned int mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

Parameters

mode – The USB hub mode.

Returns

Returns *common entity* return values

aErr **clearPortErrorStatus** (const unsigned char channel)

Clear the error status for the given port.

Parameters

channel – The port to clear error status for.

Returns

Returns *common entity* return values

aErr **getUpstreamMode** (unsigned char %mode)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

Parameters

mode – The Upstream port mode.

Returns

Returns *common entity* return values

aErr **setUpstreamMode** (const unsigned char mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0, usbUpstreamModePort1, and usbUpstreamModeNone.

Parameters

mode – The Upstream port mode.

Returns

Returns *common entity* return values

aErr **getUpstreamState** (unsigned char %state)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

Parameters

state – The Upstream port state.

Returns

Returns *common entity* return values

aErr **setEnumerationDelay** (const unsigned int ms_delay)

Set the inter-port enumeration delay in milliseconds.

Parameters

ms_delay – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

Returns *common entity* return values

aErr **getEnumerationDelay** (unsigned int %ms_delay)

Get the inter-port enumeration delay in milliseconds.

Parameters

ms_delay – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

Returns *common entity* return values

aErr **setPortCurrentLimit** (const unsigned char channel, const unsigned int microamps)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a `stem.system.save()` call.

Parameters

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

Returns

Returns *common entity* return values

aErr **getPortCurrentLimit** (const unsigned char channel, unsigned int %microamps)

Get the current limit for the port.

Parameters

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

Returns

Returns *common entity* return values

aErr **setPortMode** (const unsigned char channel, const unsigned int mode)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of `usbPortMode`.

Parameters

- **channel** – USB downstream channel to set the mode on.
- **mode** – The port mode setting as packed bit field.

Returns

Returns *common entity* return values

aErr **getPortMode** (const unsigned char channel, unsigned int %mode)

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of `usbPortMode`.

Parameters

- **channel** – USB downstream channel.
- **mode** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Returns

Returns *common entity* return values

aErr **getPortState** (const unsigned char channel, unsigned int %state)

Get the current State for the Port.

Parameters

- **channel** – USB downstream channel.
- **state** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Returns

Returns *common entity* return values

aErr **getPortError** (const unsigned char channel, unsigned int %error)

Get the current error for the Port.

Parameters

- **channel** – USB downstream channel.
- **error** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Returns

Returns *common entity* return values

aErr **setUpstreamBoostMode** (const unsigned char setting)

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4* boost, 2 - 8* boost, 3 - 12* boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0* boost is restored.

Parameters

setting – Upstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **setDownstreamBoostMode** (const unsigned char setting)

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4* boost, 2 - 8* boost, 3 - 12* boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0* boost is restored.

Parameters

setting – Downstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **getUpstreamBoostMode** (unsigned char %setting)

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4* boost, 2 - 8* boost, 3 - 12* boost.

Parameters

setting – The current Upstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **getDownstreamBoostMode** (unsigned char %setting)

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4* boost, 2 - 8* boost, 3 - 12* boost.

Parameters

setting – The current Downstream boost setting 0, 1, 2, or 3.

Returns

Returns *common entity* return values

aErr **getDownstreamDataSpeed** (const unsigned char channel, unsigned char %speed)

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

Parameters

- **channel** – USB downstream channel to check.
- **speed** – Filled with the current port data speed
 - N/A: usbDownstreamDataSpeed_na = 0
 - Hi Speed: usbDownstreamDataSpeed_hs = 1
 - SuperSpeed: usbDownstreamDataSpeed_ss = 2

Returns

Returns *common entity* return values

aErr **setConnectMode** (const unsigned char channel, const unsigned char mode)

Sets the connect mode of the switch.

Parameters

- **channel** – The USB sub channel.
- **mode** – The connect mode
 - usbManualConnect = 0
 - usbAutoConnect = 1

Returns

Returns *common entity* return values

aErr **getConnectMode** (const unsigned char channel, unsigned char %mode)

Gets the connect mode of the switch.

Parameters

- **channel** – The USB sub channel.
- **mode** – The current connect mode

Returns

Returns *common entity* return values

aErr **setCC1Enable** (const unsigned char channel, const unsigned char enable)

Set Enable/Disable on the CC1 line.

Parameters

- **channel** – USB channel.
- **enable** – State to be set

- Disabled: 0
- Enabled: 1

Returns

Returns *common entity* return values

aErr **getCC1Enable** (const unsigned char channel, unsigned char %pEnable)

Get Enable/Disable on the CC1 line.

Parameters

- **channel** – USB channel.
- **pEnable** – State to be filled
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **setCC2Enable** (const unsigned char channel, const unsigned char enable)

Set Enable/Disable on the CC2 line.

Parameters

- **channel** – USB channel.
- **enable** – State to be filled
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCC2Enable** (const unsigned char channel, unsigned char %pEnable)

Get Enable/Disable on the CC2 line.

Parameters

- **channel** – USB channel.
- **pEnable** – State to be filled
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCC1Current** (const unsigned char channel, int %microamps)

Get the current through the CC1 for a port.

Parameters

- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC2Current** (const unsigned char channel, int %microamps)

Get the current through the CC2 for a port.

Parameters

- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **getCC1Voltage** (const unsigned char channel, int %microvolts)

Get the voltage of CC1 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getCC2Voltage** (const unsigned char channel, int %microvolts)

Get the voltage of CC2 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **setSBUEnable** (const unsigned char channel, const unsigned char enable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

Parameters

- **channel** – The USB sub channel.
- **enable** – The state to be set
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getSBUEnable** (const unsigned char channel, unsigned char %pEnable)

Get the Enable/Disable status of the SBU

Parameters

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of the SBU

Returns

Returns *common entity* return values

aErr **setCableFlip** (const unsigned char channel, const unsigned char enable)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters

- **channel** – The USB sub channel.
- **enable** – The state to be set The state to be set
 - Disabled: 0
 - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCableFlip** (const unsigned char channel, unsigned char %pEnable)

Get Cable flip setting.

Parameters

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of cable flip.

Returns

Returns *common entity* return values

aErr **setAltModeConfig** (const unsigned char channel, const unsigned int configuration)

Set USB Alt Mode Configuration.

Parameters

- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

Returns

Returns *common entity* return values

aErr **getAltModeConfig** (const unsigned char channel, unsigned int %configuration)

Get USB Alt Mode Configuration.

Parameters

- **channel** – The USB sub channel
- **configuration** – The USB configuration for the given channel.

Returns

Returns *common entity* return values

aErr **getSBU1Voltage** (const unsigned char channel, int %microvolts)

Get the voltage of SBU1 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getSBU2Voltage** (const unsigned char channel, int %microvolts)

Get the voltage of SBU2 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

3.6.27 USBSystem Class

See the *USBSystem Entity* for generic information.

class **USBSystemClass** : public Acroname::BrainStem2CLI::EntityClass

USBSystemClass: The USBSystem class provides high level control of the lower level Port Class.

Subclassed by Acroname::BrainStem2CLI::MTMIOSerial::HubClass, Acroname::BrainStem2CLI::USBExt3c::HubClass, Acroname::BrainStem2CLI::USBHub2x4::HubClass, Acroname::BrainStem2CLI::USBHub3c::HubClass, Acroname::BrainStem2CLI::USBHub3p::HubClass

Public Functions

USBSystemClass ()

Constructors.

~USBSystemClass ()

Destructor.

!USBSystemClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdUSBSYSTEM index to be addressed.

aErr **getUpstream** (unsigned char %port)

Gets the upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstream** (const unsigned char port)

Sets the upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getEnumerationDelay** (unsigned int %msDelay)

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Parameters

msDelay – the current inter-port delay in milliseconds.

Returns

Returns *common entity* return values

aErr **setEnumerationDelay** (const unsigned int msDelay)

Sets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Parameters

msDelay – The delay in milliseconds to be applied between port enables

Returns

Returns *common entity* return values

aErr **getDataRoleList** (unsigned int %roleList)

Gets the data role of all ports with a single call Equivalent to calling *PortClass::getDataRole()* on each individual port.

Parameters

roleList – A bit packed representation of the data role for all ports.

Returns

Returns *common entity* return values

aErr **getEnabledList** (unsigned int %enabledList)

Gets the current enabled status of all ports with a single call. Equivalent to calling *PortClass::setEnabled()* on each port.

Parameters

enabledList – Bit packed representation of the enabled status for all ports.

Returns

Returns *common entity* return values

aErr **setEnabledList** (const unsigned int enabledList)

Sets the enabled status of all ports with a single call. Equivalent to calling *PortClass::setEnabled()* on each port.

Parameters

enabledList – Bit packed representation of the enabled status for all ports to be applied.

Returns

Returns *common entity* return values

aErr **getModeList** (unsigned int %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Gets the current mode of all ports with a single call. Equivalent to calling *PortClass::getMode()* on each port.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setModeList** (const unsigned int %buffer, const unsigned int bufferLength)

Sets the mode of all ports with a single call. Equivalent to calling *PortClass::setMode()* on each port

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getStateList** (unsigned int %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Gets the state for all ports with a single call. Equivalent to calling *PortClass::getState()* on each port.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **getPowerBehavior** (unsigned char %behavior)

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system, i.e. What happens when requested power greater than available power.

Parameters

behavior – Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setPowerBehavior** (const unsigned char behavior)

Sets the behavior of how available power is managed, i.e. What happens when requested power is greater than available power.

Parameters

behavior – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getPowerBehaviorConfig** (unsigned int %buffer, const unsigned int bufferLength, unsigned int %unloadedLength)

Gets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setPowerBehaviorConfig** (const unsigned int %buffer, const unsigned int bufferSize)

Sets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getDataRoleBehavior** (unsigned char %behavior)

Gets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters

behavior – Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataRoleBehavior** (const unsigned char behavior)

Sets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters

behavior – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getDataRoleBehaviorConfig** (unsigned int %buffer, const unsigned int bufferSize, unsigned int %unloadedLength)

Gets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine priority host priority.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setDataRoleBehaviorConfig** (const unsigned int %buffer, const unsigned int bufferSize)

Sets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine host priority.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.

- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getSelectorMode** (unsigned char %mode)

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters

mode – Variable to be filled with the selector mode

Returns

Returns *common entity* return values

aErr **setSelectorMode** (const unsigned char mode)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters

mode – Mode to be set.

Returns

Returns *common entity* return values

aErr **getUpstreamHS** (unsigned char %port)

Gets the USB HighSpeed upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstreamHS** (const unsigned char port)

Sets the USB HighSpeed upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getUpstreamSS** (unsigned char %port)

Gets the USB SuperSpeed upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstreamSS** (const unsigned char port)

Sets the USB SuperSpeed upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getOverride** (unsigned int %overrides)

Gets the current enabled overrides

Parameters

overrides – Bit mapped representation of the current override configuration.

Returns

Returns *common entity* return values

aErr **setOverride** (const unsigned int overrides)

Sets the current enabled overrides

Parameters

overrides – Overrides to be set in a bit mapped representation.

Returns

Returns *common entity* return values

aErr **setDataHSMMaxDatarate** (const unsigned int datarate)

Sets the USB HighSpeed Max datarate

Parameters

datarate – Maximum datarate for the USB HighSpeed signals.

Returns

Returns *common entity* return values

aErr **getDataHSMMaxDatarate** (unsigned int %datarate)

Gets the USB HighSpeed Max datarate

Parameters

datarate – Current maximum datarate for the USB HighSpeed signals.

Returns

Returns *common entity* return values

aErr **setDataSSMaxDatarate** (const unsigned int datarate)

Sets the USB SuperSpeed Max datarate

Parameters

datarate – Maximum datarate for the USB SuperSpeed signals.

Returns

Returns *common entity* return values

aErr **getDataSSMaxDatarate** (unsigned int %datarate)

Gets the USB SuperSpeed Max datarate

Parameters

datarate – Current maximum datarate for the USB SuperSpeed signals.

Returns

Returns *common entity* return values

3.7 CCA API Reference

3.7.1 Analog Entity

See the [Analog Entity](#) for generic information.

group **AnalogEntity**

Interface to analog entities on BrainStem modules. Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

void **analog_getValue** (unsigned int *id, struct Result *result, const int index)

Get the raw ADC output value in bits.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 16 bit analog reading with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

Note: Not all modules are provide 16 useful bits; this value's least significant bits are zero-padded to 16 bits. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **analog_getVoltage** (unsigned int *id, struct Result *result, const int index)

Get the scaled micro volt value with reference to ground.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Note: Not all modules provide 32 bits of accuracy. Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **analog_getRange** (unsigned int *id, struct Result *result, const int index)

Get the analog input range.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 8 bit value corresponding to a discrete range option

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **analog_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the analog output enable status.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 0 if disabled 1 if enabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **analog_setValue** (unsigned int *id, struct Result *result, const int index, const unsigned short value)

Set the value of an analog output (DAC) in bits.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Note: Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module’s datasheet to determine analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

void **analog_setVoltage** (unsigned int *id, struct Result *result, const int index, const int microvolts)

Set the voltage level of an analog output (DAC) in microvolts.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Note: Voltage range is dependent on the specific DAC channel range.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board’s ground and reference voltages.

void **analog_setRange** (unsigned int *id, struct Result *result, const int index, const unsigned char range)

Set the analog input range.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **range** – 8 bit value corresponding to a discrete range option

void **analog_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Set the analog output enable state.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – set 1 to enable or 0 to disable.

void **analog_setConfiguration** (unsigned int *id, struct Result *result, const int index, const unsigned char configuration)

Set the analog configuration.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **configuration** – bitAnalogConfigurationOutput configures the analog entity as an output.

void **analog_getConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the analog configuration.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Current configuration of the analog entity.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **analog_setBulkCaptureSampleRate** (unsigned int *id, struct Result *result, const int index, const unsigned int value)

Set the sample rate for this analog when bulk capturing.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – sample rate in samples per second (Hertz).
 - Minimum rate: 7,000 Hz
 - Maximum rate: 200,000 Hz

void **analog_getBulkCaptureSampleRate** (unsigned int *id, struct Result *result, const int index)

Get the current sample rate setting for this analog when bulk capturing.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: upon success filled with current sample rate in samples per second (Hertz).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **analog_setBulkCaptureNumberOfSamples** (unsigned int *id, struct Result *result, const int index, const unsigned int value)

Set the number of samples to capture for this analog when bulk capturing.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – number of samples.
 - Minimum # of Samples: 0
 - Maximum # of Samples: $(\text{BRAINSTEM_RAM_SLOT_SIZE} / 2) = (3\text{FFF} / 2) = 1\text{FFF} = 8191$

void **analog_getBulkCaptureNumberOfSamples** (unsigned int *id, struct Result *result, const int index)

Get the current number of samples setting for this analog when bulk capturing.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: number of samples.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **analog_initiateBulkCapture** (unsigned int *id, struct Result *result, const int index)

Initiate a BulkCapture on this analog. Captured measurements are stored in the module’s RAM store (RAM_STORE) slot 0. Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Note: When the bulk capture is complete `getBulkCaptureState()` will return either `bulkCaptureFinished` or `bulkCaptureError`.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

void **analog_getBulkCaptureState** (unsigned int *id, struct Result *result, const int index)

Get the current bulk capture state for this analog.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: the state of bulk capture.
- Idle: bulkCaptureIdle = 0
- Pending: bulkCapturePending = 1
- Finished: bulkCaptureFinished = 2
- Error: bulkCaptureError = 3

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.2 App Entity

See the *App Entity* for generic information.

group **AppEntity**

Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

void **app_execute** (unsigned int *id, struct Result *result, const int index, const unsigned int appParam)

Execute the app reflex on the module. Doesn't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **appParam** – The app parameter handed to the reflex.

void **app_executeAndReturn** (unsigned int *id, struct Result *result, const int index, const unsigned int appParam, const unsigned int msTimeout)

Execute the app reflex on the module. Waits for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

- **value**: The return value filled in from the result of executing the reflex routine.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **appParam** – The app parameter handed to the reflex.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

3.7.3 Clock Entity

See the [Clock Entity](#) for generic information.

group ClockEntity

Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

void **clock_getYear** (unsigned int *id, struct Result *result, const int index)

Get the four digit year value (0-4095).

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: Get the year portion of the real-time clock value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **clock_setYear** (unsigned int *id, struct Result *result, const int index, const unsigned short year)

Set the four digit year value (0-4095).

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **year** – Set the year portion of the real-time clock value.

void **clock_getMonth** (unsigned int *id, struct Result *result, const int index)

Get the two digit month value (1-12).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The two digit month portion of the real-time clock value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **clock_setMonth** (unsigned int *id, struct Result *result, const int index, const unsigned char month)

Set the two digit month value (1-12).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **month** – The two digit month portion of the real-time clock value.

void **clock_getDay** (unsigned int *id, struct Result *result, const int index)

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The two digit day portion of the real-time clock value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **clock_setDay** (unsigned int *id, struct Result *result, const int index, const unsigned char day)

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.

- **index** – The index of the entity in question.
- **day** – The two digit day portion of the real-time clock value.

void **clock_getHour** (unsigned int *id, struct Result *result, const int index)

Get the two digit hour value (0-23).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The two digit hour portion of the real-time clock value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **clock_setHour** (unsigned int *id, struct Result *result, const int index, const unsigned char hour)

Set the two digit hour value (0-23).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **hour** – The two digit hour portion of the real-time clock value.

void **clock_getMinute** (unsigned int *id, struct Result *result, const int index)

Get the two digit minute value (0-59).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The two digit minute portion of the real-time clock value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **clock_setMinute** (unsigned int *id, struct Result *result, const int index, const unsigned char minute)

Set the two digit minute value (0-59).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **minute** – The two digit minute portion of the real-time clock value.

void **clock_getSecond** (unsigned int *id, struct Result *result, const int index)

Get the two digit second value (0-59).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The two digit second portion of the real-time clock value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **clock_setSecond** (unsigned int *id, struct Result *result, const int index, const unsigned char second)

Set the two digit second value (0-59).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **second** – The two digit second portion of the real-time clock value.

3.7.4 Digital Entity

See the *Digital Entity* for generic information.

group **DigitalEntity**

Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

void **digital_setConfiguration** (unsigned int *id, struct Result *result, const int index, const unsigned char configuration)

Set the digital configuration to one of the available 5 states.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Note: Some configurations are only supported on specific pins.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **configuration** – The configuration to be applied
 - Digital Input: digitalConfigurationInput = 0
 - Digital Output: digitalConfigurationOutput = 1
 - RCServo Input: digitalConfigurationRCServoInput = 2
 - RCServo Output: digitalConfigurationRCServoOutput = 3
 - High Z State: digitalConfigurationHiZ = 4
 - Digital Input: digitalConfigurationInputPullUp = 0
 - Digital Input: digitalConfigurationInputNoPull = 4
 - Digital Input: digitalConfigurationInputPullDown = 5

void **digital_getConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the digital configuration.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Current configuration of the digital entity.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **digital_setState** (unsigned int *id, struct Result *result, const int index, const unsigned char state)

Set the logical state.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **state** – The state to be set. 0 is logic low, 1 is logic high.

void **digital_getState** (unsigned int *id, struct Result *result, const int index)

Get the state.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current state of the digital entity. 0 is logic low, 1 is logic high.

Note: If in high Z state an error will be returned.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **digital_setStateAll** (unsigned int *id, struct Result *result, const int index, const unsigned int state)

Sets the logical state of all available digitals based on the bit mapping. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **state** – The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc.

void **digital_getStateAll** (unsigned int *id, struct Result *result, const int index)

Gets the logical state of all available digitals in a bit mapped representation. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **digital_setValue** (unsigned int *id, struct Result *result, const int index, const unsigned char value)

Set the expected value of the digital pin.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – The expected value of the digital pin. 0 is logic low, 1 is logic high.

void **digital_getValue** (unsigned int *id, struct Result *result, const int index)

Get the expected value of the digital pin.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The expected value of the digital pin. 0 is logic low, 1 is logic high.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **digital_setLinkChannel** (unsigned int *id, struct Result *result, const int index, const unsigned char linkChannel)

Set the link channel (entity index) for linking digital entities. Two digital entities will link when one is configured as LinkInput, one as LinkOutput, and both have each others linkChannel indexes.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **linkChannel** – The link channel (entity index) value. Entities with matching linkChannel values can be linked.

void **digital_getLinkChannel** (unsigned int *id, struct Result *result, const int index)

Get the link channel (entity index) for linking digital entities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current link channel (entity index) value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.5 Equalizer Entity

See the [Equalizer Entity](#) for generic information.

group EqualizerEntity

Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname’s products. Please see product documentation for further details.

void **equalizer_setReceiverConfig** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char config)

Sets the receiver configuration for a given channel.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The equalizer receiver channel.
- **config** – Configuration to be applied to the receiver.

void **equalizer_getReceiverConfig** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Gets the receiver configuration for a given channel.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Configuration of the receiver.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The equalizer receiver channel.

void **equalizer_setTransmitterConfig** (unsigned int *id, struct Result *result, const int index, const unsigned char config)

Sets the transmitter configuration

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **config** – Configuration to be applied to the transmitter.

void **equalizer_getTransmitterConfig** (unsigned int *id, struct Result *result, const int index)

Gets the transmitter configuration

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Configuration of the Transmitter.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.6 Ethernet Entity

See the [Ethernet Entity](#) for generic information.

group **EthernetEntity**

IP configuration. MAC info. BrainD port.

void **ethernet_setEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enabled)

Sets the Ethernet’s interface to enabled/disabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enabled** – 1 = enabled; 0 = disabled

void **ethernet_getEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Ethernet interface.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = Fully enabled network connectivity; 0 = Ethernet MAC is disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **ethernet_getNetworkConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the method in which IP Address is assigned to this device

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Method used. Current methods
- NONE = 0
- STATIC = 1
- DHCP = 2

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **ethernet_setNetworkConfiguration** (unsigned int *id, struct Result *result, const int index,
const unsigned char addressStyle)

Get the method in which IP Address is assigned to this device

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **addressStyle** – Method to use. See getNetworkConfiguration for addressStyle enumerations.

void **ethernet_getStaticIPv4Address** (unsigned int *id, struct Result *result, const int index, unsigned
char *buffer, const unsigned int bufferLength)

Get the expected IPv4 address of this device, when networkConfiguration == STATIC

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: occupied bytes in buffer, Should be 4 post-call.

Note: The functional IPv4 address of The Module will differ if NetworkConfiguration != STATIC.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_setStaticIPv4Address** (unsigned int *id, struct Result *result, const int index, const unsigned char *buffer, const unsigned int bufferLength)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_getStaticIPv4Netmask** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get the expected IPv4 netmask of this device, when networkConfiguration == STATIC

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: occupied bytes in buffer, Should be 4 post-call.

Note: The functional IPv4 netmask of The Module will differ if NetworkConfiguration != STATIC.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_setStaticIPv4Netmask** (unsigned int *id, struct Result *result, const int index, const unsigned char *buffer, const unsigned int bufferLength)

Set the desired IPv4 address of this device, if NetworkConfiguration == STATIC

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_getStaticIPv4Gateway** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get the expected IPv4 gateway of this device, when networkConfiguration == STATIC

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: occupied bytes in buffer, Should be 4 post-call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_setStaticIPv4Gateway** (unsigned int *id, struct Result *result, const int index, const unsigned char *buffer, const unsigned int bufferLength)

Set the desired IPv4 gateway of this device, if NetworkConfiguration == STATIC

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] with an IP address
- **bufferLength** – size of buffer. Should be 4. setStaticIPv4Gateway([192, 168, 1, 1], 4) would equate with address “192.168.1.1”

void **ethernet_getIPv4Address** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get the effective IP address of this device.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: occupied bytes in buffer, Should be 4 post-call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_getIPv4Netmask** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get the effective IP netmask of this device.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: occupied bytes in buffer, Should be 4 post-call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_getIPv4Gateway** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get the effective IP gateway of this device.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: occupied bytes in buffer, Should be 4 post-call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[4] for returned output
- **bufferLength** – size of buffer. Should be 4.

void **ethernet_setStaticIPv4DNSAddress** (unsigned int *id, struct Result *result, const int index, const unsigned char *buffer, const unsigned int bufferLength)

Set IPv4 DNS Addresses (plural), if NetworkConfiguration == STATIC

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Total array length in bytes. Must be a multiple of 4.

void **ethernet_getStaticIPv4DNSAddress** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get IPv4 DNS addresses (plural), when NetworkConfiguration == STATIC

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length of occupied bytes of buffer, after the call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Maximum length of array, in bytes.

void **ethernet_getIPv4DNSAddress** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get effective IPv4 DNS addresses, for the current NetworkConfiguration

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length of occupied bytes of buffer, after the call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[N][4]
- **bufferLength** – Maximum length of array, in bytes.

void **ethernet_setHostname** (unsigned int *id, struct Result *result, const int index, const unsigned char *buffer, const unsigned int bufferLength)

Set hostname that's requested when this device sends a DHCP request.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[N]
- **bufferLength** – N, for N bytes.

void **ethernet_getHostname** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get hostname that's requested when this device sends a DHCP request.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length of occupied bytes of buffer, after the call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[N]
- **bufferLength** – N, for N bytes.

void **ethernet_getMACAddress** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Get the MAC address of the Ethernet interface.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length of occupied bytes of buffer, after the call.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – alias to an array of uint8_t[6]
- **bufferLength** – length of buffer that's writeable, should be > 6.

void **ethernet_setInterfacePort** (unsigned int *id, struct Result *result, const int index, const unsigned char service, const unsigned short port)

Set the port of a TCPIP service on the device.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **service** – The index of the service to set the port for.
- **port** – The port to be used for the TCPIP server.

void **ethernet_getInterfacePort** (unsigned int *id, struct Result *result, const int index, const unsigned char service)

Get the port of a TCPIP service on the device.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The port of the TCPIP server.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **service** – The index of the service to get the port for.

3.7.7 HDBaseT Entity

See the [HDBaseT Entity](#) for generic information.

group **HDBaseTEntity**

This entity is only available on certain modules, and provides information on HDBaseT extenders.

void **hdbaset_getSerialNumber** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Gets the serial number of the HDBaseT device (6 bytes)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **hdbaset_getFirmwareVersion** (unsigned int *id, struct Result *result, const int index)

Gets the firmware version of the HDBaseT device

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: A bit packet representation of the firmware version Major: Bits 24-31; Minor: Bits 16-23; Patch: Bits 8-15; Build: Bits 0-7

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getState** (unsigned int *id, struct Result *result, const int index)

Gets the current state of the HDBaseT link

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bit packeted representation of the state.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getCableLength** (unsigned int *id, struct Result *result, const int index)

Gets the perceived cable length

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Cable length in micro-meters

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getMSEA** (unsigned int *id, struct Result *result, const int index)

Gets the Mean Squared Error (MSE) for channel A

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current MSE for channel A in micro-dB

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getMSEB** (unsigned int *id, struct Result *result, const int index)

Gets the Mean Squared Error (MSE) for channel B

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current MSE for channel B in micro-dB

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getRetransmissionRate** (unsigned int *id, struct Result *result, const int index)

Gets the number of successful messages between retransmission

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Instantaneous number of successful messages between retransmission. To be interpreted as: $1 / \text{retransmissionRate}$ for rate interpretation. If the value is 0, there have been no retransmissions, otherwise higher is better..

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getLinkUtilization** (unsigned int *id, struct Result *result, const int index)

Gets the current link utilization

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Utilization in milli-percent

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getEncodingState** (unsigned int *id, struct Result *result, const int index)

Gets the current encoding state.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Signal modulation encoding type.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_getUSB2DeviceTree** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Gets the USB2 tree at the HDBaseT device.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **hdbaset_getUSB3DeviceTree** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Gets the USB3 tree at the HDBaseT device.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled

- **bufferLength** – Length of the buffer to be filed

void **hdbaset_getLinkRole** (unsigned int *id, struct Result *result, const int index)

Gets the current link role In the case of “Auto” the getState API will provide the current role.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Link role

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **hdbaset_setLinkRole** (unsigned int *id, struct Result *result, const int index, const unsigned char role)

Sets the active link role

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **role** – The role to be set.

3.7.8 I2C Entity

See the [I2C Entity](#) for generic information.

group **I2CEntity**

Interface the I2C buses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entities bus.

void **i2c_read** (unsigned int *id, struct Result *result, const int index, const unsigned char address, const unsigned char readLength, unsigned char *buffer)

Read from a device on this I2C bus.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **readLength** – The length of the data to read in bytes.
- **buffer** – The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to `aBRAINSTEM_MAXPACKETBYTES` - 5

`void i2c_write (unsigned int *id, struct Result *result, const int index, const unsigned char address, const unsigned char bufferLength, const unsigned char *buffer)`

Write to a device on this I2C bus.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to write.
- **bufferLength** – The length of the data to write in bytes.
- **buffer** – The data to send to the device This array should be no larger than `aBRAINSTEM_MAXPACKETBYTES` - 5

`void i2c_setPullup (unsigned int *id, struct Result *result, const int index, const unsigned char enable)`

Set bus pull-up state. This call only works with stems that have software controlled pull-ups. Check the datasheet for more information. This parameter is saved when `system.save` is called.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – true enables pull-ups false disables them.

`void i2c_setSpeed (unsigned int *id, struct Result *result, const int index, const unsigned char speed)`

Set I2C bus speed.

This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values: 1 - 100Khz 2 - 400Khz 3 - 1MHz

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.

- **index** – The index of the entity in question.
- **speed** – The speed setting value.

void **i2c_getSpeed** (unsigned int *id, struct Result *result, const int index)

Get I2C bus speed.

This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values: 1 - 100Khz 2 - 400Khz 3 - 1MHz

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The speed setting value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.9 Mux Entity

See the [Mux Entity](#) for generic information.

group **MuxEntity**

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to on or more outputs. One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs; some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

void **mux_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the mux enable/disable status

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: true: mux is enabled, false: the mux is disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **mux_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enable the mux.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – true: enables the mux for the selected channel.

void **mux_getChannel** (unsigned int *id, struct Result *result, const int index)

Get the current selected mux channel.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Indicates which channel is selected.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **mux_setChannel** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Set the current mux channel.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – mux channel to select.

void **mux_getChannelVoltage** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of the indicated mux channel.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 32 bit signed integer (in microvolts) based on the board’s ground and reference voltages.

Note: Not all modules provide 32 bits of accuracy; Refer to the module’s datasheet to determine the analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.
- **channel** – The channel in which voltage was requested.

void **mux_getConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the configuration of the mux.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: integer representing the mux configuration either default, or split-mode.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **mux_setConfiguration** (unsigned int *id, struct Result *result, const int index, const int config)

Set the configuration of the mux.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **config** – integer representing the mux configuration either muxConfig_default, or muxConfig_splitMode.

void **mux_getSplitMode** (unsigned int *id, struct Result *result, const int index)

Get the current split mode mux configuration.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **mux_setSplitMode** (unsigned int *id, struct Result *result, const int index, const int splitMode)

Sets the mux’s split mode configuration.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **splitMode** – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

3.7.10 PoE Entity

See the [PoE Entity](#) for generic information.

group **PoEEntity**

This entity is only available on certain modules, and provides a Power over Ethernet control ability.

void **poe_getPairEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the current enable value of the indicated POE pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 1 = Enabled; 0 = Disabled;

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_setPairEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char pair, const unsigned char enable)

Enables or disables the indicated POE pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2

– 1 = Pair 3/4

- **enable** – 1 = Enable port; 0 = Disable port.

void **poe_getPowerMode** (unsigned int *id, struct Result *result, const int index)

Gets the power mode of the device

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The power mode (PD, PSE, Auto, Off).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **poe_setPowerMode** (unsigned int *id, struct Result *result, const int index, const unsigned char value)

Sets the power mode of the device

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – The power mode (PD, PSE, Auto, Off).

void **poe_getPowerState** (unsigned int *id, struct Result *result, const int index)

Gets the power state of the device

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The power state (PD, PSE, Off).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **poe_getPairSourcingClass** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the sourcing class for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The POE class being offered by the device (PSE).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_setPairSourcingClass** (unsigned int *id, struct Result *result, const int index, const unsigned char pair, const unsigned char value)

Sets the sourcing class for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **value** – The POE class being offered by the device (PSE).

void **poe_getPairRequestedClass** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the requested class for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The requested POE class by the device (PD).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getPairDiscoveredClass** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the discovered class for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The negotiated POE class by the device (PSE/PD).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getPairDetectionStatus** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets detected status of the POE connection for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current detected status of the pairs.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getPairVoltage** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the Voltage for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The voltage in microvolts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getPairCurrent** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the Current for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current in microamps (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getPairResistance** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the Resistance for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The resistance in milliohms (1 == 1e-3Z).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getPairCapacitance** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the Capacitance for a given pair

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The capacitance in nanofarads (1 == 1e-9F).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getPairPower** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Get the instantaneous power consumption for a given pair The equivalent of Voltage x Current

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the pairs power in milli-watts (mW).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_getTotalPower** (unsigned int *id, struct Result *result, const int index)

Gets the total instantaneous power consumption The equivalent of Pair1(Voltage x Current) + Pair2(Voltage x Current)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the total POE power in milli-watts (mW).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **poe_getPairAccumulatedPower** (unsigned int *id, struct Result *result, const int index, const unsigned char pair)

Gets the accumulated power for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the total accumulated POE power in milli-watts (mW).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4

void **poe_setPairAccumulatedPower** (unsigned int *id, struct Result *result, const int index, const unsigned char pair, const int power)

Sets the accumulated power for a given pair.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **pair** – Selects PoE pair to access
 - 0 = Pair 1/2
 - 1 = Pair 3/4
- **power** – The power accumulator value to be set in milli-watts (mW).

void **poe_getTotalAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the total Accumulated Power

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the total accumulated POE power in milli-watts (mW).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **poe_setTotalAccumulatedPower** (unsigned int *id, struct Result *result, const int index, const int power)

Sets the total accumulated power

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **power** – The power accumulator value to be set in milli-watts (mW).

3.7.11 Pointer Entity

See the [Pointer Entity](#) for generic information.

group **PointerEntity**

Allows access to the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the BrainStem scratchpad from the host, and provides a mechanism for allowing the host application and BrainStem relexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via `setOffset`. A read of a character short or int can be made from that cursor position.

In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

void **pointer_getOffset** (unsigned int *id, struct Result *result, const int index)

Get the offset of the pointer

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The value of the offset.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **pointer_setOffset** (unsigned int *id, struct Result *result, const int index, const unsigned short offset)

Set the offset of the pointer

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **offset** – The value of the offset.

void **pointer_getMode** (unsigned int *id, struct Result *result, const int index)

Get the mode of the pointer

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **pointer_setMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Set the mode of the pointer

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

void **pointer_getTransferStore** (unsigned int *id, struct Result *result, const int index)

Get the handle to the store.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The handle of the store.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **pointer_setTransferStore** (unsigned int *id, struct Result *result, const int index, const unsigned char handle)

Set the handle to the store.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **handle** – The handle of the store.

void **pointer_initiateTransferToStore** (unsigned int *id, struct Result *result, const int index, const unsigned char transferLength)

Transfer data to the store.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **transferLength** – The length of the data transfer.

void **pointer_initiateTransferFromStore** (unsigned int *id, struct Result *result, const int index, const unsigned char transferLength)

Transfer data from the store.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **transferLength** – The length of the data transfer.

void **pointer_getChar** (unsigned int *id, struct Result *result, const int index)

Get a char (1 byte) value from the pointer at this object’s index, where elements are 1 byte long.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The value of a single character (1 byte) stored in the pointer.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **pointer_setChar** (unsigned int *id, struct Result *result, const int index, const unsigned char value)

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – The single char (1 byte) value to be stored in the pointer.

void **pointer_getShort** (unsigned int *id, struct Result *result, const int index)

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The value of a single short (2 byte) stored in the pointer.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **pointer_setShort** (unsigned int *id, struct Result *result, const int index, const unsigned short value)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – The single short (2 byte) value to be set in the pointer.

void **pointer_getInt** (unsigned int *id, struct Result *result, const int index)

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The value of a single int (4 byte) stored in the pointer.

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **pointer_setInt** (unsigned int *id, struct Result *result, const int index, const unsigned int value)

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – The single int (4 byte) value to be stored in the pointer.

3.7.12 Port Entity

See the [Port Entity](#) for generic information.

group **PortEntity**

The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

void **port_getVbusVoltage** (unsigned int *id, struct Result *result, const int index)

Gets the Vbus Voltage

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The voltage in microvolts (1 == 1e-6V) currently present on Vbus.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getVbusCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the Vbus Current

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current in microamps (1 == 1e-6A) currently present on Vbus.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.

void **port_getVconnVoltage** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Voltage

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The voltage in microvolts (1 == 1e-6V) currently present on Vconn.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getVconnCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Current

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current in microamps (1 == 1e-6A) currently present on Vconn.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getPowerMode** (unsigned int *id, struct Result *result, const int index)

Gets the Port Power Mode: Convenience Function of get/setPortMode

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current power mode.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setPowerMode** (unsigned int *id, struct Result *result, const int index, const unsigned char powerMode)

Sets the Port Power Mode: Convenience Function of get/setPortMode

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **powerMode** – The power mode to be set.

void **port_getEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = Fully enabled port; 0 = One or more disabled components.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the entire port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Fully enable port; 0 = Fully disable port.

void **port_getDataEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the data lines. Sub-component (Data) of getEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = Data enabled; 0 = Data disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the data lines. Sub-component (Data) of setEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

void **port_getDataHSEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = Data enabled; 0 = Data disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataHSEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

void **port_getDataHS1Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the High Speed A side (HSA) data lines. Sub-component of getDataHSEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = Data enabled; 0 = Data disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataHS1Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the High Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

void **port_getDataHS2Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the High Speed B side (HSB) data lines. Sub-component of getDataHSEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 1 = Data enabled; 0 = Data disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataHS2Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the High Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

void **port_getDataSSEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 1 = Data enabled; 0 = Data disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataSSEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

void **port_getDataSS1Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Super Speed A side (SSA) data lines. Sub-component of getDataSSEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = Data enabled; 0 = Data disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataSS1Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Super Speed A side (SSA) data lines. Sub-component of setDataEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

void **port_getDataSS2Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Super Speed B side (SSB) data lines. Sub-component of get-DataSSEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 1 = Data enabled; 0 = Data disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataSS2Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Super Speed B side (SSB) data lines. Sub-component of setDataSSEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

void **port_getPowerEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the power lines. Sub-component (Power) of getEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 1 = Power enabled; 0 = Power disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setPowerEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or Disables the power lines. Sub-component (Power) of setEnable.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable power; 0 = Disable disable.

void **port_getDataRole** (unsigned int *id, struct Result *result, const int index)

Gets the Port Data Role.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The data role to be set. See datasheet for details.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getVconnEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Vconn lines. Sub-component (Vconn) of getEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = Vconn enabled; 0 = Vconn disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setVconnEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn lines; 0 = Disable Vconn lines.

void **port_getVconn1Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 1 = Vconn1 enabled; 0 = Vconn1 disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setVconn1Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn1 lines; 0 = Disable Vconn1 lines.

void **port_getVconn2Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 1 = Vconn2 enabled; 0 = Vconn2 disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setVconn2Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn2 lines; 0 = Disable Vconn2 lines.

void **port_getCCEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the CC lines. Sub-component (CC) of getEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = CC enabled; 0 = CC disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCCEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC lines; 0 = Disable CC lines.

void **port_getCC1Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the CC1 lines. Sub-component of getCCEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = CC1 enabled; 0 = CC1 disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCC1Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the CC1 lines. Sub-component of setCCEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC1 lines; 0 = Disable CC1 lines.

void **port_getCC2Enabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the CC2 lines. Sub-component of getCCEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: 1 = CC2 enabled; 0 = CC2 disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCC2Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the CC2 lines. Sub-component of setCCEnabled.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC2 lines; 0 = Disable CC2 lines.

void **port_getVoltageSetpoint** (unsigned int *id, struct Result *result, const int index)

Gets the current voltage setpoint value for the port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: the voltage setpoint of the port in uV.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setVoltageSetpoint** (unsigned int *id, struct Result *result, const int index, const unsigned int value)

Sets the current voltage setpoint value for the port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – the voltage setpoint of the port in uV.

void **port_getState** (unsigned int *id, struct Result *result, const int index)

A bit mapped representation of the current state of the port. Reflects what the port IS which may differ from what was requested.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the current state.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getDataSpeed** (unsigned int *id, struct Result *result, const int index)

Gets the speed of the enumerated device.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bit mapped value representing the devices speed. See “Devices” reference for details.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getMode** (unsigned int *id, struct Result *result, const int index)

Gets current mode of the port

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bit mapped value representing the ports mode. See “Devices” reference for details.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setMode** (unsigned int *id, struct Result *result, const int index, const unsigned int mode)

Sets the mode of the port

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – Port mode to be set. See “Devices” documentation for details.

void **port_getErrors** (unsigned int *id, struct Result *result, const int index)

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bit mapped field representing the current errors of the ports

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCurrentLimit** (unsigned int *id, struct Result *result, const int index)

Gets the current limit of the port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the limit in microAmps (uA).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCurrentLimit** (unsigned int *id, struct Result *result, const int index, const unsigned int limit)

Sets the current limit of the port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **limit** – Current limit to be applied in microAmps (uA).

void **port_getCurrentLimitMode** (unsigned int *id, struct Result *result, const int index)

Gets the current limit mode. The mode determines how the port will react to an over current condition.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCurrentLimitMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

void **port_getAvailablePower** (unsigned int *id, struct Result *result, const int index)

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with the available power in milli-watts (mW).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.

void **port_getAllocatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the currently allocated power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the allocated power in milli-watts (mW).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getPowerLimit** (unsigned int *id, struct Result *result, const int index)

Gets the user defined power limit for the port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the power limit in milli-watts (mW).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setPowerLimit** (unsigned int *id, struct Result *result, const int index, const unsigned int limit)

Sets a user defined power limit for the port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **limit** – Power limit to be applied in milli-watts (mW).

void **port_getPowerLimitMode** (unsigned int *id, struct Result *result, const int index)

Gets the power limit mode. The mode determines how the port will react to an over power condition.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setPowerLimitMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

void **port_getName** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **port_setName** (unsigned int *id, struct Result *result, const int index, const unsigned char *buffer, const unsigned int bufferLength)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

void **port_getCCCurrentLimit** (unsigned int *id, struct Result *result, const int index)

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with an enumerated representation of the CC Current limit.
- 0 = None
- 1 = Default (500/900mA)
- 2 = 1.5A
- 3 = 3.0A

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCCCurrentLimit** (unsigned int *id, struct Result *result, const int index, const unsigned char value)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotiations.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **value** – Variable to be filled with an enumerated representation of the CC Current limit.
 - 0 = None
 - 1 = Default (500/900mA)
 - 2 = 1.5A
 - 3 = 3.0A

void **port_getDataHSRoutingBehavior** (unsigned int *id, struct Result *result, const int index)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

- **value**: Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataHSRoutingBehavior** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

void **port_getDataSSRoutingBehavior** (unsigned int *id, struct Result *result, const int index)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- **value**: Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setDataSSRoutingBehavior** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.

- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

void **port_getVbusAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the Vbus Accumulated Power

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The accumulated power on Vbus in milliwatt-hours.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setVbusAccumulatedPower** (unsigned int *id, struct Result *result, const int index, const int milliwatthours)

Sets the Vbus Accumulated Power

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **milliwatthours** – The accumulated power on Vbus in milliwatt-hours to be set.

void **port_resetVbusAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Resets the Vbus Accumulated Power to zero.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Deprecated:

Replace with setVbusAccumulatedPower(0)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

void **port_getVconnAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Accumulated Power

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The accumulated power on Vconn in milliwatt-hours.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setVconnAccumulatedPower** (unsigned int *id, struct Result *result, const int index, const int milliwatthours)

Sets the Vconn Accumulated Power

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **milliwatthours** – The accumulated power on Vconn to be set.

void **port_resetVconnAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Resets the Vconn Accumulated Power to zero.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Deprecated:

Replace with setVconnAccumulatedPower(0)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

void **port_setHSBoost** (unsigned int *id, struct Result *result, const int index, const unsigned char boost)

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

- **boost** – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

void **port_getHSBoost** (unsigned int *id, struct Result *result, const int index)

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCC1State** (unsigned int *id, struct Result *result, const int index)

Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:
- None = 0 = portCC1State_None
- Invalid = 1 = portCC1State_Invalid
- Rp (default) = 2 = portCC1State_RpDefault
- Rp (1.5A) = 3 = portCC1State_Rp1p5
- Rp (3A) = 4 = portCC1State_Rp3p0
- Rd = 5 = portCC1State_Rd
- Ra = 6 = portCC1State_Ra
- Managed by controller = 7 = portCC1State_Managed
- Unknown = 8 = portCC1State_Unknown

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCC2State** (unsigned int *id, struct Result *result, const int index)

Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:
 - None = 0 = portCC2State_None
 - Invalid = 1 = portCC2State_Invalid
 - Rp (default) = 2 = portCC2State_RpDefault
 - Rp (1.5A) = 3 = portCC2State_Rp1p5
 - Rp (3A) = 4 = portCC2State_Rp3p0
 - Rd = 5 = portCC2State_Rd
 - Ra = 6 = portCC2State_Ra
 - Managed by controller = 7 = portCC2State_Managed
 - Unknown = 8 = portCC2State_Unknown

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getSBU1Voltage** (unsigned int *id, struct Result *result, const int index)

Get the voltage of SBU1 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getSBU2Voltage** (unsigned int *id, struct Result *result, const int index)

Get the voltage of SBU2 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCC1Voltage** (unsigned int *id, struct Result *result, const int index)

Get the voltage of CC1 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCC2Voltage** (unsigned int *id, struct Result *result, const int index)

Get the voltage of CC2 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCC1Current** (unsigned int *id, struct Result *result, const int index)

Get the current through the CC1 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel current in micro-amps (1 == 1e-6A).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCC2Current** (unsigned int *id, struct Result *result, const int index)

Get the current through the CC2 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

- **value**: The USB channel current in micro-amps (1 == 1e-6A).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_getCC1AccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the CC1 Accumulated Power

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: The accumulated power on Vconn in milliwatt-hours.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCC1AccumulatedPower** (unsigned int *id, struct Result *result, const int index, const int milliwatthours)

Sets the CC1 Accumulated Power

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **milliwatthours** – The accumulated power on Vconn to be set.

void **port_getCC2AccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the CC2 Accumulated Power

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: The accumulated power on Vconn in milliwatt-hours.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **port_setCC2AccumulatedPower** (unsigned int *id, struct Result *result, const int index, const int milliwatthours)

Sets the CC2 Accumulated Power

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **milliwatthours** – The accumulated power on Vconn to be set.

3.7.13 PowerDelivery Entity

See the [PowerDelivery Entity](#) for generic information.

group **PowerDeliveryEntity**

Power Delivery or PD is a power specification which allows more charging options and device behaviors within the USB interface. This Entity will allow you to directly access the vast landscape of PD.

void **powerdelivery_getConnectionState** (unsigned int *id, struct Result *result, const int index)

Gets the current state of the connection in the form of an enumeration.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Pointer to be filled with the current connection state.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getNumberOfPowerDataObjects** (unsigned int *id, struct Result *result, const int index, const unsigned char partner, const unsigned char powerRole)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the number of PDOs.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

void **powerdelivery_getPowerDataObject** (unsigned int *id, struct Result *result, const int index, const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with the requested power rule.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

void **powerdelivery_setPowerDataObject** (unsigned int *id, struct Result *result, const int index, const unsigned char powerRole, const unsigned char ruleIndex, const unsigned int pdo)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource

- Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Power Data Object to be set.

void **powerdelivery_resetPowerDataObjectToDefault** (unsigned int *id, struct Result *result, const int index, const unsigned char powerRole, const unsigned char ruleIndex)

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

void **powerdelivery_getPowerDataObjectList** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets all Power Data Objects (PDOs). Equivalent to calling PowerDeliveryClass::getPowerDataObject() on all partners, power roles, and index's.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length that was actually received and filled. On success this value should be 28 (7 rules * 2 partners * 2 power roles)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled The order of which is:
 - Rules 1-7 Local Source
 - Rules 1-7 Local Sink
 - Rules 1-7 Partner Source
 - Rules 1-7 Partner Sink.
- **bufferLength** – Length of the buffer to be filled

```
void powerdelivery_getPowerDataObjectEnabled (unsigned int *id, struct Result *result, const int
                                             index, const unsigned char powerRole, const
                                             unsigned char ruleIndex)
```

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with enabled state.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

```
void powerdelivery_setPowerDataObjectEnabled (unsigned int *id, struct Result *result, const int
                                             index, const unsigned char powerRole, const
                                             unsigned char ruleIndex, const unsigned char
                                             enabled)
```

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – The state to be set.

```
void powerdelivery_getPowerDataObjectEnabledList (unsigned int *id, struct Result *result, const
                                                  int index, const unsigned char powerRole)
```

Gets all Power Data Object enables for a given power role. Equivalent of calling `PowerDeliveryClass::getPowerDataObjectEnabled()` for all indexes.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with a mapped representation of the enabled PDOs for a given power role. Values align with a given rule index (bits 1-7, bit 0 is invalid)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
 - Source = 1 = `powerdeliveryPowerRoleSource`
 - Sink = 2 = `powerdeliveryPowerRoleSink`

void **powerdelivery_getRequestDataObject** (unsigned int *id, struct Result *result, const int index, const unsigned char partner)

Gets the current Request Data Object (RDO) for a given partner. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with the current RDO. Zero indicates the RDO is not active.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = `powerdeliveryPartnerLocal`
 - Remote = 1 = `powerdeliveryPartnerRemote`

void **powerdelivery_setRequestDataObject** (unsigned int *id, struct Result *result, const int index, const unsigned int rdo)

Sets the current Request Data Object (RDO) for a given partner. Only the local partner can be changed. RDOs are provided by the sinking device and exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **rdo** – Request Data Object to be set.

void **powerdelivery_getLinkState** (unsigned int *id, struct Result *result, const int index)

Gets the current state of the connection in the form of a bitmask.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Pointer to be filled with the current connection state bits.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getAttachTimeElapsed** (unsigned int *id, struct Result *result, const int index,
unsigned int *secondsAndMicroseconds, const
unsigned int bufferLength)

Gets the length of time that the port has been in the attached state. Returned as a list of two unsigned integers, first seconds, then microseconds.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **secondsAndMicroseconds** – Pointer to list of unsigned integers to fill with attach time elapsed
- **bufferLength** – Length of the buffer to be filled

void **powerdelivery_getPowerRoleCapabilities** (unsigned int *id, struct Result *result, const int
index)

Gets the power roles that may be advertised by the local partner. (CC Strapping).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with the power role
- None = 0 = pdPowerRoleCapabilities_None
- Source = 1 = pdPowerRoleCapabilities_Source
- Sink = 2 = pdPowerRoleCapabilities_Sink
- Source/Sink = 3 = pdPowerRoleCapabilities_DualRole (Dual Role Port)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getPowerRole** (unsigned int *id, struct Result *result, const int index)

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filed with the power role
- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_setPowerRole** (unsigned int *id, struct Result *result, const int index, const unsigned char powerRole)

Set the current power role to be advertised by the Local partner. (CC Strapping).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **powerRole** – Value to be applied.
 - Disabled = 0 = powerdeliveryPowerRoleDisabled
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
 - Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

void **powerdelivery_getPowerRolePreferred** (unsigned int *id, struct Result *result, const int index)

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

- value: Value to be applied.
- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_setPowerRolePreferred** (unsigned int *id, struct Result *result, const int index, const unsigned char powerRole)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **powerRole** – Value to be applied.
 - Disabled = 0 = powerdeliveryPowerRoleDisabled
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

void **powerdelivery_getDataRoleCapabilities** (unsigned int *id, struct Result *result, const int index)

Gets the data roles that may be advertised by the local partner.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the data role
- None = 0 = pDataRoleCapabilities_None
- DFP = 1 = pDataRoleCapabilities_DFP
- UFP = 2 = pDataRoleCapabilities_UFP
- DFP/UFP = 3 = pDataRoleCapabilities_DualRole (Dual Role Port)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getCableVoltageMax** (unsigned int *id, struct Result *result, const int index)

Gets the maximum voltage capability reported by the e-mark of the attached cable.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with an enumerated representation of voltage.
- Unknown/Unattached (0)
- 20 Volts DC (1)
- 30 Volts DC (2)
- 40 Volts DC (3)
- 50 Volts DC (4)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getCableCurrentMax** (unsigned int *id, struct Result *result, const int index)

Gets the maximum current capability report by the e-mark of the attached cable.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with an enumerated representation of current.
- Unknown/Unattached (0)
- 3 Amps (1)
- 5 Amps (2)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getCableSpeedMax** (unsigned int *id, struct Result *result, const int index)

Gets the maximum data rate capability reported by the e-mark of the attached cable.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with an enumerated representation of data speed.
- Unknown/Unattached (0)
- USB 2.0 (1)
- USB 3.2 gen 1 (2)
- USB 3.2 / USB 4 gen 2 (3)

- USB 4 gen 3 (4)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getCableType** (unsigned int *id, struct Result *result, const int index)

Gets the cable type reported by the e-mark of the attached cable.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with an enumerated representation of the cable type.
- Invalid, no e-mark and not Vconn powered (0)
- Passive cable with e-mark (1)
- Active cable (2)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getCableOrientation** (unsigned int *id, struct Result *result, const int index)

Gets the current orientation being used for PD communication

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable filled with an enumeration of the orientation.
- Unconnected (0)
- CC1 (1)
- CC2 (2)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_request** (unsigned int *id, struct Result *result, const int index, const unsigned char request)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **request** – Request to be issued to the remote partner
 - pdRequestHardReset (0)
 - pdRequestSoftReset (1)
 - pdRequestDataReset (2)
 - pdRequestPowerRoleSwap (3)
 - pdRequestPowerFastRoleSwap (4)
 - pdRequestDataRoleSwap (5)
 - pdRequestVconnSwap (6)
 - pdRequestSinkGoToMinimum (7)
 - pdRequestRemoteSourcePowerDataObjects (8)
 - pdRequestRemoteSinkPowerDataObjects (9)
 - pdRequestRemoteSourceExtendedCapabilities (10)
 - pdRequestRemoteSinkExtendedCapabilities (11)
 - pdRequestStatus (12)
 - pdRequestPPSStatus (13)
 - pdRequestBatteryCapabilities (14)
 - pdRequestBatteryStatus (15)
 - pdRequestManufacturerInfoSop (16)
 - pdRequestManufacturerInfoSopp (17)
 - pdRequestManufacturerInfoSoppp (18)
 - pdRequestDiscoverIdentitySop (19)
 - pdRequestDiscoverIdentitySopp (20)
 - pdRequestDiscoverIdentitySoppp (21)
 - pdRequestRevision (22)
 - pdRequestSourceInfo (23)
 - pdRequestCountryCode (24)
 - pdRequestCountryInfo (25)
 - pdRequestRemoteSourceEPRCapabilities (26)
 - pdRequestRemoteSinkEPRCapabilities (27)

void **powerdelivery_requestStatus** (unsigned int *id, struct Result *result, const int index)

Gets the status of the last request command sent.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

- **value**: Variable to be filled with the status

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_getOverride** (unsigned int *id, struct Result *result, const int index)

Gets the current enabled overrides

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: Bit mapped representation of the current override configuration.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_setOverride** (unsigned int *id, struct Result *result, const int index, const unsigned int overrides)

Sets the current enabled overrides

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **overrides** – Overrides to be set in a bit mapped representation.

void **powerdelivery_getProtocolRevisionMax** (unsigned int *id, struct Result *result, const int index)

Gets the configured maximum USB Power Delivery protocol revision used as a negotiation ceiling (pdProtocolRevision2 = 2, pdProtocolRevision3 = 3). This is unrelated to hardware or product revision.

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: Filled with pdProtocolRevision2 (2) or pdProtocolRevision3 (3).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_setProtocolRevisionMax** (unsigned int *id, struct Result *result, const int index, const unsigned char maxProtocolRevision)

Sets the maximum USB Power Delivery protocol revision for negotiation. Only pdProtocolRevision2 (2) and pdProtocolRevision3 (3) are valid API values.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **maxProtocolRevision** – pdProtocolRevision2 (2) or pdProtocolRevision3 (3).

void **powerdelivery_getFlagMode** (unsigned int *id, struct Result *result, const int index, const unsigned char flag)

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the current mode.
- Disabled (0)
- Enabled (1)
- Auto (2) default

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **flag** – Flag/Advertisement to be modified

void **powerdelivery_setFlagMode** (unsigned int *id, struct Result *result, const int index, const unsigned char flag, const unsigned char mode)

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.

- **index** – The index of the entity in question.
- **flag** – Flag/Advertisement to be modified
- **mode** – Value to be applied.
 - Disabled (0)
 - Enabled (1)
 - Auto (2) default

void **powerdelivery_getPeakCurrentConfiguration** (unsigned int *id, struct Result *result, const int index)

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: An enumerated value referring to the current configuration.
- Allowable values are 0 - 4

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_setPeakCurrentConfiguration** (unsigned int *id, struct Result *result, const int index, const unsigned char configuration)

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **configuration** – An enumerated value referring to the configuration to be set
 - Allowable values are 0 - 4

void **powerdelivery_getFastRoleSwapCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: An enumerated value referring to current swap value.
- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **powerdelivery_setFastRoleSwapCurrent** (unsigned int *id, struct Result *result, const int index, const unsigned char swapCurrent)

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **swapCurrent** – An enumerated value referring to value to be set.
 - 0A (0)
 - 900mA (1)
 - 1.5A (2)
 - 3A (3)

void **powerdelivery_packDataObjectAttributes** (struct Result *result, const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex)

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with packed values.

Parameters

- **result** – Output object containing result code and the requested value if successful.
- **partner** – Indicates which side of the PD connection.

- Local = 0 = powerdeliveryPartnerLocal
- Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

void **powerdelivery_unpackDataObjectAttributes** (struct Result *result, const unsigned char attributes, unsigned char *partner, unsigned char *powerRole, unsigned char *ruleIndex)

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **result** – Output object containing result code.
- **attributes** – Variable to be filled with packed values.
- **partner** – Indicates which side of the PD connection.
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

3.7.14 RCServo Entity

See the *RCServo Entity* for generic information.

group RCServoEntity

Interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

void **rcservo_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enable the servo channel

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – The state to be set. 0 is disabled, 1 is enabled.

void **rcservo_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the enable status of the servo channel.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The current enable status of the servo entity. 0 is disabled, 1 is enabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rcservo_setPosition** (unsigned int *id, struct Result *result, const int index, const unsigned char position)

Set the position of the servo channel

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **position** – The position to be set. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

void **rcservo_getPosition** (unsigned int *id, struct Result *result, const int index)

Get the position of the servo channel

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The current position of the servo channel. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rcservo_setReverse** (unsigned int *id, struct Result *result, const int index, const unsigned char reverse)

Set the output to be reversed on the servo channel

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **reverse** – Reverses the value set by “setPosition”. For example, if the position is set to 64 (1ms pulse) the output will now be 192 (2ms pulse), however “getPostion” will return the set value of 64. 0 = not reversed, 1 = reversed.

void **rcservo_getReverse** (unsigned int *id, struct Result *result, const int index)

Get the reverse status of the servo channel

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current reverse status of the servo entity. 0 = not reversed, 1 = reversed.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.15 Rail Entity

See the [Rail Entity](#) for generic information.

group **RailEntity**

Provides power rail functionality on certain modules. The RailClass can be used to control power to downstream devices. It has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

void **rail_getCurrent** (unsigned int *id, struct Result *result, const int index)

Get the rail current.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current in micro-amps (1 == 1e-6A).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.

void **rail_setCurrentSetpoint** (unsigned int *id, struct Result *result, const int index, const int microamps)

Set the rail supply current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microamps** – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

void **rail_getCurrentSetpoint** (unsigned int *id, struct Result *result, const int index)

Get the rail setpoint current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current in micro-amps (1 == 1e-6A) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setCurrent interface. Refer to the module datasheet to determine if this is a measured or stored value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setCurrentLimit** (unsigned int *id, struct Result *result, const int index, const int microamps)

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microamps** – The current in micro-amps (1 == 1e-6A).

void **rail_getCurrentLimit** (unsigned int *id, struct Result *result, const int index)

Get the rail current limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current in micro-amps (1 == 1e-6A).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_getTemperature** (unsigned int *id, struct Result *result, const int index)

Get the rail temperature.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The measured temperature associated with the rail in micro-Celsius (1 == 1e-6°C). The temperature may be associated with the module’s internal rail circuitry or an externally connected temperature sensors. Refer to the module datasheet for definition of the temperature measurement location and specific capabilities.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Set the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

void **rail_getVoltage** (unsigned int *id, struct Result *result, const int index)

Get the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The voltage in micro-volts (1 == 1e-6V) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setVoltageSetpoint** (unsigned int *id, struct Result *result, const int index, const int microvolts)

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

void **rail_getVoltageSetpoint** (unsigned int *id, struct Result *result, const int index)

Get the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The voltage in micro-volts (1 == 1e-6V) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setVoltageMinLimit** (unsigned int *id, struct Result *result, const int index, const int microvolts)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V).

void **rail_getVoltageMinLimit** (unsigned int *id, struct Result *result, const int index)

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setVoltageMaxLimit** (unsigned int *id, struct Result *result, const int index, const int microvolts)

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V).

void **rail_getVoltageMaxLimit** (unsigned int *id, struct Result *result, const int index)

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_getPower** (unsigned int *id, struct Result *result, const int index)

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The power in micro-watts (1 == 1e-6W) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to determine if this is a measured or stored value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setPowerSetpoint** (unsigned int *id, struct Result *result, const int index, const int microwatts)

Set the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microwatts** – The power in micro-watts (1 == 1e-6W) to be supplied by the rail.

void **rail_getPowerSetpoint** (unsigned int *id, struct Result *result, const int index)

Get the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

- **value:** The power in micro-watts ($1 == 1e-6W$) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the `setPower` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Parameters

- **id** – ID assigned through “`module_createStem`”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setPowerLimit** (unsigned int *id, struct Result *result, const int index, const int microwatts)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “`module_createStem`”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microwatts** – The power in micro-watts ($1 == 1e-6W$).

void **rail_getPowerLimit** (unsigned int *id, struct Result *result, const int index)

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- **value:** The power in micro-watts ($1 == 1e-6W$).

Parameters

- **id** – ID assigned through “`module_createStem`”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_getResistance** (unsigned int *id, struct Result *result, const int index)

Get the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- **value:** The resistance in micro-ohms ($1 == 1e-6Ohms$) currently drawn by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the `setResistance` interface. Refer to the module datasheet to determine if this is a measured or stored value.

Parameters

- **id** – ID assigned through “`module_createStem`”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.

void **rail_setResistanceSetpoint** (unsigned int *id, struct Result *result, const int index, const int microohms)

Set the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **microohms** – The resistance in micro-ohms (1 == 1e-6Ohms) to be drawn by the rail.

void **rail_getResistanceSetpoint** (unsigned int *id, struct Result *result, const int index)

Get the rail setpoint resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The resistance in micro-ohms (1 == 1e-6Ohms) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setResistance interface. Refer to the module datasheet to determine if this is a measured or stored value.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setKelvinSensingEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – enable or disable kelvin sensing.

void **rail_getKelvinSensingEnable** (unsigned int *id, struct Result *result, const int index)

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Kelvin sensing is enabled or disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_getKelvinSensingState** (unsigned int *id, struct Result *result, const int index)

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Kelvin sensing is enabled or disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_setOperationalMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – The operational mode to employ.

void **rail_getOperationalMode** (unsigned int *id, struct Result *result, const int index)

Determine the current operational mode of the system. Refer to the module datasheet for definition of the rail operational mode capabilities.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

- **value**: The current operational mode setting.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_getOperationalState** (unsigned int *id, struct Result *result, const int index)

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: The current operational state, hardware configuration, faults, and operating mode.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **rail_clearFaults** (unsigned int *id, struct Result *result, const int index)

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

3.7.16 Relay Entity

See the [Relay Entity](#) for generic information.

group **RelayEntity**

Interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

void **relay_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Set the enable/disable state.

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – False or 0 = Disabled, True or 1 = Enabled

void **relay_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the state.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: False or 0 = Disabled, True or 1 = Enabled

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **relay_getVoltage** (unsigned int *id, struct Result *result, const int index)

Get the scaled micro volt value with reference to ground.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: 32 bit signed integer (in micro Volts) based on the boards ground and reference voltages.

Note: Not all modules provide 32 bits of accuracy. Refer to the module’s datasheet to determine the analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.17 Signal Entity

See the [Signal Entity](#) for generic information.

group **SignalEntity**

Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

void **signal_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enable/Disable the signal output.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – True to enable, false to disable

void **signal_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the Enable/Disable of the signal.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: True to enable, false to disable

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **signal_setInvert** (unsigned int *id, struct Result *result, const int index, const unsigned char invert)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **invert** – True to invert, false for normal mode.

void **signal_getInvert** (unsigned int *id, struct Result *result, const int index)

Get the invert status the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: True to invert, false for normal mode.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **signal_setT3Time** (unsigned int *id, struct Result *result, const int index, const unsigned int t3_nsec)

Set the signal period or T3 in nanoseconds.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **t3_nsec** – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

void **signal_getT3Time** (unsigned int *id, struct Result *result, const int index)

Get the signal period or T3 in nanoseconds.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **signal_setT2Time** (unsigned int *id, struct Result *result, const int index, const unsigned int t2_nsec)

Set the signal active period or T2 in nanoseconds.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **t2_nsec** – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

void **signal_getT2Time** (unsigned int *id, struct Result *result, const int index)

Get the signal active period or T2 in nanoseconds.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.18 Store Entity

See the [Store Entity](#) for generic information.

group StoreEntity

The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

void **store_getSlotState** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Get slot state.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: true: enabled, false: disabled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_loadSlot** (unsigned int *id, struct Result *result, const int index, const unsigned char slot, const unsigned char *buffer, const unsigned short bufferLength)

Load the slot.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **buffer** – The data.
- **bufferLength** – The data length.

void **store_unloadSlot** (unsigned int *id, struct Result *result, const int index, const unsigned char slot, unsigned char *buffer, const unsigned int bufferLength)

Unload the slot data.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length of data that was unloaded. Unloaded length will never be larger than dataLength.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **bufferLength** – The length of buffer buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **buffer** – Byte array that the unloaded data will be placed into.

void **store_slotEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Enable slot.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_slotDisable** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Disable slot.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_getSlotCapacity** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The slot capacity.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_getSlotSize** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The slot size.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_getSlotLocked** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Gets the current lock state of the slot Allows for write protection on a slot.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the locked state.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

- **slot** – The slot number

void **store_setSlotLocked** (unsigned int *id, struct Result *result, const int index, const unsigned char slot, const unsigned char lock)

Sets the locked state of the slot Allows for write protection on a slot.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **slot** – The slot number
- **lock** – state to be set.

3.7.19 System Entity

See the [System Entity](#) for generic information.

group **SystemEntity**

The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc.

void **system_getModule** (unsigned int *id, struct Result *result, const int index)

Get the current address the module uses on the BrainStem network.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The address the module is using on the BrainStem network.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getModuleBaseAddress** (unsigned int *id, struct Result *result, const int index)

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The address the module is using on the BrainStem network.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setRouter** (unsigned int *id, struct Result *result, const int index, const unsigned char address)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **address** – The router address to be used.

void **system_getRouter** (unsigned int *id, struct Result *result, const int index)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The address.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setHBInterval** (unsigned int *id, struct Result *result, const int index, const unsigned char interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

- **interval** – The desired heartbeat delay.

void **system_getHBInterval** (unsigned int *id, struct Result *result, const int index)

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The current heartbeat delay.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setLED** (unsigned int *id, struct Result *result, const int index, const unsigned char ledOn)

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **ledOn** – true: turn the LED on, false: turn LED off.

void **system_getLED** (unsigned int *id, struct Result *result, const int index)

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: true: LED on, false: LED off.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setLEDMaxBrightness** (unsigned int *id, struct Result *result, const int index, const unsigned char brightness)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **brightness** – Brightness value relative to 255

void **system_getLEDMaxBrightness** (unsigned int *id, struct Result *result, const int index)

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum).

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Brightness value relative to 255

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setBootSlot** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **slot** – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

void **system_getBootSlot** (unsigned int *id, struct Result *result, const int index)

Get the store slot which is mapped when the module boots.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The slot number in aSTORE_INTERNAL that is mapped after the module boots.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getVersion** (unsigned int *id, struct Result *result, const int index)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The build version date code.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getBuild** (unsigned int *id, struct Result *result, const int index)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with build.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getModel** (unsigned int *id, struct Result *result, const int index)

Get the module’s model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under “BrainStem

model codes”. Other codes are be used by Acroname for proprietary module types.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

- **value**: The module's model enumeration.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getHardwareVersion** (unsigned int *id, struct Result *result, const int index)

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: The module's hardware version information.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getSerialNumber** (unsigned int *id, struct Result *result, const int index)

Get the module's serial number. The serial number is a unique 32 bit integer which is usually communicated in hexadecimal format.

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value
- **value**: The module's serial number.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_save** (unsigned int *id, struct Result *result, const int index)

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heart-beat interval, module address, module router address.

The result parameter will output the following fields:

- **error**: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

void **system_reset** (unsigned int *id, struct Result *result, const int index)

Reset the system. A return value of aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

void **system_logEvents** (unsigned int *id, struct Result *result, const int index)

Saves system log events to a slot defined by the module (usually ram slot 0).

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

void **system_getUptime** (unsigned int *id, struct Result *result, const int index)

Get the module’s accumulated uptime in minutes

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module’s accumulated uptime in minutes.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getTemperature** (unsigned int *id, struct Result *result, const int index)

Get the module’s current temperature in micro-C

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module’s system temperature in micro-C

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getMinimumTemperature** (unsigned int *id, struct Result *result, const int index)

Get the module's minimum temperature ever recorded in micro-C (uC). This value will persist through a power cycle.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module's minimum system temperature in micro-C

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getMaximumTemperature** (unsigned int *id, struct Result *result, const int index)

Get the module's maximum temperature ever recorded in micro-C (uC). This value will persist through a power cycle.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module's maximum system temperature in micro-C

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getInputVoltage** (unsigned int *id, struct Result *result, const int index)

Get the module's input voltage.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module's input voltage reported in microvolts.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getInputCurrent** (unsigned int *id, struct Result *result, const int index)

Get the module's input current.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module's input current reported in microamps.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getModuleHardwareOffset** (unsigned int *id, struct Result *result, const int index)

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The module address offset.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setModuleSoftwareOffset** (unsigned int *id, struct Result *result, const int index, const unsigned char address)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **address** – The address for the module. Value must be even from 0-254.

void **system_getModuleSoftwareOffset** (unsigned int *id, struct Result *result, const int index)

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The address for the module. Value must be even from 0-254.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.

void **system_getRouterAddressSetting** (unsigned int *id, struct Result *result, const int index)

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The address for the module. Value must be even from 0-254.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_routeToMe** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables/Disables the route to me function. This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enable** – Enable or disable of the route to me function 1 = enable.

void **system_getPowerLimit** (unsigned int *id, struct Result *result, const int index)

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The available power in milli-Watts (mW, 1 t)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getPowerLimitMax** (unsigned int *id, struct Result *result, const int index)

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the power limit in milli-Watts (mW)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setPowerLimitMax** (unsigned int *id, struct Result *result, const int index, const unsigned int power)

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **power** – Limit in milli-Watts (mW) to be set.

void **system_getPowerLimitState** (unsigned int *id, struct Result *result, const int index)

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit().

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the state.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getUnregulatedVoltage** (unsigned int *id, struct Result *result, const int index)

Gets the voltage present at the unregulated port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the voltage in micro-Volts (uV).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getUnregulatedCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the current passing through the unregulated port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the current in micro-Amps (uA).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getInputPowerSource** (unsigned int *id, struct Result *result, const int index)

Provides the source of the current power source in use.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with enumerated representation of the source.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getInputPowerBehavior** (unsigned int *id, struct Result *result, const int index)

Gets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with an enumerated value representing behavior.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setInputPowerBehavior** (unsigned int *id, struct Result *result, const int index, const unsigned char behavior)

Sets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **behavior** – An enumerated representation of behavior to be set.

void **system_getInputPowerBehaviorConfig** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **system_setInputPowerBehaviorConfig** (unsigned int *id, struct Result *result, const int index, const unsigned int *buffer, const unsigned int bufferLength)

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

void **system_getName** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **system_setName** (unsigned int *id, struct Result *result, const int index, const unsigned char *buffer, const unsigned int bufferLength)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

void **system_resetDeviceToFactoryDefaults** (unsigned int *id, struct Result *result, const int index)

Resets the device to it factory default configuration.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

void **system_getLinkInterface** (unsigned int *id, struct Result *result, const int index)

Gets the link interface configuration. This refers to which interface is being used for control by the device.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

- **value:** Variable to be filled with an enumerated value representing interface.
- 0 = Auto= systemLinkAuto
- 1 = Control Port = systemLinkUSBControl
- 2 = Hub Upstream Port = systemLinkUSBHub

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_setLinkInterface** (unsigned int *id, struct Result *result, const int index, const unsigned char linkInterface)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **linkInterface** – An enumerated representation of interface to be set.
 - 0 = Auto= systemLinkAuto
 - 1 = Control Port = systemLinkUSBControl
 - 2 = Hub Upstream Port = systemLinkUSBHub

void **system_getErrors** (unsigned int *id, struct Result *result, const int index)

Gets any system level errors. Calling this function will clear the current errors. If the error persists it will be set again.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bit mapped field representing the devices errors

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **system_getProtocolFeatures** (unsigned int *id, struct Result *result, const int index)

Gets the firmware protocol features

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

- **value**: Value representing the firmware protocol features

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.20 Temperature Entity

See the [Temperature Entity](#) for generic information.

group TemperatureEntity

This entity is only available on certain modules, and provides a temperature reading in microcelsius.

void **temperature_getValue** (unsigned int *id, struct Result *result, const int index)

Get the modules temperature in micro-C

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The temperature in micro-Celsius (1 == 1e-6C).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **temperature_getValueMin** (unsigned int *id, struct Result *result, const int index)

Get the module’s minimum temperature in micro-C since the last power cycle.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module’s minimum temperature in micro-C

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **temperature_getValueMax** (unsigned int *id, struct Result *result, const int index)

Get the module’s maximum temperature in micro-C since the last power cycle.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The module’s maximum temperature in micro-C

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.21 Timer Entity

See the *Timer Entity* for generic information.

group **TimerEntity**

The Timer Class provides access to a simple scheduler. The timer can set to fire only once, or to repeat at a certain interval. Additionally, a timer entity can execute custom Reflex routines upon firing.

void **timer_getExpiration** (unsigned int *id, struct Result *result, const int index)

Get the currently set expiration time in microseconds. This is not a “live” timer. That is, it shows the expiration time originally set with setExpiration; it does not “tick down” to show the time remaining before expiration.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The timer expiration duration in microseconds.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **timer_setExpiration** (unsigned int *id, struct Result *result, const int index, const unsigned int usecDuration)

Set the expiration time for the timer entity. When the timer expires, it will fire the associated timer[index]() reflex.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **usecDuration** – The duration before timer expiration in microseconds.

void **timer_getMode** (unsigned int *id, struct Result *result, const int index)

Get the mode of the timer which is either single or repeat mode.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The mode of the time. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_setBaudRate** (unsigned int *id, struct Result *result, const int index, const unsigned int rate)

Set the UART baud rate. If zero, automatic baud rate selection is used.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **rate** – baud rate.

void **uart_getBaudRate** (unsigned int *id, struct Result *result, const int index)

Get the UART baud rate. If zero, automatic baud rate selection is used.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Pointer variable to be filled with baud rate.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_setProtocol** (unsigned int *id, struct Result *result, const int index, const unsigned char protocol)

Set the UART protocol.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **protocol** – An enumeration of serial protocols.

void **uart_getProtocol** (unsigned int *id, struct Result *result, const int index)

Get the UART protocol.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Pointer to where result is placed.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_setLinkChannel** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Set the index of another UART Entity that should be linked to this UART.

If set to the index of this entity, the channel will not be linked. If set to the index of another UART entity, data will be sent between the two UART entities with no additional processing.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – Index of the UART Entity to link

void **uart_getLinkChannel** (unsigned int *id, struct Result *result, const int index)

Gets the index of the UART Entity that this entity is linked to.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Pointer to where result is placed.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_setStopBits** (unsigned int *id, struct Result *result, const int index, const unsigned char stopBits)

Set the UART stop bit configuration

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **stopBits** – Stop Bits of UART Channel. Allowed options:
 - uartStopBits_1_Value
 - uartStopBits_1p5_Value
 - uartStopBits_2_Value

void **uart_getStopBits** (unsigned int *id, struct Result *result, const int index)

Set the UART stop bit configuration

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Pointer to where result is placed. Possible values:
 - uartStopBits_1_Value
 - uartStopBits_1p5_Value
 - uartStopBits_2_Value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_setParity** (unsigned int *id, struct Result *result, const int index, const unsigned char parity)

Set the UART parity.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **parity** – Parity of UART Channel. Allowed options:
 - uartParity_None_Value
 - uartParity_Odd_Value
 - uartParity_Even_Value
 - uartParity_Mark_Value
 - uartParity_Space_Value

void **uart_getParity** (unsigned int *id, struct Result *result, const int index)

Get the UART parity.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Pointer variable to be filled with value. Possible values:
- uartParity_None_Value
- uartParity_Odd_Value
- uartParity_Even_Value
- uartParity_Mark_Value
- uartParity_Space_Value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_setDataBits** (unsigned int *id, struct Result *result, const int index, const unsigned char dataBits)

Set the number of bits per character

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **dataBits** – Data Bits of UART Channel.

void **uart_getDataBits** (unsigned int *id, struct Result *result, const int index)

Get the number of bits per character

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Pointer to where result is placed.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_setFlowControl** (unsigned int *id, struct Result *result, const int index, const unsigned char flowControl)

Set the UART flow control configuration

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **flowControl** – Flow Control of UART Channel as a bitmask. Allowed bits:
 - uartFlowControl_RTS_CTS_Bit
 - uartFlowControl_DSR_DTR_Bit
 - uartFlowControl_XON_XOFF_Bit

void **uart_getFlowControl** (unsigned int *id, struct Result *result, const int index)

Set the UART flow control configuration

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Pointer to bitmask where result is placed. Possible bits:
 - uartFlowControl_RTS_CTS_Bit
 - uartFlowControl_DSR_DTR_Bit
 - uartFlowControl_XON_XOFF_Bit

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **uart_getCapableProtocols** (unsigned int *id, struct Result *result, const int index)

Returns a bitmask containing a list of protocols that this UART entity is allowed to select. This does not guarantee that selecting a protocol with “setProtocol” will have an available resource.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bitmask containing list of protocols that may be selected. The value of the uartProtocol is mapped to the bit index (e.g. uartProtocol_undefined is bit 0, uartProtocol_ExtronResponder_Value is bit 1, etc.)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.

void **uart_getAvailableProtocols** (unsigned int *id, struct Result *result, const int index)

Returns a bitmask containing a list of protocols that this UART entity is capable of selecting, and has an available protocol resource to assign.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Bitmask containing list of protocols that are available to select. The value of the uartProtocol is mapped to the bit index (e.g. uartProtocol_Undefined is bit 0, uartProtocol_ExtronResponder_Value is bit 1, etc.)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.23 USB Entity

See the *USB Entity* for generic information.

group **USBEntity**

The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

void **usb_setPortEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable both power and data lines for a port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setPortDisable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable both power and data lines for a port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setDataEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setDataDisable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setHiSpeedDataEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setHiSpeedDataDisable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, Hi- Speed (2.0) only.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setSuperSpeedDataEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setSuperSpeedDataDisable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setPowerEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable only the power line for a port without changing the state of the data lines.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setPowerDisable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the power line for a port without changing the state of the data lines.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getPortCurrent** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current through the power line for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel current in micro-amps (1 == 1e-6A).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getPortVoltage** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage on the power line for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel voltage in microvolts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.

- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getHubMode** (unsigned int *id, struct Result *result, const int index)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB hub mode.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usb_setHubMode** (unsigned int *id, struct Result *result, const int index, const unsigned int mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – The USB hub mode.

void **usb_clearPortErrorStatus** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Clear the error status for the given port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The port to clear error status for.

void **usb_getUpstreamMode** (unsigned int *id, struct Result *result, const int index)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

- value: The Upstream port mode.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usb_setUpstreamMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0, usbUpstreamModePort1, and usbUpstreamModeNone.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – The Upstream port mode.

void **usb_getUpstreamState** (unsigned int *id, struct Result *result, const int index)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The Upstream port state.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usb_setEnumerationDelay** (unsigned int *id, struct Result *result, const int index, const unsigned int ms_delay)

Set the inter-port enumeration delay in milliseconds.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.

- **ms_delay** – Millisecond delay in 100mS increments (100, 200, 300 etc.)

void **usb_getEnumerationDelay** (unsigned int *id, struct Result *result, const int index)

Get the inter-port enumeration delay in milliseconds.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Millisecond delay in 100mS increments (100, 200, 300 etc.)

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usb_setPortCurrentLimit** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned int microamps)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a `stem.system.save()` call.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

void **usb_getPortCurrentLimit** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current limit for the port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current limit setting.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel to limit.

void **usb_setPortMode** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned int mode)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of usbPortMode.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel to set the mode on.
- **mode** – The port mode setting as packed bit field.

void **usb_getPortMode** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode, as sub-definitions of usbPortMode.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel.

void **usb_getPortState** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current State for the Port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel.

void **usb_getPortError** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current error for the Port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel.

void **usb_setUpstreamBoostMode** (unsigned int *id, struct Result *result, const int index, const unsigned char setting)

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **setting** – Upstream boost setting 0, 1, 2, or 3.

void **usb_setDownstreamBoostMode** (unsigned int *id, struct Result *result, const int index, const unsigned char setting)

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **setting** – Downstream boost setting 0, 1, 2, or 3.

void **usb_getUpstreamBoostMode** (unsigned int *id, struct Result *result, const int index)

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The current Upstream boost setting 0, 1, 2, or 3.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usb_getDownstreamBoostMode** (unsigned int *id, struct Result *result, const int index)

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The current Downstream boost setting 0, 1, 2, or 3.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usb_getDownstreamDataSpeed** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Filled with the current port data speed
- N/A: usbDownstreamDataSpeed_na = 0
- Hi Speed: usbDownstreamDataSpeed_hs = 1
- SuperSpeed: usbDownstreamDataSpeed_ss = 2

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel to check.

void **usb_setConnectMode** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char mode)

Sets the connect mode of the switch.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.
- **mode** – The connect mode
 - usbManualConnect = 0
 - usbAutoConnect = 1

void **usb_getConnectMode** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Gets the connect mode of the switch.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current connect mode

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setCC1Enable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char enable)

Set Enable/Disable on the CC1 line.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – USB channel.
- **enable** – State to be set
 - Disabled: 0
 - Enabled: 1

void **usb_getCC1Enable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get Enable/Disable on the CC1 line.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: State to be filled
- Disabled: 0
- Enabled: 1

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – USB channel.

void **usb_setCC2Enable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char enable)

Set Enable/Disable on the CC2 line.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – USB channel.
- **enable** – State to be filled
 - Disabled: 0
 - Enabled: 1

void **usb_getCC2Enable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get Enable/Disable on the CC2 line.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: State to be filled
- Disabled: 0
- Enabled: 1

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – USB channel.

void **usb_getCC1Current** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current through the CC1 for a port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The USB channel current in micro-amps (1 == 1e-6A).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getCC2Current** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current through the CC2 for a port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The USB channel current in micro-amps (1 == 1e-6A).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getCC1Voltage** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of CC1 for a port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getCC2Voltage** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of CC2 for a port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setSBUEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char enable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.
- **enable** – The state to be set
 - Disabled: 0
 - Enabled: 1

void **usb_getSBUEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the Enable/Disable status of the SBU

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The enable/disable status of the SBU

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setCableFlip** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char enable)

Set Cable flip. This will flip SBU, CC and SS data lines.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.
- **enable** – The state to be set
– Disabled: 0
– Enabled: 1

void **usb_getCableFlip** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get Cable flip setting.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: The enable/disable status of cable flip.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setAltModeConfig** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned int configuration)

Set USB Alt Mode Configuration.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

void **usb_getAltModeConfig** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get USB Alt Mode Configuration.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB configuration for the given channel.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel

void **usb_getSBU1Voltage** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of SBU1 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getSBU2Voltage** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of SBU2 for a port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

3.7.24 USBSystem Entity

See the [USBSystem Entity](#) for generic information.

group USBSystemEntity

The USBSystem class provides high level control of the lower level Port Class.

void **usbssystem_getUpstream** (unsigned int *id, struct Result *result, const int index)

Gets the upstream port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current upstream port.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbssystem_setUpstream** (unsigned int *id, struct Result *result, const int index, const unsigned char port)

Sets the upstream port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **port** – The upstream port to set.

void **usbsystem_getEnumerationDelay** (unsigned int *id, struct Result *result, const int index)

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: the current inter-port delay in milliseconds.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setEnumerationDelay** (unsigned int *id, struct Result *result, const int index, const unsigned int msDelay)

Sets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **msDelay** – The delay in milliseconds to be applied between port enables

void **usbsystem_getDataRoleList** (unsigned int *id, struct Result *result, const int index)

Gets the data role of all ports with a single call Equivalent to calling PortClass::getDataRole() on each individual port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: A bit packed representation of the data role for all ports.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_getEnabledList** (unsigned int *id, struct Result *result, const int index)

Gets the current enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bit packed representation of the enabled status for all ports.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setEnabledList** (unsigned int *id, struct Result *result, const int index, const unsigned int enabledList)

Sets the enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **enabledList** – Bit packed representation of the enabled status for all ports to be applied.

void **usbsystem_getModeList** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the current mode of all ports with a single call. Equivalent to calling PortClass::getMode() on each port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **usbsystem_setModeList** (unsigned int *id, struct Result *result, const int index, const unsigned int *buffer, const unsigned int bufferLength)

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

void **usbsystem_getStateList** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **usbsystem_getPowerBehavior** (unsigned int *id, struct Result *result, const int index)

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system, i.e. What happens when requested power greater than available power.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setPowerBehavior** (unsigned int *id, struct Result *result, const int index, const unsigned char behavior)

Sets the behavior of how available power is managed, i.e. What happens when requested power is greater than available power.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.

- **index** – The index of the entity in question.
- **behavior** – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

void **usbsystem_getPowerBehaviorConfig** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **usbsystem_setPowerBehaviorConfig** (unsigned int *id, struct Result *result, const int index, const unsigned int *buffer, const unsigned int bufferLength)

Sets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

void **usbsystem_getDataRoleBehavior** (unsigned int *id, struct Result *result, const int index)

Gets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setDataRoleBehavior** (unsigned int *id, struct Result *result, const int index, const unsigned char behavior)

Sets the behavior of how upstream and downstream ports are determined, i.e. How do you manage requests for data role swaps and new upstream connections.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **behavior** – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

void **usbsystem_getDataRoleBehaviorConfig** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine priority host priority.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value
- value: Length that was actually received and filled.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filled

void **usbsystem_setDataRoleBehaviorConfig** (unsigned int *id, struct Result *result, const int index, const unsigned int *buffer, const unsigned int bufferLength)

Sets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine host priority.

The result parameter will output the following fields:

- error: *Common EntityClass Return Values* common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.

- **index** – The index of the entity in question.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

void **usbsystem_getSelectorMode** (unsigned int *id, struct Result *result, const int index)

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Variable to be filled with the selector mode

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setSelectorMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **mode** – Mode to be set.

void **usbsystem_getUpstreamHS** (unsigned int *id, struct Result *result, const int index)

Gets the USB HighSpeed upstream port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current upstream port.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setUpstreamHS** (unsigned int *id, struct Result *result, const int index, const unsigned char port)

Sets the USB HighSpeed upstream port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **port** – The upstream port to set.

void **usbsystem_getUpstreamSS** (unsigned int *id, struct Result *result, const int index)

Gets the USB SuperSpeed upstream port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: The current upstream port.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setUpstreamSS** (unsigned int *id, struct Result *result, const int index, const unsigned char port)

Sets the USB SuperSpeed upstream port.

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **port** – The upstream port to set.

void **usbsystem_getOverride** (unsigned int *id, struct Result *result, const int index)

Gets the current enabled overrides

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Bit mapped representation of the current override configuration.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setOverride** (unsigned int *id, struct Result *result, const int index, const unsigned int overrides)

Sets the current enabled overrides

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **overrides** – Overrides to be set in a bit mapped representation.

void **usbsystem_setDataHSMaxDatarate** (unsigned int *id, struct Result *result, const int index, const unsigned int datarate)

Sets the USB HighSpeed Max datarate

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **datarate** – Maximum datarate for the USB HighSpeed signals.

void **usbsystem_getDataHSMaxDatarate** (unsigned int *id, struct Result *result, const int index)

Gets the USB HighSpeed Max datarate

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Current maximum datarate for the USB HighSpeed signals.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

void **usbsystem_setDataSSMaxDatarate** (unsigned int *id, struct Result *result, const int index, const unsigned int datarate)

Sets the USB SuperSpeed Max datarate

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code.
- **index** – The index of the entity in question.
- **datarate** – Maximum datarate for the USB SuperSpeed signals.

void **usbsystem_getDataSSMaxDatarate** (unsigned int *id, struct Result *result, const int index)

Gets the USB SuperSpeed Max datarate

The result parameter will output the following fields:

- error: [Common EntityClass Return Values](#) common entity return value
- value: Current maximum datarate for the USB SuperSpeed signals.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Output object containing result code and the requested value if successful.
- **index** – The index of the entity in question.

3.7.25 Module Entity

group **ModuleEntity**

The Module Entity provides a generic interface to a BrainStem hardware module. The Module Class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

void **module_createStem** (unsigned int *id, struct Result *result, unsigned char moduleAddress, bool autoNetworking, unsigned char model)

Creates a brainstem object that the library will manage internally and creates a unique identifier that will be used for other function calls in this library.

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_disconnectAndDestroyStem** (unsigned int *id, struct Result *result)

Disconnects from device defined by the ID and will destroy any internal memory associated with the Device.

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_discoverAndConnect** (unsigned int *id, struct Result *result, int transport, unsigned int serialNumber)

Finds and connects to the first device found on the given transport. If a serial number was provided when module_createStem was called then it will only connect to that specific id.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR or a non zero Error code.
- **transport** – Defines what connection method should be searched for BrainStem devices. (i.e. USB, TCPIP, etc.)

void **module_sDiscover** (unsigned int *id, struct Result *result, struct *linkSpec_CCA* *stemList, int listLength, unsigned int networkInterface, int transport)

Discovers all of the BrainStem devices on a given transport. The return list is filled with device specifiers which contains information about the device.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR or a non zero Error code.
- **stemList** – List of device specifiers for each of the discovered devices
- **listLength** – Indicates how long the list is.
- **networkInterface** – Defines the network interface to use when multiple are present. A value of 0 or LOCALHOST_IP_ADDRESS will result in local searches only. Values other than this will have firewall implications.
- **transport** – Defines what connection method should be searched for BrainStem devices. (i.e. USB, TCPIP, etc.)

void **module_disconnect** (unsigned int *id, struct Result *result)

Disconnects device, but does not destroy underlying object. i.e. “module_reconnect” could be called without calling “module_createStem” again.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_reconnect** (unsigned int *id, struct Result *result)

Reestablishes a connection with a preexisting stem object. The original stem would of been created with “module_createStem”.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_connectThroughLinkModule** (unsigned int *id, unsigned int *id_linkStem, struct Result *result)

Establishes connection through another stems link/connection (i.e. transport: USB, TCPIP). Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through “module_createStem”
- **id_linkStem** – The link stem’s id assigned through “module_createStem” (The stem providing the connection.)
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_setModuleAddress** (unsigned int *id, struct Result *result, int address)

Changes the module address of the stem object that was created via “module_createStem”. Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR or a non zero Error code.
- **address** – New address to be set.

void **module_getModuleAddress** (unsigned int *id, struct Result *result)

Retrieves the module address of the stem object that was created via “module_createStem”. Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR and the module/stems current module address or a non zero Error code.

void **module_isConnected** (unsigned int *id, struct Result *result)

Returns the current state of the module/stem’s connection. Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR and the status of the connection or a non zero Error code. (0 = disconnected; 1 = connected.)

void **module_setNetworkingMode** (unsigned int *id, struct Result *result, int mode)

Changes the networking mode of the stem object. Auto mode is enabled by default which allows automatic adjustment of the module/stems networking configuration. Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – object, containing NO_ERROR or a non zero Error code.
- **mode** – New mode to be set.

void **module_clearAllStems** ()

Disconnects and destroys all modules/stems. During development the dll doesn’t always “detach” between runs. This can create problematic scenarios if there are not subsequent disconnect and destroy calls for every create call made. This function can be a helpful post-execution/tear-down process when bringing up new BrainStem Networks. However, not required if connections are handles correctly. Refer to BrainStem Networking at www.acroname.com/support

3.7.26 Link Entity

group **LinkEntity**

The Link class provides link level access to streaming information.

struct **linkSpec_CCA**

CCA linkSpec structure - It contains the necessary information for connecting to a BrainStem module.

Public Members

unsigned int **type**

The transport type of this spec.

unsigned int **serial_num**

The serial number of the module

unsigned int **module**

The module address

unsigned int **router**

The BrainStem network router address

unsigned int **router_serial_num**

The BrainStem network router serial number

unsigned int **model**

The model type

unsigned int **usb_id**

The usb_id of the BrainStem module.

unsigned int **ip_address**

The IP4 address of the module.

unsigned int **ip_port**

The TCP port for socket connection on the module.

unsigned int **baudrate**

The serial port baudrate

char **port**[100]

The serial port path or name

struct **StreamStatusEntry_CCA**

StreamStatusEntry structure - It contains members of streaming entries in the form of key value pairs. Keys are comprised of the devices module address, command, option, index, and subindex API values.

Public Members

unsigned long long **key**

The stream key (64bit).

unsigned int **value**

The value associated with the key (32bit).

void **link_enableStream** (unsigned int *id, struct Result *result, const unsigned char moduleAddress, const unsigned char cmd, const unsigned char option, const unsigned char index, const bool enable)

Enables streaming for the supplied criteria.

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **moduleAddress** – Address to filter on.
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - Enables streaming; False - disables streaming

void **link_getLinkSpecifier** (unsigned int *id, struct Result *result, struct [linkSpec_CCA](#) *spec)
Get linkSpecifier

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **spec** – - allocated linkspec struct will be filled with spec.

void **link_registerStreamCallback** (unsigned int *id, struct Result *result, const unsigned char moduleAddress, const unsigned char cmd, const unsigned char option, const unsigned char index, const bool enable, cStreamCallback_t cb, void *pRef)

Registers a callback function based on a specific module, cmd, option, and index.

::aErrNotFound - Item not found (uninstalling only)

::aErrNone - success

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - installs/updates callback and ref; False - uninstalls callback

- **cb** – Callback to be executed when a new packet matching the criteria is received.
- **pRef** – Pointer to user reference for use inside the callback function.

void **link_getStreamStatus** (unsigned int *id, struct Result *result, const unsigned char moduleAddress, const unsigned char cmd, const unsigned char option, const unsigned char index, const unsigned char subindex, struct [StreamStatusEntry_CCA](#) *buffer, const unsigned int bufferLength)

Gets all available stream values based on the search criteria.

::aErrParam if status or unloadedSize is null

::aErrNone - success

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **subindex** – subindex to filter by (supports Wildcards)
- **buffer** – Buffer of user allocated memory to be filled with stream data Note: Link::getStreamKeyElement should be used to decode the keys
- **bufferLength** – Number of elements the buffer can hold.

void **link_getStreamKeyElement** (struct Result *result, const unsigned long long key, const unsigned char element)

Convenience function to unpack a stream key.

Parameters

- **key** – The key to be unpacked
- **element** – The element to unpack from the key.

3.7.27 PDChannelLogger

group PDChannelLogger

PDChannelLogger: Provides an interface for managing BrainStem Power Delivery Packets. Packets are accepted and decoded asynchronously. Pay careful attention to packet cleanup. Portions of the structure do not belong to the caller.

struct **BS_PD_Packet_CCA**

BrainStem Power Delivery Packet Structure - Contains information representing a Power Delivery packet along with contextual device information

Public Members

unsigned char **channel**

Channel/Index

unsigned int **seconds**

Seconds in device time since power on.

unsigned int **uSeconds**

Micro Seconds in device time since power on .

unsigned char **direction**

Direction of packet transmission relative to the device.

unsigned char **sop**

See bs_pd_packet.h for more details

unsigned int **event**

Packet type - See powerdeliveryLogEvent in aProtocolDefs.h

unsigned int **payloadSize**

Length of the payload.

unsigned char ***payload**

Raw PD Packet data

unsigned char **ccChannel**

CC Channel type

unsigned int **crc**

CRC of payload

void **PDChannelLogger_create** (unsigned int *id, struct Result *result, const int index, const unsigned int bufferLength)

Creates internal object for managing BrainStem Power Delivery logging packets.

Returns *common entity* return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.
- **bufferLength** – Number of packets the class should queue before dropping.

void **PDChannelLogger_destroy** (unsigned int *id, struct Result *result, const int index)

Destroys internal object for managing BrainStem Power Delivery logging packets.

Returns *common entity* return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.

void **PDChannelLogger_setEnabled** (unsigned int *id, struct Result *result, const int index, const bool enable)

Enables Power Delivery logging.

True on success.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.
- **enable** – True enables logging; False disables logging

void **PDChannelLogger_getPacket** (unsigned int *id, struct Result *result, const int index, struct *BS_PD_Packet_CCA* *packet)

Attempts to takes a packet from the internal buffer.

True if the function successfully acquired any number of packets. False if no packets were available.

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.
- **packet** – Reference to a packet to be filled by the function.

void **PDChannelLogger_getPackets** (unsigned int *id, struct Result *result, const int index, struct *BS_PD_Packet_CCA* *packetBuffer, const unsigned int bufferLength)

Attempts to take a multiple packets (up to a maximum) from the internal buffer.

True if the function successfully acquired any number of packets. False if no packets were available.

Parameters

- **id** – ID assigned through “module_createStem”

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.
- **packetBuffer** – pointer to a buffer to be filled
- **bufferLength** – The length of the buffer provided.

void **PDChannelLogger_freePayloadBuffer** (unsigned int *id, struct Result *result, struct *BS_PD_Packet_CCA* *packet)

Releases the internal memory contained within the payload parameter.

Returns *common entity* return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **packet** – Reference to a packet to have its memory freed.

3.7.28 PortMapping

group PortMapping

PortMapping Class: Provides an interface for usb descriptor information of devices downstream of Acroname hub products.

struct DeviceNode_CCA

Device Node Structure - Contains information linking the downstream device to the Acroname Hub.

Public Members

unsigned int **hubSerialNumber**

Serial number of the Acroname hub where the device was found.

unsigned char **hubPort**

Port of the Acroname hub where the device was found.

unsigned short **idVendor**

Manufactures Vendor ID of the downstream device.

unsigned short **idProduct**

Manufactures Product ID of the downstream device.

unsigned char **speed**

The devices downstream device speed. (PORT_SPEED_CCA_t)

char **productName**[255]
USB string descriptor

char **serialNumber**[255]
USB string descriptor

char **manufacturer**[255]
USB string descriptor

void **portMapping_getDownstreamDevices** (struct Result *result, struct *DeviceNode_CCA* *buffer, const unsigned int bufferSize)

Gets downstream device USB information for all Acroname hubs.

::aErrNone on success

::aErrParam: Passed in values are not valid. (NULL, size etc).

::aErrMemory: No more room in the list

::aErrNotFound: No Acroname devices were found.

Parameters

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The number of devices found.
- **buffer** – Pointer to the start of a list/array to be used by the function.
- **bufferLength** – Size of the list/array in *DeviceNode_CCA*'s, not bytes.

3.7.29 Version

group Version

Functions for getting and comparing software and firmware version.

void **version_ParseMajor** (struct Result *result, unsigned int build)
Parses the major revision level from the given build number.

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_ParseMinor** (struct Result *result, unsigned int build)
Parses the minor revision level from the given build number.

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_ParsePatch** (struct Result *result, unsigned int build)
Parses the revision patch level from the given build number.

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_IsLegacyFormat** (struct Result *result, unsigned int build)

Check if the given build version is of the legacy packing format

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_GetMajor** (struct Result *result)

Return the major revision number for the software package.

Parameters

result – Object containing aErrNone and the requested value on success.

void **version_GetMinor** (struct Result *result)

Return the minor revision number for the software package.

Parameters

result – Object containing aErrNone and the requested value on success.

void **version_GetPatch** (struct Result *result)

Return the patch revision number for the software package.

Parameters

result – Object containing aErrNone and the requested value on success.

void **version_IsAtLeast** (struct Result *result, unsigned int major, unsigned int minor, unsigned int patch)

Check that the current software version is at least major.minor.patch

Parameters

- **result** – Object containing aErrNone and the requested value on success.
- **major** – The major revision level.
- **minor** – The minor revision level.
- **patch** – The patch revision level.

void **version_IsAtLeastCompare** (struct Result *result, unsigned int major_lhs, unsigned int minor_lhs, unsigned int patch_lhs, unsigned int major_rhs, unsigned int minor_rhs, unsigned int patch_rhs)

Check that the supplied left hand side (lhs) version is at least (\geq) the right hand side (rhs).

Parameters

- **result** – Object containing aErrNone and the requested value on success.
- **major_lhs** – The lhs major revision level.
- **minor_lhs** – The lhs minor revision level.
- **patch_lhs** – The lhs patch revision level.
- **major_rhs** – The rhs major revision level.
- **minor_rhs** – The rhs minor revision level.
- **patch_rhs** – The rhs patch revision level.

void **version_Pack** (struct Result *result, unsigned int major, unsigned int minor, unsigned int patch)
Packs the given version into a single integer

Parameters

- **result** – Object containing aErrNone and the requested value on success.
- **major** – The major revision level.
- **minor** – The minor revision level.
- **patch** – The patch revision level.

3.8 LabVIEW API Reference

The LabVIEW API has been renamed to the CCA API. Generally speaking the functionality is the same with some exceptions which are outlined in the 2.11.x release notes. Please refer to the CCA API reference for information regarding the LabVIEW API.

[CCA API Reference](#)

3.9 Q-Sys API Reference

Welcome to the BrainStem Q-Sys API reference documentation. This interface provides access to the BrainStem software interfaces.

Changes that do not break a Q-Sys API will not constitute a major version change. Removing features or making significant changes will result in a minor version change. Deploying breaking changes may be necessary for legal, performance, or security reasons. When a new version is released, the current version will remain supported for one year and may become unavailable anytime after this period.

3.9.1 Product Plugins

Table 22: Plugin Product Availability

Product	USBHub3c
<i>usbhub3c</i>	RS232 interface for control and telemetry

USBHub3c Interfaces

Release Notes

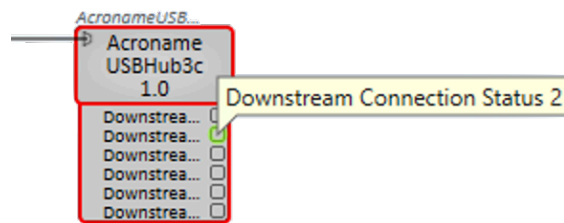
```
# 1.0.0.0
- Initial Release
  - Upstream port switching control
  - Downstream port enable and disable
  - Port enable status
  - USBHub3c firmware version and model information
```

Interface Requirements

A valid Acroname USBHub3c with the Serial Control add-on software feature must be installed. For additional information is available [USBHub3c](#).

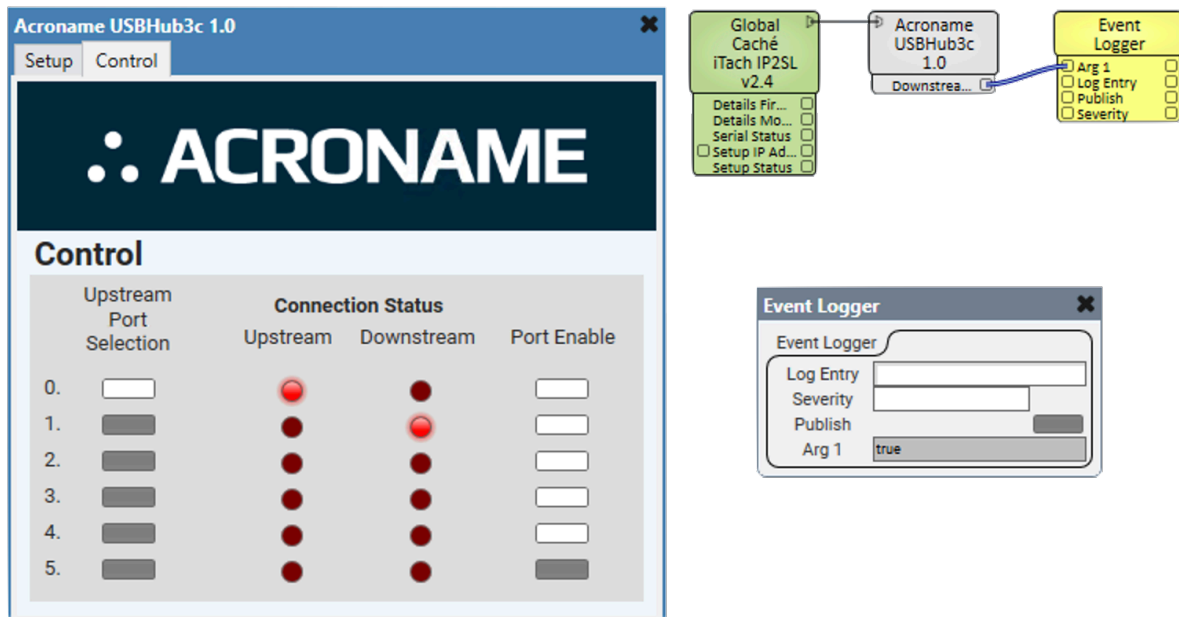
Downstream Connection Status

The downstream connection status interface indicates whether a port is enabled or disabled. The return value is “true” when a USB device is connected, and “false” when disconnected. An active host is required for USB devices to be connected.



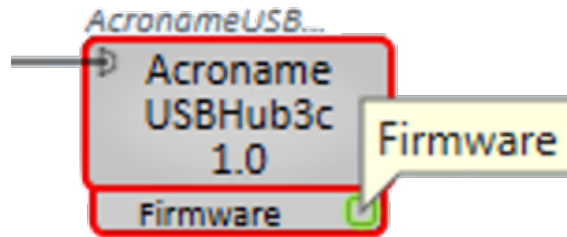
Example Read of Downstream Connection Status Information

Reading the downstream connection status information passed into debugger logger demonstrates reading the model value. Arg1 shows the model read information into the Event Logger.



Firmware Information

Current firmware information can be retrieved as a string with a format of `MAJOR.MINOR.PATCH.REVISION`. The value is equivalent of calling the *Module Entities* BrainStem System Entity call for `get Version`.

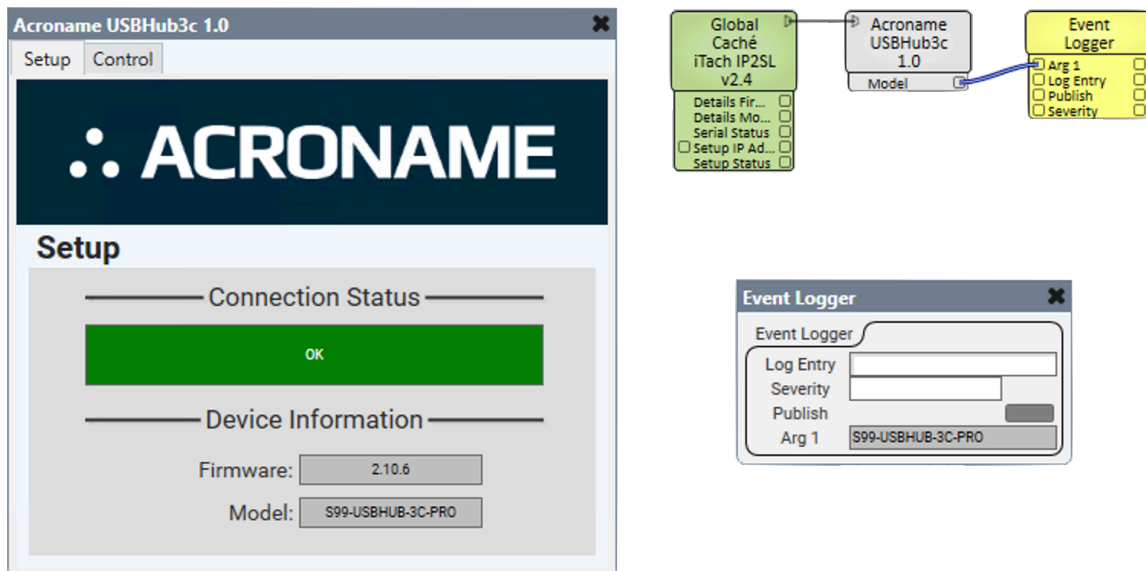


Model Information

Model information can be retrieved as a string with a format of `S99-USBHUB-3C-PRO` or `S99-USBHUB-3C-LAB`.

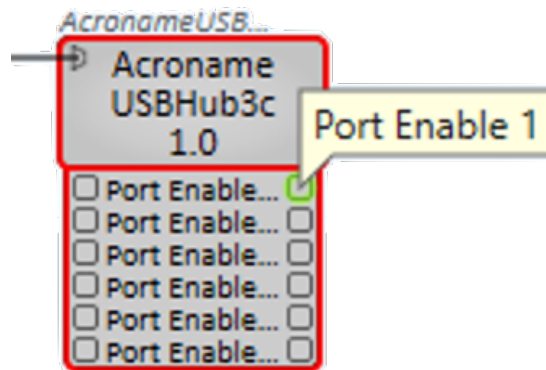
Example Read of Model Information

Reading the model information passed into debugger logger demonstrates reading the model value. Arg1 shows the model read information into the Event Logger.



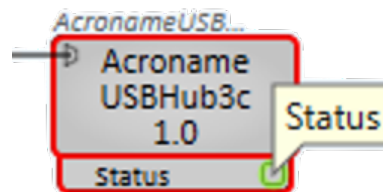
Port Enable

The “port-enable” interfaces enable turning a specific port on or off. When a port is enabled, both power and data lines become active. Conversely, power and data lines get disconnected when the port is disabled, simulating the removal of a USB cable. Note that for USB-C ports, the CC line will be activated to negotiate the appropriate power level for Vbus power. Vbus will not be active until proper CC negotiation has occurred.



Status

Indication of the plugin connection status to the USBHub3c.



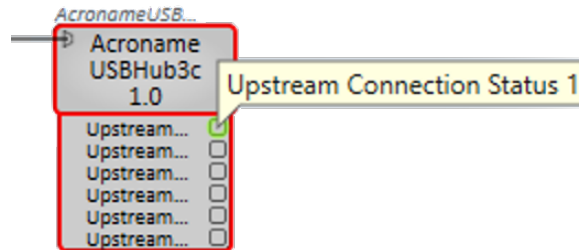
Status Codes

The following is a list of all status codes the USBHub3c supports with descriptions.

Status	Description
0	Connection established or data is received successfully and everything is OK.
1	Compromised connection.
2	Connection or data received with an error indicating a fault.
3	Missing serial port to connect to device.
4	Plugin initializing.

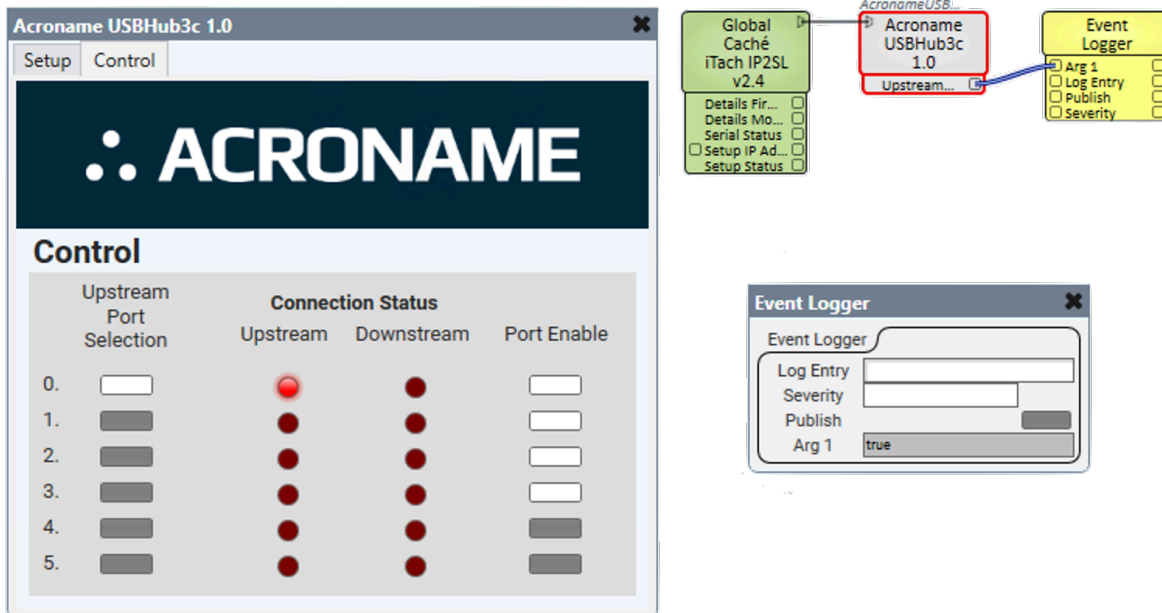
Upstream Connection Status

The “upstream connection status” indicates that a valid USB host is connected to the device. If the connection is established, the output value of the plugin will be “true”, which means the device is ready to communicate with the host. However, if the upstream host connection is removed, the output value will be “false”, indicating that the communication between the device and the host is no longer available.



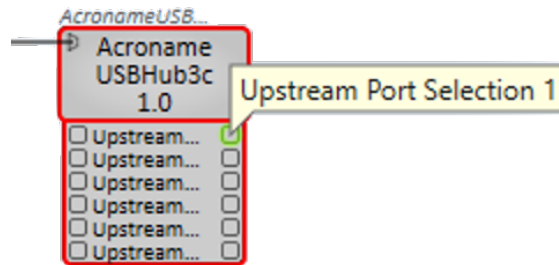
Example Read of Upstream Connection Status Information

Reading the upstream connection status information passed into debugger logger demonstrates reading the model value. Arg1 shows the model read information into the Event Logger.



Upstream Port Selection

The interface for selecting the upstream port determines which port to use as an upstream interface. When calling the function, any input value below 0 will be set to 0.



Version

The `version` endpoint shows version information for the Q-Sys plugin version.

4.1 What is BrainStem

Acroname's BrainStem technology is a foundational tool for bridging the gap between hardware and software systems. BrainStem controllers gather and relay peripheral information to other systems, which may be a host computer, additional microcontrollers or an array of sensors and testing equipment. The core BrainStem concept is metaphorically based on the human nervous system's data gathering, interpreting and transmitting processes.

4.1.1 Scalable

BrainStem modules are the ideal controllers for hierarchical control and autonomous systems. Using a robust networking protocol, BrainStem devices relay information across industry standardized interfaces such as serial, I2C, USB, Ethernet and BlueTooth. Each controller has an embedded virtual machine kernel that allows for rich embedded application execution, reflexive software creation and direct hardware access using a structured packet format. The BrainStem network can support more than 100 microcontrollers, and each microcontroller can have several subordinate networks of devices, sensors or interfaces.

4.1.2 Usable

The BrainStem platform and associated APIs provide powerful desktop computer access to sensors and actuators. Since these devices exist across such a wide application range, BrainStem modules are designed to be as generalized and adaptable as possible. This includes cross platform interface software, defining and publishing interconnect standards and protocols, selecting a common form factor and focusing on expandability.

4.1.3 Next Steps

The best place to get started with your new BrainStem hardware is by checking out the [Getting Started](#) section of our documentation.

4.2 aEther

4.2.1 Overview

What is it?

aEther is a software feature that encapsulates the BrainStem software object and provides synchronization across multiple threads, processes and computers by way of a client-server model.

How does aEther help me utilize Acroname hardware?

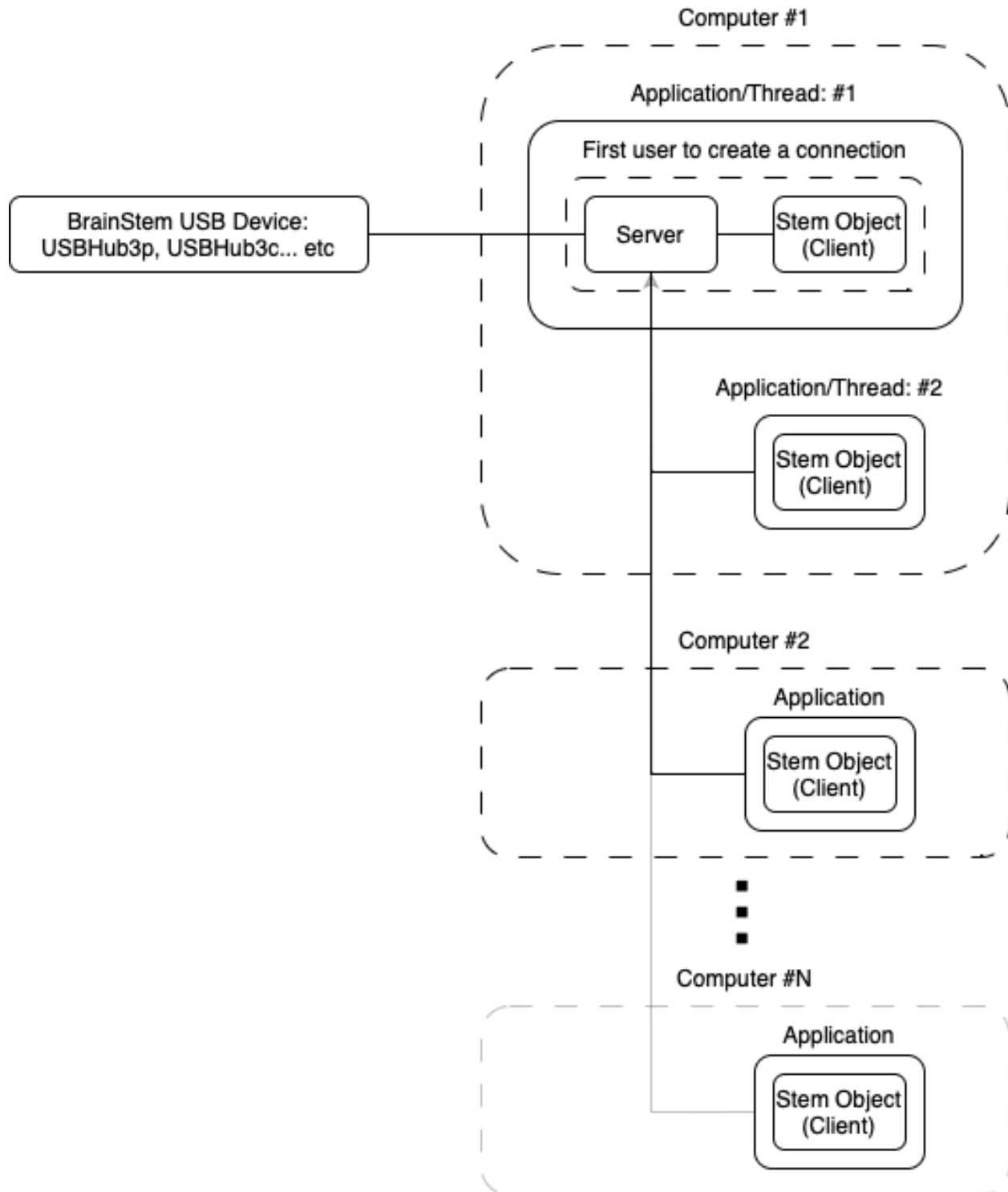
1. **Simplified Synchronization:** Tired of working with Mutexes? Simply create and connect to an object for each thread and we will handle the rest.
2. **Improved Debugging:** Want to know what's going on with Acroname hardware while your application is running or paused on a break point? No problem, just open up HubTool and monitor the devices state.
3. **Remote Access:** Is remote desktop too sluggish or overkill? With the proper network configuration you can access your devices from the other side of the world.

4.2.2 Communication Models

Models define a specific pattern or behavior. These behaviors have different attributes that might be more applicable for specific situations. The BrainStem protocol supports multiple communications models. By default the Client-Server model (aEther) is used because it is the most user friendly and adaptable.

Client-Server Model (aEther)

The Client-Server model works by routing all traffic through the operating systems socket layer. When the first connection is made a "server" and "client" pair will be created. All subsequent connections will be "clients" only. Note that the server is owned by the first user to connect to the device. Deletion of that object will result in ALL clients being disconnected. The server encapsulates the device by way of the direct connection model (discussed below). By doing this the server can manage simultaneous access of many clients.



Configuring the device for external exposure

By default all sockets are restricted to the localhost. In other words, by default these sockets do NOT have access to the outside world and therefore should not trigger any firewall warnings. In the event you do want the device to be accessible remotely it simply needs to be enabled. Generally speaking this only exposes it to your local network; further exposure to the internet will depend on configurations outside of BrainStem.

C++

```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.localOnly = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config_result = stem.getConfig()
config = config_result.value
config.localOnly = False

#NOTE: This must be done before connection
stem.setConfig(config)

stem.discoverAndConnect(brainstem.link.Spec.USB)
```

Configuring the device for a specific network adapter

Sometimes it is necessary to define the network interface you would like to use. If your system has many network adapters (including virtual ones) you might need to programmatically configure this setting. Typically, this is only common for the following cases.

1. Hosting a device for external access (localOnly = False) - By default the first interface will be used. Typically, this is the default interface. If the default is not the desired interface you will need to configure it as shown below.
2. Discovery - By default discovery will search the localhost only; however, if you provide a valid network interface it will search that instead. This is particularly important if using stem.discoverAndConnect() for a remote device that is not directly connected to the computer. The network interface of the module must be set before calling a connect function as shown below.

Note that you can programmatically request the available interfaces on your machine through “aDiscovery_GetIPv4Interfaces”

C++

```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.networkInterface = 0xA00A8C0; //Example: 192.168.0.10 (network byte order)
config.localOnly = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config_result = stem.getConfig()
config = config_result.value
config.networkInterface = 0xA00A8C0 #Example: 192.168.0.10 (network byte order)
config.localOnly = False

#NOTE: This must be done before connection
stem.setConfig(config)

stem.discoverAndConnect(brainstem.link.Spec.USB)
```

Disabling auto fallback

In order to make a more pleasant user experience, the “discoverAndConnect” process will automatically check other transport types for devices if a device is unavailable with the given transport type. For instance, we want to create 2x BrainStem software objects that connect to the same USB module (ie USBHub3p etc). The first caller should use the transport type “USB” while the second should use “AETHER”. When fallback is enabled, this can be ignored and the underlying software will automatically determine the appropriate transport for you. If you use the “USB” transport and it is determined that someone is already connected to this device it will automatically try the “AETHER” transport next. Depending on your situation you might not want this behavior. If so, it can be disabled through the NetworkConfig.

C++

```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.fallback = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config_result = stem.getConfig()
config = config_result.value
config.fallback = False

#NOTE: This must be done before connection
stem.setConfig(config)

stem.discoverAndConnect (brainstem.link.Spec.USB)
```

Disabling the Client-Server model

There might be specific situations in which you want to disable the client-server model and revert to the previous direct connection model paradigm. This is very simply done through the aEtherConfig structure.

C++

```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.enabled = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect (USB)
```

Python

```
import brainstem

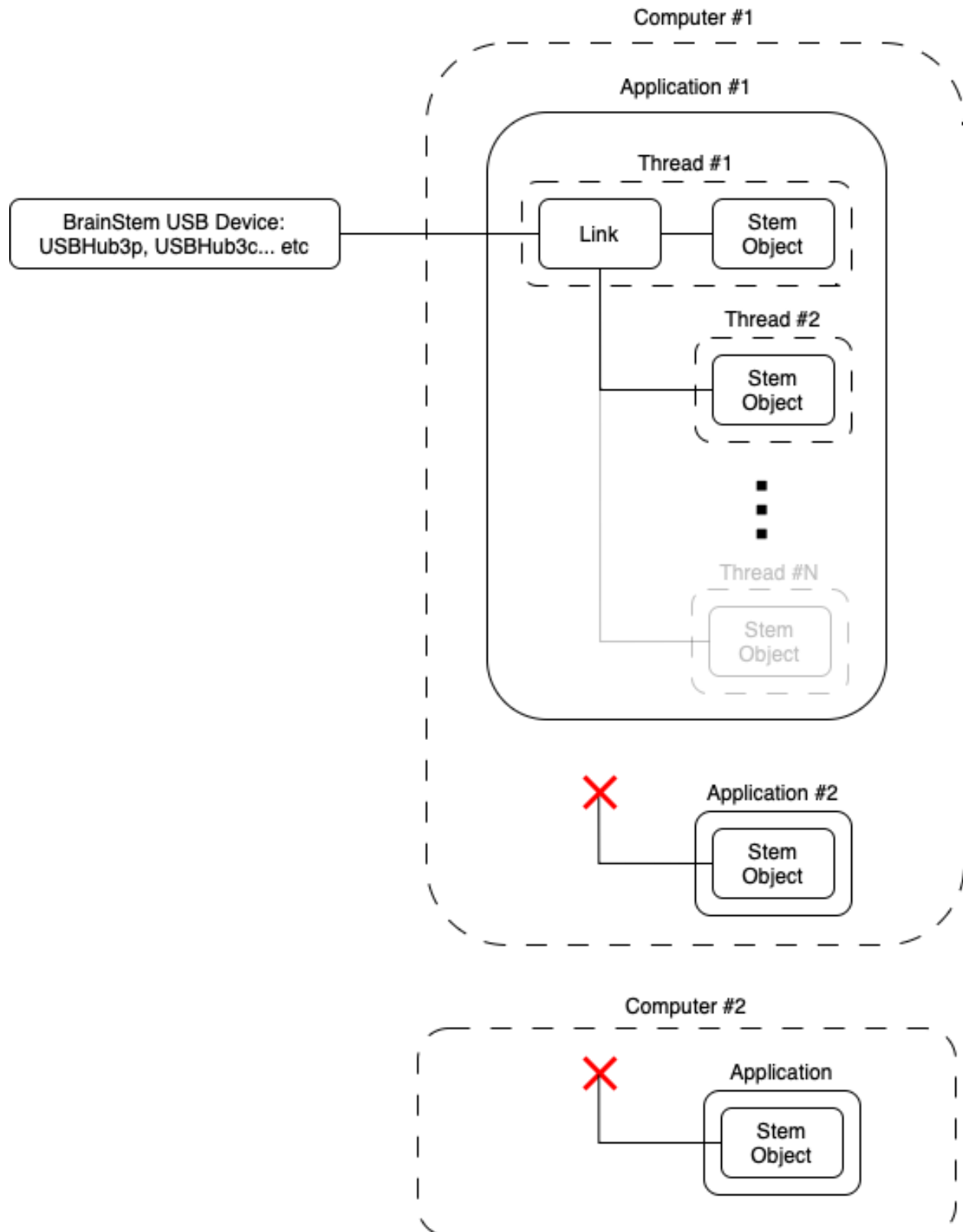
stem = brainstem.stem.USBHub3c()
config_result = stem.getConfig()
config = config_result.value
config.enabled = False

#NOTE: This must be done before connection
stem.setConfig(config)

stem.discoverAndConnect (brainstem.link.Spec.USB)
```

Direct Connection Model

The direct connect model is Acroname's original model and is the core of the server in the client-server model. In this model the BrainStem software object owns the device communications path and is generally thread safe. When using this model the device is NOT accessible in any other processes nor is it accessible by another computer. This model is slightly faster, but more restrictive.



To select this configuration it must be done through the aEtherConfig structure.

C++


```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.enabled = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config_result = stem.getConfig()
config = config_result.value
config.enabled = False

#NOTE: This must be done before connection
stem.setConfig(config)

stem.discoverAndConnect(brainstem.link.Spec.USB)
```

4.3 Getting Started

Before you begin you'll need to collect the following items.

- A BrainStem Module with development board or an Acroname Hub/Switch.
- A Link Transport Cable (Ethernet or USB).
- The BrainStem Support software package.

BrainStem Devices and Development boards can be purchased from the [Acroname Products](#).

The latest version of the [BrainStem Development Kit \(BDK\)](#)¹²⁶ or [HubTool](#)¹²⁷ can be downloaded from the [Acroname Download Page](#). Extract the package to the location of your choice when the download has finished.

4.3.1 Do I need Drivers?

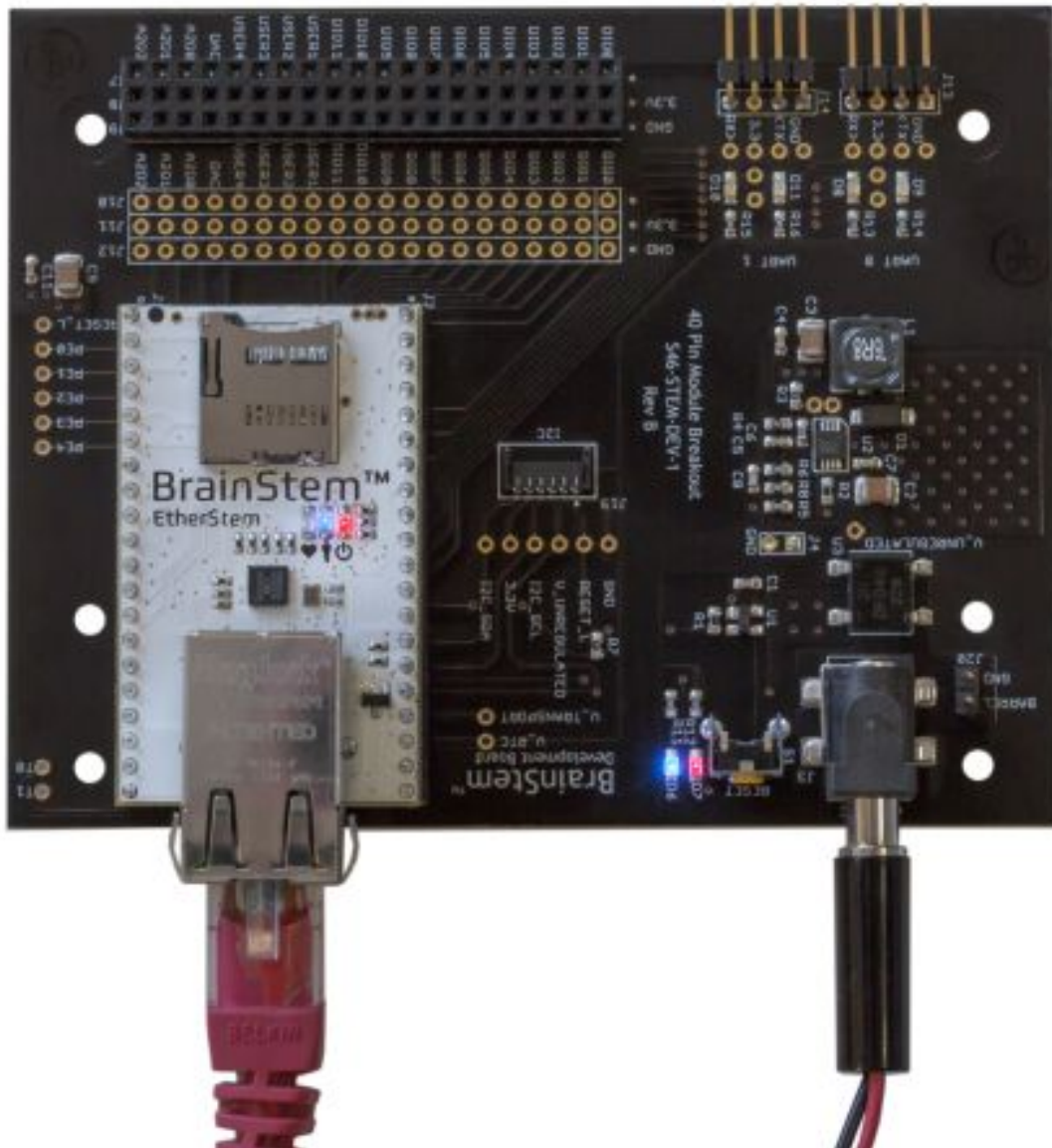
Acroname kernel drivers are no longer required for most modern operating systems; However, if you are running Windows 7 or Linux, please read the [BrainStem USB installation instructions](#) for your particular operating system.

¹²⁶ <https://acroname.com/api>

¹²⁷ <https://acroname.com/hubtool>

4.3.2 Connecting to a BrainStem Device

With the exception of our Hubs/Switches most BrainStem devices must be plugged into the BrainStem Development Board to receive power. Once the device has been plugged into a development board it is safe to insert the link transport cable. The red power LED should be on and the green heartbeat LED should flicker rapidly when power is first applied. If you are using a Hub/Switch connect the supplied power adapter to the device.



4.3.3 Launch HubTool

For BrainStem devices and Hubs navigate to the bin folder of the BrainStem Development Kit to find HubTool.

4.3.4 Toggling the LED

HubTool will periodically search for connected devices and display them in the lower right corner. Selecting your device will cause a UI to be created for that device based on its capabilities. Click the LED button and observe the illumination of the user LED on the device.

Updating your module firmware with Updater.

We are constantly fixing bugs and improving our products. It's good practice to keep your modules up-to-date with the latest firmware. Please see the [BrainStem Firmware Management](#) to update your firmware.

Introduction to the C++ API.

Interested in communicating with your BrainStem module from a host computer via the [C++ API](#)? See the [Getting Started C++ guide](#)

Introduction to the Python API.

Want to work with Python? See the [Python API](#) section of this reference to get started.

4.4 Firmware Management

4.4.1 Updating Firmware via HubTool

BrainStem firmware can be loaded using the HubTool application, which is available as a separate download package. HubTool can be downloaded from the [Software page](#) on Acroname's website.

When HubTool is opened, click the "Firmware Management" button to switch to the Updater window:

When the Updater window is opened, it will automatically populate with a list of all attached Acroname devices. If a new device is attached after this window is open, the "Discover" button will refresh the device list.

Each device entry in the Updater shows the model name, the serial number, the current firmware version, the firmware version to update to, and the status of the update. The update version will automatically select the most recent release. To select a different version, click the Target version field and click the desired version:

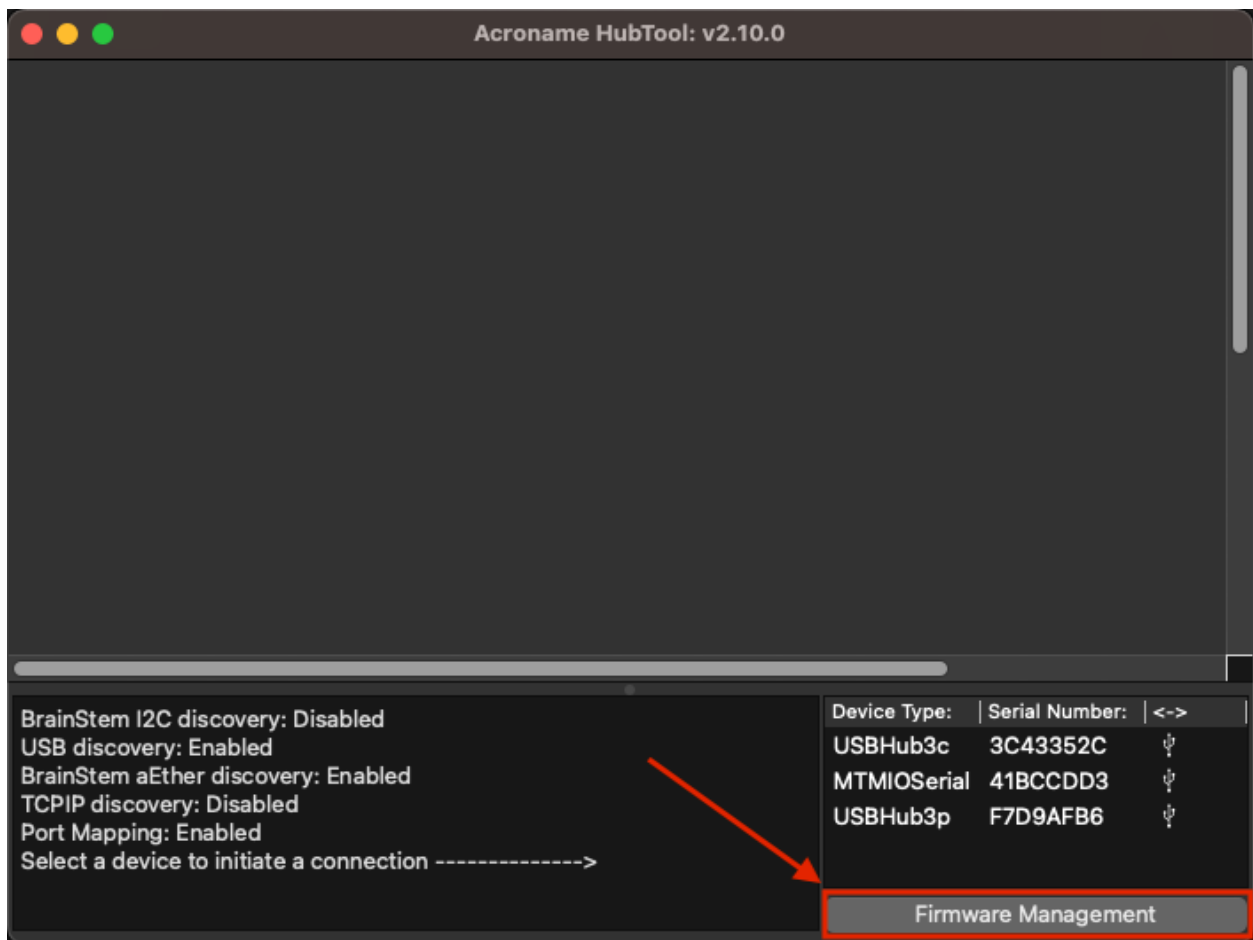
When the correct target version for a device has been selected, press the "Update" button for that device:

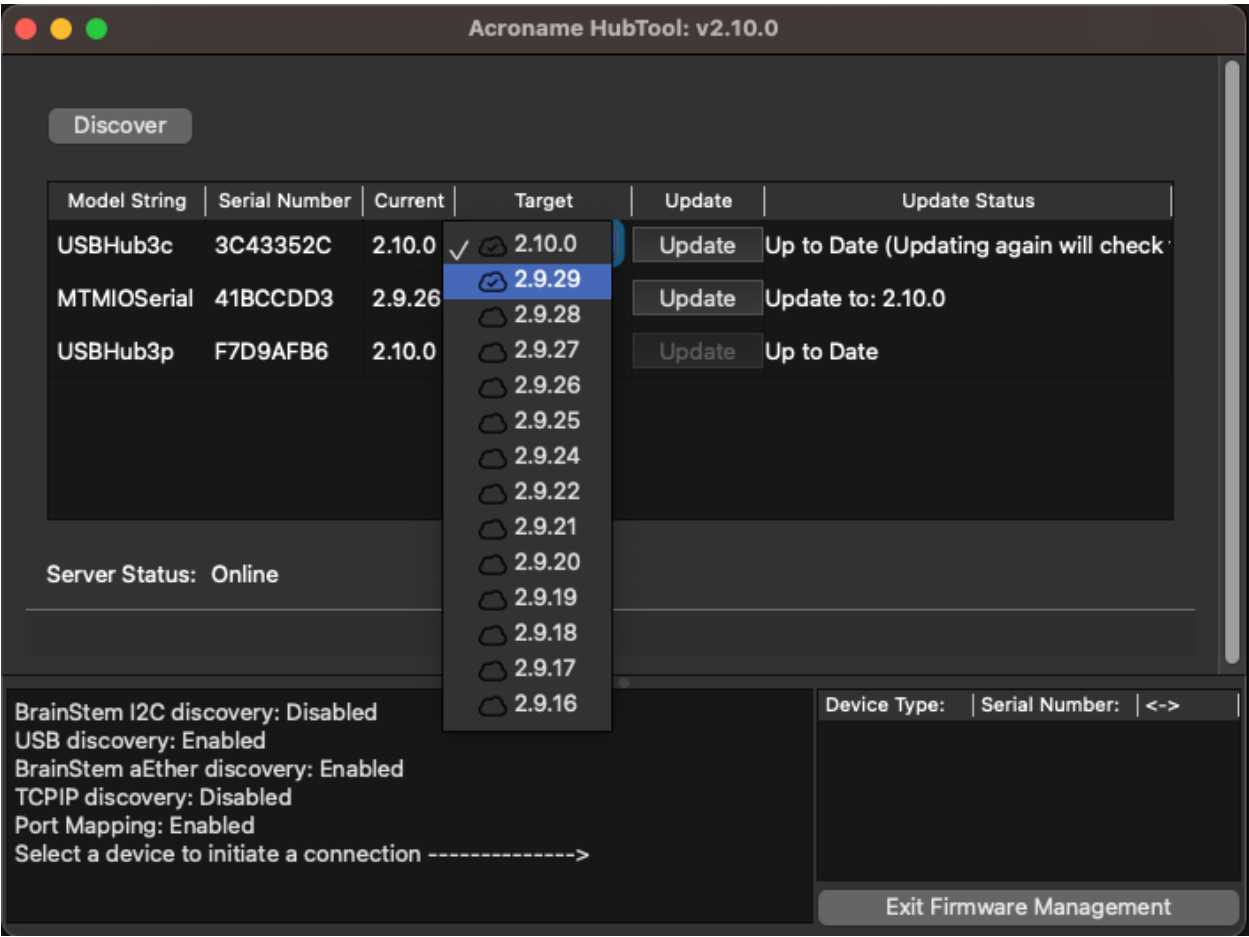
While the device is updating, the process will be shown in the right-most column.

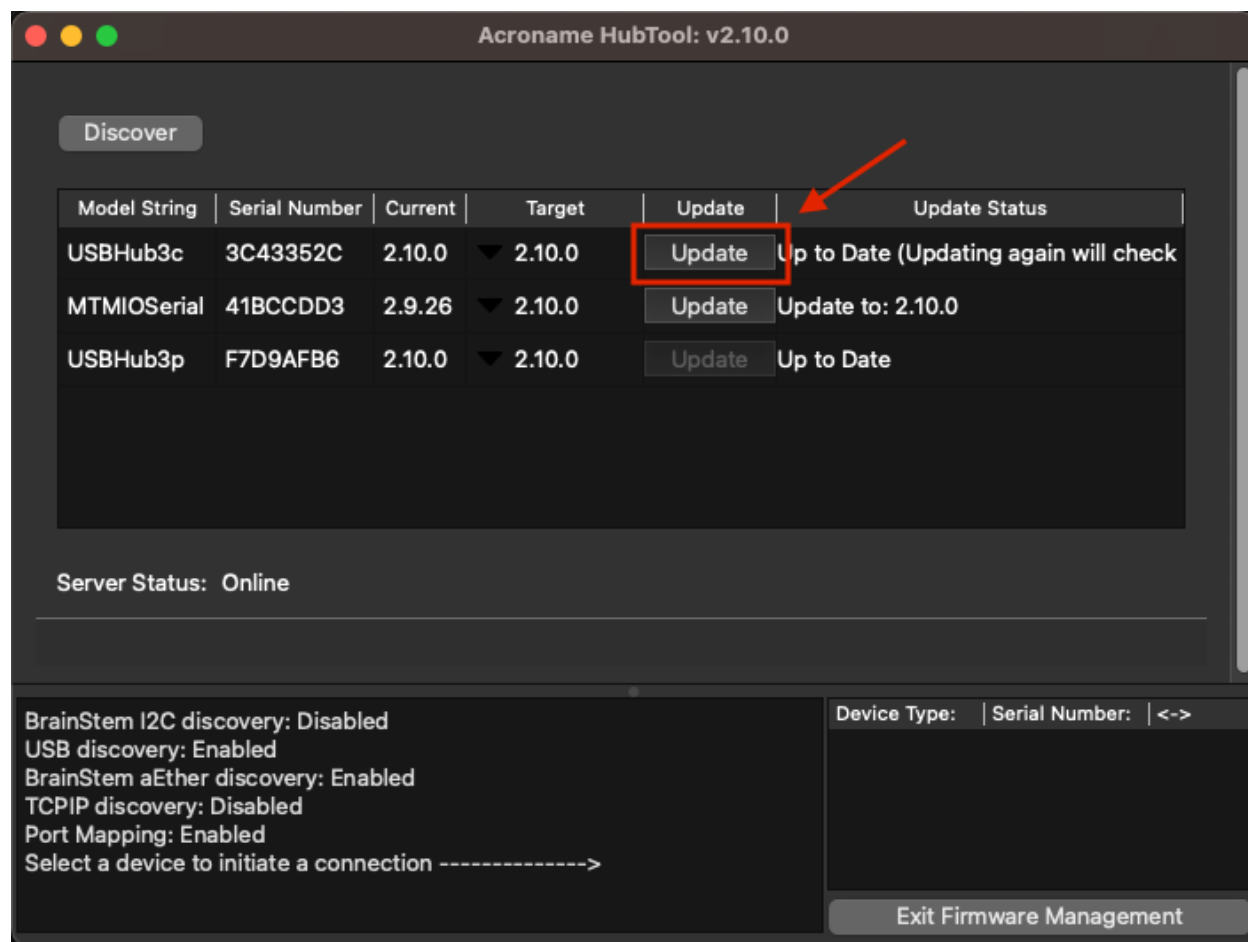
Note: Do not quit HubTool during the update. This may cause the device firmware to become corrupted, requiring a separate firmware recovery.

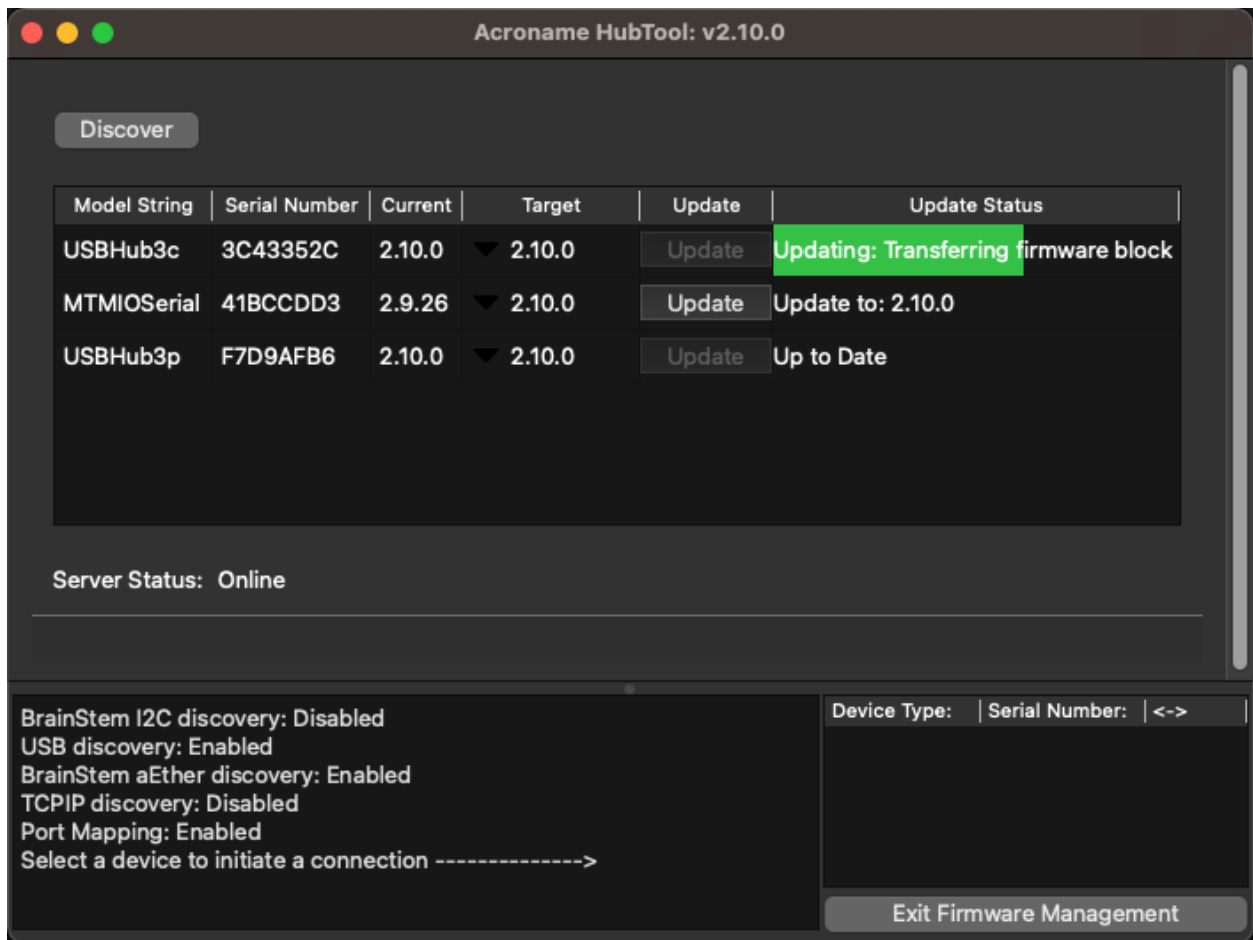
If the update was successful, the device that was updated will show the current version and "Update Success" in the right-most column.

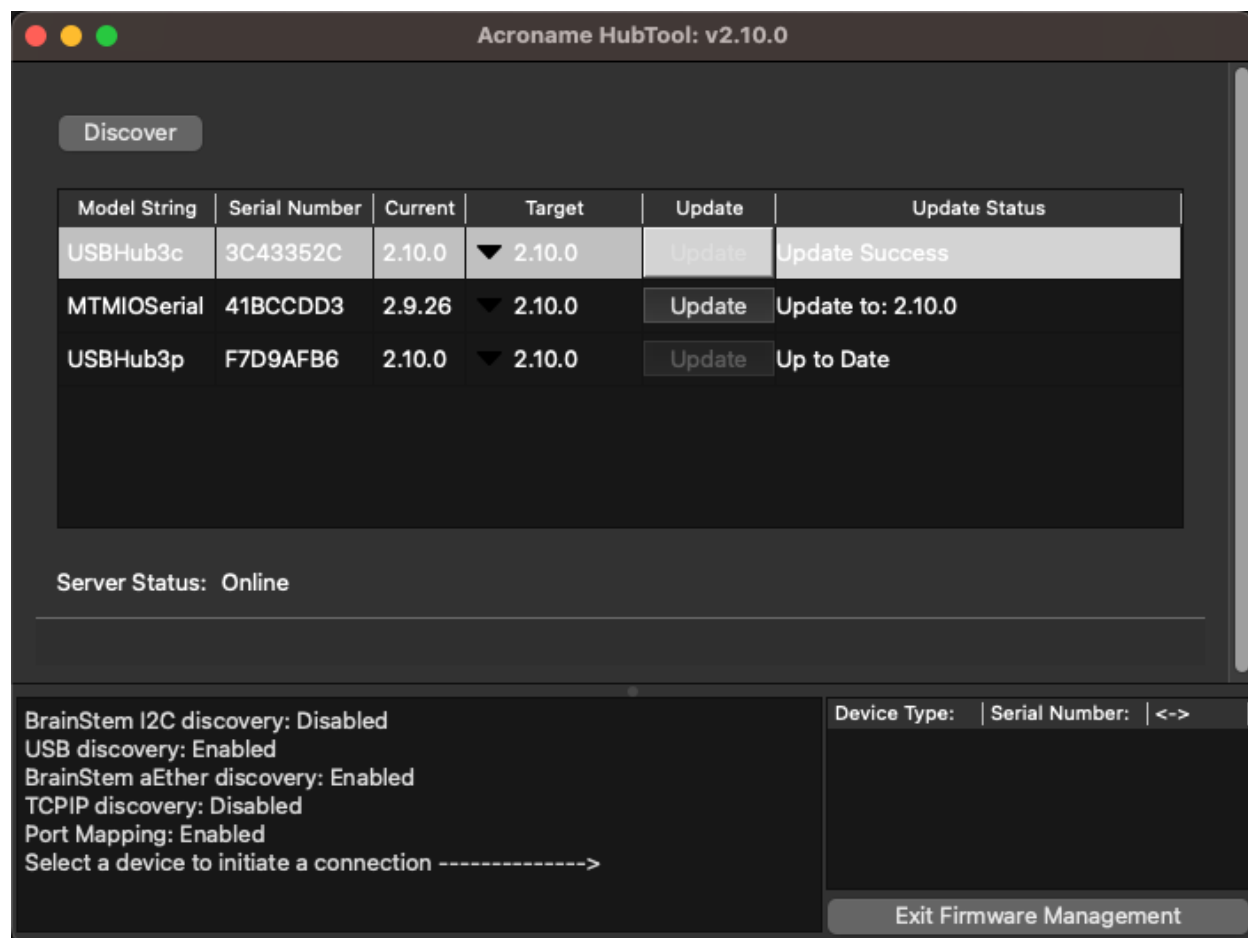
Click "Exit Firmware Management" in the lower-right corner to return to the main HubTool application.











4.4.2 Updating Firmware via CLI

BrainStem firmware can be modified using the Updater CLI utility. This method is often very useful in remote installations and large scale automation systems as it is easily invoked with a simple script. Using your computer's operating systems terminal interface, navigate to the "cli" folder located in the Brainstem Dev Kit download. Downloads for all of Acroname's products can be found at [Software page](#) on Acroname's website.

Parameters and commands available for the Updater may be found by running the utility without any commands or arguments (When running Updater on a Mac or Linux you will need to use ".". Windows does not require this).

Note: The following examples will be preformed from a Mac.

```
$> ./Updater
```

If you would like additional help information, including examples you can use the -H command.

```
$> ./Updater -H
```

```
Updater (Version: 2.10.1 Sep 28 2023)
(Copyright (C) 1994-2023, Acroname Inc.)

Application Usage:
Updater <-B | -D | -G | -U > [-Hh | --help] [-b build_num]
    [-d serial_num]
    [-t transport] [-r router_serial]
    [-s serial_port] [-o baud_rate]

BrainStem Update and Maintenance Application
-H -h or --help for full usage information.

Parameters to commands:
    -b <build number>          - Build version of the firmware to transfer.
                                [Default: latest version available].
    -d <device>                - Device serial number of the device to access.
    -t <transport type>        - communications method [USB,TCP,RECOVER].
    -r <router serial>         - indirect connection through router device
    -s <serial port>           - name of serial port to use for recovery
    -o <baud rate>             - baud rate for serial port (2400, 4800, 9600,
                                19200, 38400, 57600, 115200, 230400)

Commands:
    -B                          - list the Builds available for given device.
    -D                          - Discover devices that are currently connected.
    -G                          - Get build file from the Acroname server and
                                store on local machine for download to device.
                                Must specify a specific device ["-d" option].
                                Default is to obtain the latest build for the
                                given device, override with "-b" option.
    -U                          - Update the firmware on device specified.
                                Must specify a specific device ["-d" option].
                                Default is to download the latest build for the
                                given device, override with "-b" option or
                                specify specific BIRD file using "-f" option.
    -H or -h                    - Display extended usage information.
```

(continues on next page)

(continued from previous page)

Sample usage:	
-D	Discover all devices connected by either USB or TCPIP. Records the device information into the settings for that device.
-D -t USB	Discover all the devices connected by USB. Records the device information into the settings for that device.
-D -t USB -d 0F5849A ↳number 0x40F5849A)	Discover settings of the specific device (serial number 0x40F5849A)
-G -d 40F5849A ↳device directory.	Get the latest firmware for the given device. Downloads the firmware file into the updater specific device directory.
-U -d 40F5849A ↳the device.	Update the firmware on the specific device. Without other options, this command will try to locate the latest firmware version to be used for updating.
-G -d 40F5849A -b 99528558 ↳device.	Get the specified build's firmware for the given device. Downloads the firmware file into the updater's specific device directory with the build number as the filename.
-G -U -d 40F5849A	Get the latest firmware for the given device. Downloads the firmware file into the updater specific device directory. After download, update the firmware using the latest release.
-U -t RECOVER -s /dev/serial ↳serial interface ↳firmware.	Update the firmware as described above, but uses the serial interface in communicating with the device to install the latest firmware.
Sample usage for network discovery and updates:	
-D -r D272031D ↳network	Discover all devices connected to the I2C BrainStem of the given routing device D272031D.
-G -U -d 2181F0EE -r D272031D ↳2181F0EE	Get and Update to the latest firmware for device indirectly through routing device D272031D.

As you can see there are a lot of options for customizing the Updater utility to meet your needs; however, there are just a few basic command that will fit the needs of most users.

Next we will look at a few examples of how to use the updater.

Example: Updating to the Latest Firmware

In this example we will go through the steps required to update our BrainStem module. We will be using a 40pin USBStem module throughout this excersize; however, the other modules work in a similar way.

Make sure your device is connected and has power. Refer to the [Getting Started](#) page for additional information.

Checking for connected devices:

This command will check for USB devices connected to your machine.

```
$> ./Updater -D
```

Looking at the output below you can see that two devices were discovered, one USBHub3+ and one USB-Hub3c. In this example we will be using the USBHub3c information. This information will be handy in the next step as we will need the serial number of our device in order to update it.

```
Updater (Version: 2.10.1 Sep 28 2023)
Searching for BrainStem devices
Discovered Devices: (USB)
  Device      Module  Router  Model      Firmware Version (Build)
  3C43352C    06       06      USBHub3c    2.10.2 (1234567890)
  F7D9AFB6    06       06      USBHub3p    2.10.2 (4101717688)
```

Getting the latest firmware from Acroname's servers:

Using the serial number for the relevant device above, we will construct the following command. The "-G" will pull the most recent firmware from Acroname's server for the given device serial number ("-d").

```
$> ./Updater -G -d 0x3C43352C
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Retrieving firmware for device with serial number 3C43352C
No build specified
Using Latest build
Retrieving build 1256480617
Validating build 1256480617 with server
Build 1256480617 is valid
```

Loading the latest firmware:

Now that we have successfully pulled the most up to date firmware we now need to apply it to the device. The following code will apply the most up to date firmware to the given device.

```
$> ./Updater -U -d 0x3C43352C
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Updating from Birdfile: /Users/cgoss/.acroname/updater/3C43352C/1256480617.bird
Transferring loader to device
Transferring loader block 1 of 3, 65540 bytes
Transferring loader block 2 of 3, 50068 bytes
```

(continues on next page)

(continued from previous page)

```

Transferring loader block 3 of 3, 12288 bytes
Transferred 3 blocks, 127896 total bytes
Starting Load on Device
Transferring firmware block 1 of 20, 65540 bytes
Transferring firmware block 2 of 20, 65540 bytes
Transferring firmware block 3 of 20, 65540 bytes
Transferring firmware block 4 of 20, 65540 bytes
Transferring firmware block 5 of 20, 65540 bytes
Transferring firmware block 6 of 20, 65540 bytes
Transferring firmware block 7 of 20, 65540 bytes
Transferring firmware block 8 of 20, 49268 bytes
Transferring firmware block 9 of 20, 57348 bytes
Transferring firmware block 10 of 20, 65540 bytes
Transferring firmware block 11 of 20, 65540 bytes
Transferring firmware block 12 of 20, 65540 bytes
Transferring firmware block 13 of 20, 65540 bytes
Transferring firmware block 14 of 20, 65540 bytes
Transferring firmware block 15 of 20, 65540 bytes
Transferring firmware block 16 of 20, 49268 bytes
Transferring firmware block 17 of 20, 3924 bytes
Transferring firmware block 18 of 20, 3924 bytes
Transferring firmware block 19 of 20, 4104 bytes
Transferring firmware block 20 of 20, 8 bytes
Transferred 20 blocks, 1019864 total bytes
Update time: 57.5460 sec

```

At this point the firmware has been downloaded to the device and the device has reset and is running with the new firmware.

Example: Reverting to a Previous Version

In our next example we are going to assume that we have just updated to the 2.1.5 firmware; however, for some reason we are not satisfied with how the device is behaving and we want to return back to 2.1.4. With the Updater utility this is easy to do.

Before we begin let's make sure your device is connected and has power. Refer to the [Getting Started](#) page for additional information.

Locating the previous firmware:

Using the '-B' command we can request the available builds for a given device. Since firmware is specific to each device we must also supply the device's serial number with the '-d' parameter.

```
$> ./Updater -B -d 0x3C43352C
```

Below you will see what was printed on my machine¹²⁸. All we need to do is take note of the specific build number that is associated with the version we would like to revert to.

```

Updater (Version: 2.10.1 Sep 28 2023)
Build List for device [3C43352C]:

```

(continues on next page)

¹²⁸ This output was spliced in to reflect the changes in Updaters output. Therefore, the build numbers will not align with the rest of the example.

(continued from previous page)

Build	Version
910961995	2.10.0
3592403412	2.10.1
1256480617	2.10.2
287855797	2.9.16
273238664	2.9.17
3802420316	2.9.18
804960244	2.9.19
3723629684	2.9.20
526817175	2.9.21
3401723282	2.9.22
619738343	2.9.24
2286324745	2.9.25
3108222154	2.9.26
2422939150	2.9.27
227123222	2.9.28
2425394894	2.9.29

Applying a specific build to a device:

As previously explained we are hypothetically having issues with our device after updating and we want to revert to a previous version. Now that we have located the previous build number lets apply it by adding the “-b” parameter to the last command in the example before.

```
$> ./Updater -G -U -d 0x3C43352C -b 227123222
```

You should see something similar to the below output.

```
Updater (Version: 2.10.1 Sep 28 2023)
Retrieving firmware for device with serial number 3C43352C
Retrieving firmware build 227123222
Retrieving build 227123222
Validating build 227123222 with server
Build 227123222 is valid
Updating firmware with build: 227123222
Transferring loader to device
Transferring loader block 1 of 3, 65540 bytes
Transferring loader block 2 of 3, 50068 bytes
Transferring loader block 3 of 3, 12288 bytes
Transferred 3 blocks, 127896 total bytes
Starting Load on Device
Transferring firmware block 1 of 20, 65540 bytes
Transferring firmware block 2 of 20, 65540 bytes
Transferring firmware block 3 of 20, 65540 bytes
Transferring firmware block 4 of 20, 65540 bytes
Transferring firmware block 5 of 20, 65540 bytes
Transferring firmware block 6 of 20, 65540 bytes
Transferring firmware block 7 of 20, 65540 bytes
Transferring firmware block 8 of 20, 49268 bytes
Transferring firmware block 9 of 20, 57348 bytes
Transferring firmware block 10 of 20, 65540 bytes
Transferring firmware block 11 of 20, 65540 bytes
Transferring firmware block 12 of 20, 65540 bytes
Transferring firmware block 13 of 20, 65540 bytes
```

(continues on next page)

(continued from previous page)

```
Transferring firmware block 14 of 20, 65540 bytes
Transferring firmware block 15 of 20, 65540 bytes
Transferring firmware block 16 of 20, 49268 bytes
Transferring firmware block 17 of 20, 3924 bytes
Transferring firmware block 18 of 20, 3924 bytes
Transferring firmware block 19 of 20, 4104 bytes
Transferring firmware block 20 of 20, 8 bytes
Transferred 20 blocks, 1019864 total bytes
Update time: 57.5460 sec
```

Confirm that you have successfully restored the old firmware:

Just to be safe let's confirm that we have successfully restored the old firmware. This can be done by issuing the discover command.

```
$> ./Updater -D
```

As you can see the device is now running the 2.9.28 firmware.

```
Updater (Version: 2.10.1 Sep 28 2023)
Searching for BrainStem devices
Discovered Devices: (USB)
  Device   Module  Router  Model      Firmware Version (Build)
  3C43352C  06       06      USBHub3c   2.9.28 (227123222)
  F7D9AFB6  06       06      USBHub3p   2.10.2 (4101717688)
```

4.4.3 Updating Software Licenses

Software licenses (also known as software features or entitlements) control which features and capabilities are available on your BrainStem device. These licenses are embedded within the firmware image and are automatically loaded when you update your device's firmware through either HubTool or the Updater CLI.

Note: Software licenses are tied to your device's serial number and are automatically included in firmware updates when available from Acroname's servers. Please allow 1-2 business days for new license purchases to be processed and activated in our system. If you have any questions or concerns, please contact Acroname support.

How Software Licenses Work

When you purchase additional features or capabilities for your BrainStem device, the corresponding software licenses are associated with your device's serial number in Acroname's licensing system. These licenses are then automatically embedded into the firmware image that is served to your device during firmware updates.

The license update process works as follows:

1. **License Assignment:** Licenses are assigned to your device's serial number in Acroname's server system after purchase and processing
2. **Firmware Generation:** When you request a firmware update, the server generates a custom firmware image that includes your device's specific licenses

3. **Firmware Update:** The licensed firmware is downloaded and installed on your device
4. **Feature Activation:** Once the firmware update completes, the new features become available on your device

Updating Licenses via HubTool

The easiest way to update your software licenses is through HubTool's Firmware Management interface. When you update your firmware through HubTool, any new licenses associated with your device will be automatically included.

For detailed instructions on using HubTool for firmware updates, see [HubTool](#).

Updating Licenses via Updater CLI

For automated systems or remote installations, you can update software licenses using the Updater CLI utility. The CLI approach works the same way as HubTool - licenses are automatically included when you update to the latest firmware.

For detailed instructions on using the Updater CLI for firmware updates, see [CLI](#).

Verifying License Updates

After updating your firmware (and licenses), you can verify that the new features are available by:

1. **Checking firmware version:** Use the discover command to confirm the new firmware version is installed
2. **Testing licensed features:** Attempt to use the newly licensed features through your application or HubTool

Verify via CLI:

```
$> ./Updater -D
```

This will show your device's current firmware version and confirm the update was successful.

Troubleshooting License Updates

If you're not seeing your expected licensed features after a firmware update:

1. **Verify license assignment:** Contact Acroname support to confirm your licenses are properly assigned to your device's serial number
2. **Check firmware version:** Ensure you're running the latest firmware version
3. **Network connectivity:** Ensure your device has internet connectivity during firmware updates to download the licensed firmware image

Note: If you continue to experience issues with licensed features, contact Acroname support with your device's serial number for assistance.

Important Notes

- **Serial Number Dependency:** Licenses are tied to your device's serial number.
- **Automatic Inclusion:** Licenses are automatically included in all firmware updates no separate license installation step is required.
- **Server-Side Processing:** License embedding happens on Acroname's servers when you request firmware updates.
- **Firmware Version:** Always ensure you're running the latest firmware version to have access to all your licensed features.
- **Old Firmware:** Licenses will also be loaded into old firmware versions but some features may not be available.

4.4.4 Updating Firmware without an Internet Connection

In order for the Updater to retrieve a firmware image from the Acroname servers, a connection to the public internet is required. However, in certain circumstances, a BrainStem device may be used on a system which is unable to access the external internet. In this case, some additional steps are necessary to retrieve the required firmware image for a product.

Using your computer's operating systems terminal interface, navigate to the "cli" folder located in the Brainstem Dev Kit download. Downloads for all of Acroname's products can be found at [Software page](#) on Acroname's website.

Make sure your device is connected and has power. Refer to the [Getting Started](#) page for additional information.

Retrieve the Serial Number for the device to be updated

This command will check for USB devices connected to your machine:

```
$> ./Updater -D
```

Looking at the output below you can see that two devices were discovered, one USBHub3+ and one USB-Hub3c. In this example we will be using the USBHub3c information. This information will be handy in the next step as we will need the serial number of our device in order to update it.

```
Updater (Version: 2.10.1 Sep 28 2023)
Searching for BrainStem devices
Discovered Devices: (USB)
  Device      Module  Router  Model      Firmware Version (Build)
  3C43352C    06      06      USBHub3c    2.10.2 (1234567890)
  F7D9AFB6    06      06      USBHub3p    2.10.2 (4101717688)
```


Getting the latest firmware from Acroname's servers

Using the serial number for the relevant device above, we will construct the following command. The “-G” will pull the most recent firmware from Acroname’s server for the given device serial number (“-d”).

```
$> ./Updater -G -d 0x3C43352C
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Retrieving firmware for device with serial number 3C43352C
No build specified
Using Latest build
Retrieving build 1256480617
Validating build 1256480617 with server
Build 1256480617 is valid
```

The firmware files will be stored in a hidden directory in the user’s home directory:

1. Windows: \Users\\.acroname\updater\\
2. Mac/Linux: ~/.acroname/updater/<serial number>/

The directory for each serial number will have at least one <build number>.bird file; these files are the firmware image to be flashed onto the device. The build number in the filename corresponds to the build number in the output of the Updater command.

These files can be copied to the offline computer that is attached to the device that is to be updated.

Note: The downloaded firmware files are specific to one serial number; a .bird file that was downloaded for one serial number cannot be installed onto a different device.

Loading the firmware files

After the firmware files for the hub to be updated are copied to the offline host computer, the following command will apply the firmware file to the given device.

```
$> ./Updater -U -d 0x3C43352C -f <path to .bird file>
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Updating from Birdfile: /Users/cgoss/.acroname/updater/3C43352C/1256480617.bird
Transferring loader to device
Transferring loader block 1 of 3, 65540 bytes
Transferring loader block 2 of 3, 50068 bytes
Transferring loader block 3 of 3, 12288 bytes
Transferred 3 blocks, 127896 total bytes
Starting Load on Device
Transferring firmware block 1 of 20, 65540 bytes
Transferring firmware block 2 of 20, 65540 bytes
Transferring firmware block 3 of 20, 65540 bytes
Transferring firmware block 4 of 20, 65540 bytes
Transferring firmware block 5 of 20, 65540 bytes
Transferring firmware block 6 of 20, 65540 bytes
Transferring firmware block 7 of 20, 65540 bytes
Transferring firmware block 8 of 20, 49268 bytes
Transferring firmware block 9 of 20, 57348 bytes
```

(continues on next page)

(continued from previous page)

```

Transferring firmware block 10 of 20, 65540 bytes
Transferring firmware block 11 of 20, 65540 bytes
Transferring firmware block 12 of 20, 65540 bytes
Transferring firmware block 13 of 20, 65540 bytes
Transferring firmware block 14 of 20, 65540 bytes
Transferring firmware block 15 of 20, 65540 bytes
Transferring firmware block 16 of 20, 49268 bytes
Transferring firmware block 17 of 20, 3924 bytes
Transferring firmware block 18 of 20, 3924 bytes
Transferring firmware block 19 of 20, 4104 bytes
Transferring firmware block 20 of 20, 8 bytes
Transferred 20 blocks, 1019864 total bytes
Update time: 57.5460 sec

```

At this point the firmware has been downloaded to the device and the device has reset and is running with the new firmware.

4.4.5 Updating a Brainstem Module via the Brainstem Network.

The new updater utility also has the ability to update devices via the *BrainStem network*. This can be very handy when you have multiple BrainStem products connected to a single host computer.

Transport vs Brainstem Network

Before digging in it is important to know the difference between a transport and the Brainstem network.

“Transport”

When we refer to transport we are speaking of the interface between devices and more specifically we are referring to the hardware. Acroname currently offers three transports mediums; TCPIP, USB and I2C. While all are capable of trafficking the Brainstem network only TCPIP and USB are directly available to the user. Although I2C is technically a transport when it comes to the Brainstem network it is handled internally. This is not to be confused with the *I2C Entity* which allows communication to third party devices.

“Brainstem Network”

Keeping in mind that the transport is at the hardware level the Brainstem network is at the software level. It is what handles all communication between Brainstem devices. One thing that makes the Brainstem network particularly interesting and useful is the fact that it can transition between transports. Additionally, this transition is handled internally. For more information see *BrainStem networking* located in the appendix of our support documentation. There it will explain the details of how it works and how to configure your devices.

In this example we will be using the TCPIP transport to communicate via the Brainstem network from our host machine, through our local network, to an *MTM-EtherStem*¹²⁹ where it will then be converted to the I2C transport and sent to a *MTM-PM1*¹³⁰ module.

Before we begin let's make sure your device is connected and has power. Refer to the *Getting Started* page for additional information.

¹²⁹ <https://acroname.com/store/s67-mtm-etherstem>

¹³⁰ <https://acroname.com/products/ACRONAME-MTM-1-CHANNEL-POWER-MODULE>

Discovery

Since we will be updating a device through another device we will need to know the serial number of the device we will be communicating through. To find the serial number we can simply use the '-D' discover command.

```
$> ./Updater -D
```

```
Updater [Version 0.2 Dec 28 2015 13:54:37] [Copyright (C) 1994-2015, Acroname Inc.]

Discovering Devices [USB]:
  Device      Module  Router  Model                      Firmware Version
Discovering Devices [TCPIP]:
  Device      Module  Router  Model                      Firmware Version  [IP address]
D272031D     04       04      0F [MTMEtherStem]         2.2.0 (0)         [10.128.38.159]
856C1C03     02       02      05 [EtherStem ]          2.1.4 (99528558)  [10.128.38.122]

Completed processing: Updater [Version 0.2 Dec 28 2015 13:54:37] [Copyright (C) 1994-
↳2015,
Acroname Inc.]
```

Brainstem Network Discovery

From the discovery we found a MTM-EtherStem with serial number D272031D. We will need this number in order to preform a Brainstem network discovery. To preform the indirect discovery we will need to use the '-r' parameter. The '-r' is for router and this tells Updater to look for anything at that devices router level and below.

```
$> ./Updater -D -r 0xD272031D
```

```
Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem Release:2.1.5] [Copyright (C)
1994-2015, Acroname Inc.]

Discovering Network Devices from [D272031D] via [TCPIP]:
  Device      Module  Router  Model                      Firmware Version  [IP address]
D272031D     04       04      0F [MTMEtherStem]         2.2.0 (0)         [10.128.38.159]
2181F0EE     06       04      0E [MTMPM ]               2.1.4 (239384838) [10.128.38.159]
CA6A1B05     08       04      0D [MTMIOSerial ]         2.2.0 (0)         [10.128.38.159]

Completed processing: Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem↳
↳Release:2.1.5]
[Copyright (C) 1994-2015, Acroname Inc.]
```

As you can see Updater returned 3 devices, One of which being the router device that we specified in the discovery. In other words Updater has returned all of the devices on the MTM-EtherStem's I2C [BrainStem network](#).

Updating via the Brainstem Network.

Now that we have the serial number of the Brainstem network device we can form our final command to update the device. The command is very similar to the previous one with the exception of swapping out ‘-D’ (discovery) for ‘-GU’ (get and update). Additionally, we will need to add the ‘-d’ (device) parameter so that it knows which device to update.

```
$> ./Updater -G -U -r 0xD272031D -d 0x2181F0EE
```

```
Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem Release:2.1.5] [Copyright
(C) 1994-2015, Acroname Inc.]

Latest firmware for [2181F0EE][aMTMPM1] is build [264993366].
Getting firmware build [264993366] for [2181F0EE][aMTMPM1].
Downloaded firmware into local file [/Users/Mitch/.acroname/updater/2181F0EE/264993366
.bird] VERIFIED.
Network Update of device [2181F0EE] via [D272031D][00]
Discovering Module # of Network Device [2181F0EE] thru [D272031D]
Using Module #[06] for Network Device
Update using firmware file [/Users/Mitch/.acroname/updater/2181F0EE/264993366.bird].
Transferring loader to device
Transferring loader block 0, 8800 bytes
Transferred 1 blocks, 8800 total bytes
Transferring firmware to device [/Users/Mitch/.acroname/updater/2181F0EE/264993366.
.bird]
Transferring firmware block 0, 772 bytes

(4 of 772 bytes (0%) of firmware block transferred.
(772 of 772 bytes (100%) of firmware block transferred.
Transferring firmware block 1, 33528 bytes

(4 of 33528 bytes (0%) of firmware block transferred.
(1028 of 33528 bytes (3%) of firmware block transferred.
(2052 of 33528 bytes (6%) of firmware block transferred.
(3076 of 33528 bytes (9%) of firmware block transferred.
(4100 of 33528 bytes (12%) of firmware block transferred.
(5124 of 33528 bytes (15%) of firmware block transferred.
(6148 of 33528 bytes (18%) of firmware block transferred.
(7172 of 33528 bytes (21%) of firmware block transferred.
(8196 of 33528 bytes (24%) of firmware block transferred.
(9220 of 33528 bytes (27%) of firmware block transferred.
(10244 of 33528 bytes (30%) of firmware block transferred.
(11268 of 33528 bytes (33%) of firmware block transferred.
(12292 of 33528 bytes (36%) of firmware block transferred.
(13316 of 33528 bytes (39%) of firmware block transferred.
(14340 of 33528 bytes (42%) of firmware block transferred.
(15364 of 33528 bytes (45%) of firmware block transferred.
(16388 of 33528 bytes (48%) of firmware block transferred.
(17412 of 33528 bytes (51%) of firmware block transferred.
(18436 of 33528 bytes (54%) of firmware block transferred.
(19460 of 33528 bytes (58%) of firmware block transferred.
(20484 of 33528 bytes (61%) of firmware block transferred.
(21508 of 33528 bytes (64%) of firmware block transferred.
(22532 of 33528 bytes (67%) of firmware block transferred.
(23556 of 33528 bytes (70%) of firmware block transferred.
(24580 of 33528 bytes (73%) of firmware block transferred.
(25604 of 33528 bytes (76%) of firmware block transferred.
```

(continues on next page)

(continued from previous page)

```

(26628 of 33528 bytes (79%) of firmware block transferred.
(27652 of 33528 bytes (82%) of firmware block transferred.
(28676 of 33528 bytes (85%) of firmware block transferred.
(29700 of 33528 bytes (88%) of firmware block transferred.
(30724 of 33528 bytes (91%) of firmware block transferred.
(31748 of 33528 bytes (94%) of firmware block transferred.
(32772 of 33528 bytes (97%) of firmware block transferred.
(33528 of 33528 bytes (100%) of firmware block transferred.
Transferred 2 blocks, 34300 total bytes
Resetting router number of device:[04:2181f0ee] with module# of router[D272031D]
Device [2181F0EE] may need a physical reset before router number can be reset.
Completed updating firmware on device [2181F0EE]
Sending Reset to device [2181F0EE]

Completed processing: Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem
->Release:2.1.5]
[Copyright (C) 1994-2015, Acroname Inc.]

```

Please note that the Updater utility may appear to hang after it has transferred the firmware. This is because the device is waiting for the device to reset so that it can try and return all the previous BrainStem network settings (ie router and module addresses) for a seamless update process; however, with some updates a hard reset will be required. Simply press the reset button on the development board or power cycle the device.

4.4.6 Recovering BrainStem Firmware via CLI

In rare circumstances, a BrainStem device may become unresponsive after a failed firmware update (aka: “Bricked”). Several Acroname devices have the capability to be recovered in the field, using the `Updater` CLI utility.

Using your computer’s terminal interface, navigate to the “cli” folder within the Brainstem Software Development Kit. This directory contains the *Updater CLI utility* that will be used for recovery. Downloads for all of Acroname’s products can be found at [Software page](#) on Acroname’s website.

USBHub3c

In order to recover a USBHub3c, you will need the following equipment on hand:

1. USB-C to USB-C cable or USB-C to USB-A cable
2. A pen or other sharp object to press the recessed Reset button.

Execute the following procedure to recover to a working firmware image:

1. Unplug everything from the USBHub3c, including all the USB-C ports and the DC power input.
2. On the bottom of the hub, press and hold the recessed Reset button.
3. While holding the reset button, insert the USB-C cable into the Control port.
4. After 5 seconds, release the Reset button.
5. Run the following updater command: `./Updater -U -t RECOVER -s USB`
6. After all the lights turn off, unplug everything from the hub.

Note: If Updater cannot find the device, try flipping the USB-C connector on the Control port and repeat the process over again.

If you are still experiencing issues, contact [Acroname](https://acroname.com/contact-us)¹³¹ for assistance.

USBHub3p

In order to recover a USBHub3p, you will need the following equipment on hand:

1. DC Power Supply
2. USB-B cable
3. A pen or other sharp object to press the recessed Reset button.

Execute the following procedure to recover to a working firmware image:

1. Unplug everything from the USBHub3p, including all the USB-C ports and the DC power input.
2. Connect the DC Power Supply to the hub
3. Connect the USB-B cable to the “Up0” port.
4. On the side of the hub, press and hold the recessed Reset button for at least 5 seconds, and then release.
5. *Identify the recovery Serial Port for the hub*
6. Run the following updater command:
 1. Windows: `./Updater -U -t RECOVER -s COM#` where # is the number of the COM port
 2. Mac/Linux: `./Updater -U -t RECOVER -s /dev/tty<NAME>` where <NAME> is the full TTY file path.

If you are still experiencing issues, contact [Acroname](https://acroname.com/contact-us)¹³² for assistance.

USBCSwitchPro

In order to recover a USBCSwitchPro, you will need the following equipment on hand:

1. USB-C to USB-C cable or USB-C to USB-A cable
2. A pen or other sharp object to press the recessed Reset button.

Execute the following procedure to recover to a working firmware image:

1. Unplug everything from the USBCSwitchPro, including all the USB-C ports and the DC power input.
2. On the bottom of the hub, press and hold the recessed Reset button.
3. While holding the reset button, insert the USB-C cable into the Control port.
4. After 5 seconds, release the Reset button.
5. Run the following updater command: `./Updater -U -t RECOVER -s USB`
6. After all the lights turn off, unplug everything from the hub.

If you are still experiencing issues, contact [Acroname](https://acroname.com/contact-us)¹³³ for assistance.

¹³¹ <https://acroname.com/contact-us>

¹³² <https://acroname.com/contact-us>

¹³³ <https://acroname.com/contact-us>

USBCSwitch

Note: The USBCSwitch is unable to be recovered in the field. Contact [Acroname](#)¹³⁴ for more information.

USBHub2x4

Note: The USBHub2x4 is unable to be recovered in the field. Contact [Acroname](#)¹³⁵ for more information.

4.4.7 Recovering MTM Module Firmware via CLI

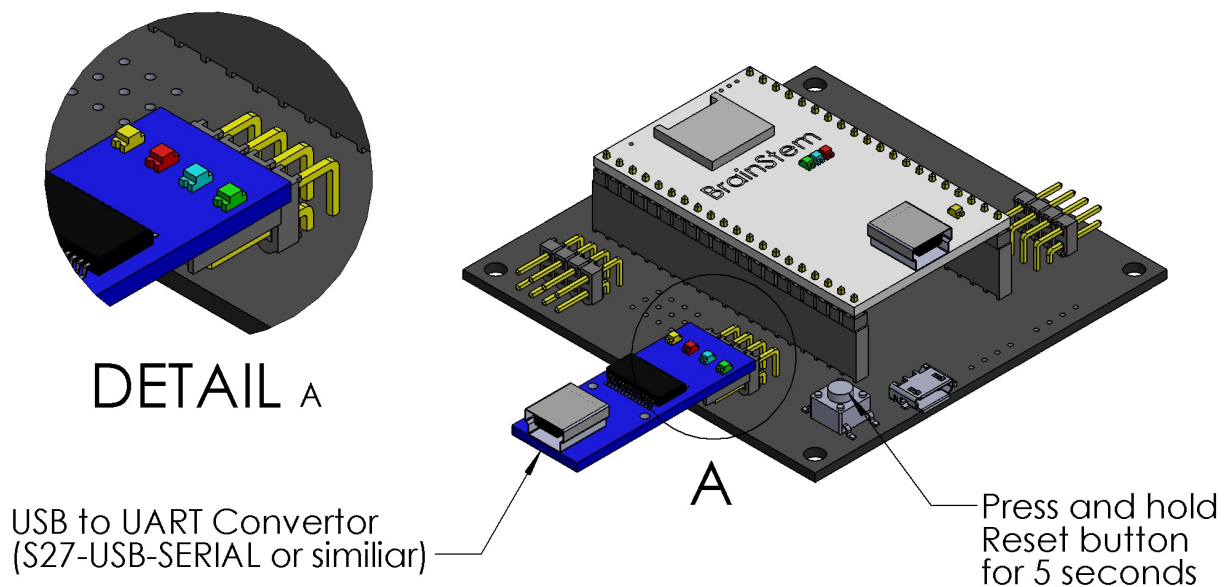
Lets take a look at how we can recover a BrainStem module after it has become unresponsive (aka: “Bricked”). This is also very helpful when dealing with old devices with a firmware version that might not be compatible with the new Updater utility.

In order to preform this recovery process you will need a [USB to Serial Module](#)¹³⁶. There are many other devices that will also work; however, this one is equipped with a connector that easily connects to the UART port on our [breakout boards](#)¹³⁷.

Before starting make sure your device is connected and has power. Refer to the [Getting Started](#) page for additional information.

Preparing Device for Recovery

Using the image below for reference make the UART connection as show. Additionally, you will need to press and hold the reset button for 5 seconds; This will prepare the device to be programmed via the UART port.



¹³⁴ <https://acroname.com/contact-us>

¹³⁵ <https://acroname.com/contact-us>

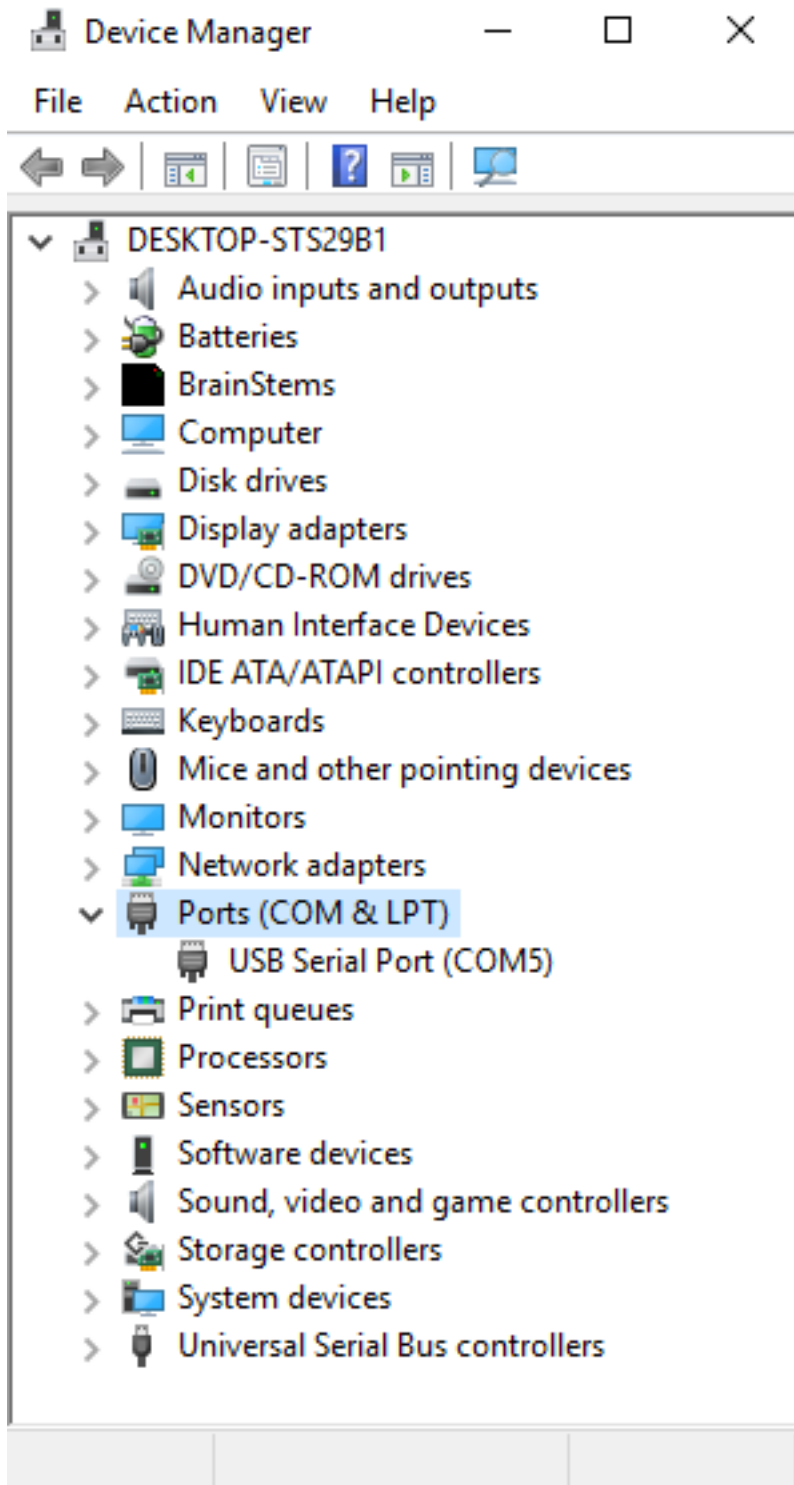
¹³⁶ <https://acroname.com/store/serial-converter-s27-usb-serial>

¹³⁷ <https://acroname.com/site-search/dev>

Finding your Communications Port

Windows

The communications port (COM) on Windows can be found by navigating to the Device Manager and expanding the “Ports (COM & LPT)” section. If you do not immediately recognize your device you can open each and inspect the details or you can simply disconnect and reconnect the device and monitor which one disappears and then reappears (You may need to select Action > Scan for hardware changes between the disconnect and reconnect).



Once you have figured out which device is yours make note of the port number. In my case it would be “COM5”

Mac/Linux

Open terminal and type in the following command (root access is required).

```
$> ls /dev/tty.*
```

This command will return all the serial devices connected to your machine. Locate the one you are wanting to work with. If you are not sure which serial device to use you can run the command twice. Once with the device connected and once without. The one that changes is the device you are interested in.

```
/dev/tty.Bluetooth-Incoming-Port  
/dev/tty.Bluetooth-Modem  
/dev/tty.usbserial-A601R8QJ
```

On my machine the device I am interested in is: “/dev/tty.usbserial-A601R8QJ”. Make note of your device as it will be needed later.

Recovering your Device with Updater

Finally, we are ready to recover the device. Below you will see the command required to recover the device. We will be using the communications port we found [above](#) and don't forget to [configure](#) your device for recovery.

If you have forgotten some of the commands please see [Using Updater via CLI](#) where we explained how to use the -H command to find more information about the Updater utility including examples.

Windows

```
$> ./Updater -U -t RECOVER -s COM5
```

Mac/Linux

```
$> ./Updater -U -t RECOVER -s /dev/tty.usbserial-A601R8QJ
```

After the recover process has completed you will need to press the reset button twice to put the device back into normal operating mode. Whether you are using a Mac, Linux or Windows your output should look similar to the following.

```
Home directory:[/Users/Mitch]  
  
Application parameters:[Updater]  
    [Updater]  
    [-U]  
    [-t]  
    [RECOVER]  
    [-s]  
    [/dev/tty.usbserial-A601R8QJ]  
Updater [Version 0.2 Dec  9 2015 14:14:42] [Copyright (C) 1994-2015, Acroname Inc.]  
  
Firmware_Recovery via Serial Interface:  
  
    Sync to Device:
```

(continues on next page)

(continued from previous page)

```

    Get Info from Device:
Device responded with version: 2.4
Device responded with part number: 637615927
Device ID: [ 05005006 AE061446 508B34C6 F5001E43 ]
Device [05005006 AE061446 508B34C6 F5001E43] ==> Serial#:71E3928C, Build=130702287
GetBuild BIRD [/Users/Mitch/.acroname/updater/71E3928C/130702287.bird] VERIFIED.

    Unlock Memory:
Device ID: [ 05005006 AE061446 508B34C6 F5001E43 ]
Transferring firmware to device from [/Users/Mitch/.acroname/updater/71E3928C/
→130702287.bird]
Transferring firmware block 0, 772 bytes
Transferred 768 of 768 bytes (100%)
Transferring firmware block 1, 49668 bytes
Transferred 4096 of 49664 bytes (8%)
Transferred 8192 of 49664 bytes (16%)
Transferred 12288 of 49664 bytes (25%)
Transferred 16384 of 49664 bytes (33%)
Transferred 20480 of 49664 bytes (41%)
Transferred 24576 of 49664 bytes (49%)
Transferred 28672 of 49664 bytes (58%)
Transferred 32768 of 49664 bytes (66%)
Transferred 36864 of 49664 bytes (74%)
Transferred 40960 of 49664 bytes (82%)
Transferred 45056 of 49664 bytes (91%)
Transferred 49152 of 49664 bytes (99%)
Transferred 49664 of 49664 bytes (100%)
Transferred 2 blocks, 50432 total bytes
Rebooting Device

END of Firmware_Recovery via Serial Interface

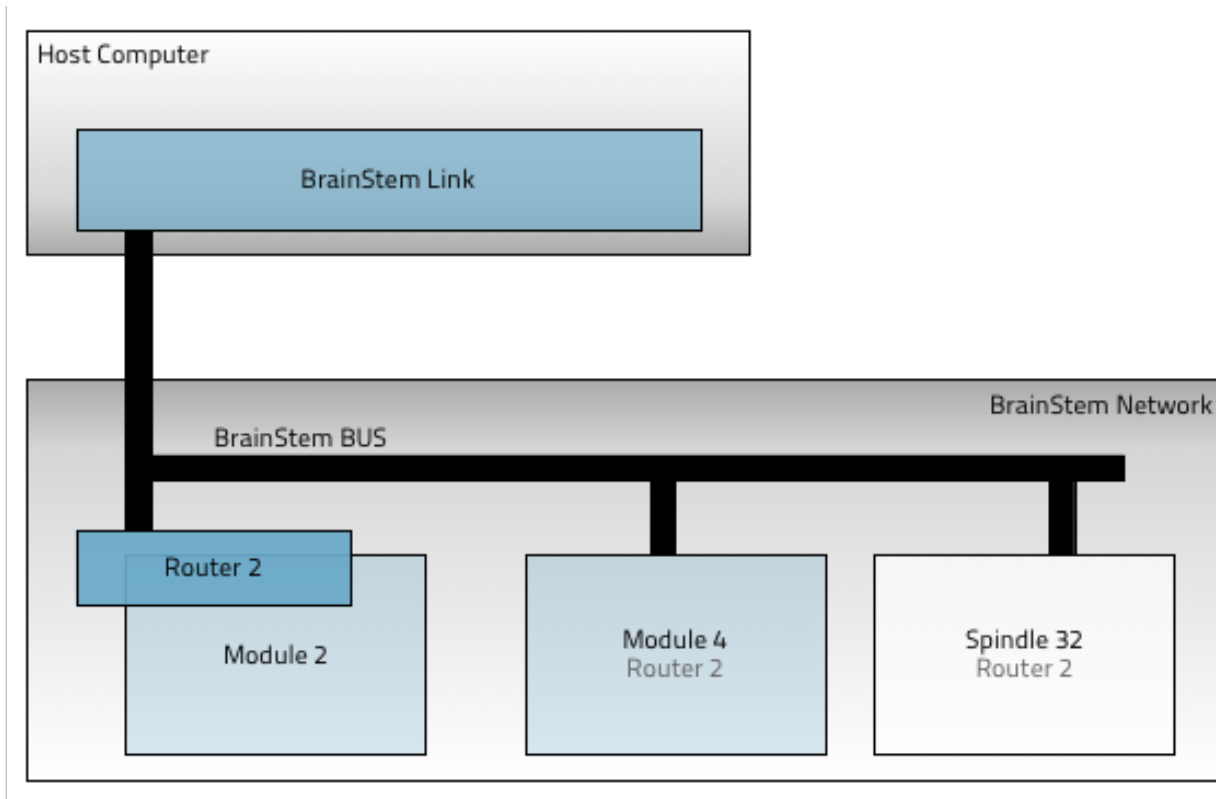
Completed processing: Updater [Version 0.2 Dec 9 2015 14:14:42] [Copyright (C)
1994-2015, Acroname Inc.]

```

4.5 Terminology

4.5.1 BrainStem® Network

The BrainStem is a network of devices that offer rich I/O capabilities and comprehensive interactions. The hardware comes primarily in the form of modules and that can be combined to accomplished I/O specific tasks. These hardware devices all share a common backbone network called the BrainStem Network that uses the I2C bus to exchange and route information around the system. A host or hosts can be employed using a link to then inject information or receive information from this BrainStem network of hardware modules.



4.5.2 BrainStem® Bus

The BrainStem Bus is the network backbone of the BrainStem network. For existing BrainStem modules, the bus uses I2C¹³⁸ as the hardware transport. The brainstem network is a multiple master I2C fast mode plus network. Traffic on this bus generally follows the specification of the BrainStem protocol. Third party devices can be connected to this network, but it is most common to connect I2C peripherals to the BrainStem Module's peripheral I2C ports.

4.5.3 Routing

Each module in the BrainStem Network has a unique I2C address. When a link is employed, it connects through a specific transport to one of the modules in the BrainStem Network. If the host wants to send information to a module or 3rd-party device in the network, it uses the address to send the information over the link. If the linked module is not the recipient, it acts as the router to relay the information from the link's transport to the destination module over the BrainStem Network's I2C bus.

Modules can interact with one another on the BrainStem Network as peers where each can manipulate one-another's I/O. From the I2C parlance, this means the BrainStem Network has multiple masters.

When a module needs to communicate back to the host (if present), the module can send the information to the router and the router will take care of relaying that information back across the link transport to the host.

¹³⁸ <http://i2c.info/i2c-bus-specification>

4.5.4 Module

Modules are the heart of the BrainStem architecture. Each is a self-contained hardware solution that employs the BrainStem OS and can communicate with other modules over the BrainStem Network. Additionally, all modules have a link transport that enables them to talk with a host computer outside the BrainStem Network. Examples of link transports a module may use are TCP/IP, Bluetooth, USB, or other industry standards. Available BrainStem modules are listed in the [BrainStem Products](#)¹³⁹ webpage.

4.5.5 Host

The host is typically a larger computer or compute environment. These are most often desktop, mobile, or embedded processors running MacOS, Windows, or Linux operating systems. The protocols and transports are well documented so there is no practical restriction on what the host is running or how it works, it simply must be able to support the industry standard transport link mechanism. Many tools are provided for the above mentioned operating systems to allow control, configuration, and updating of the BrainStem modules across the host's link to the BrainStem Network.

4.5.6 Entity

An Entity provides a way to interact with a type of Hardware I/O within a BrainStem module. Entities include Digital inputs and outputs, analog inputs and outputs, I²C bus', Serial UARTS, system components, and other specialized hardware classes. Entities are the fundamental building blocks of interaction between BrainStem 'clients' and the hardware that BrainStems interact with. See the reference section on [Entities](#) for more in-depth discussion.

4.5.7 Discovery

The BrainStem API provides a mechanism for discovering the devices that are currently connected to the Host computer. This is part of the [C API](#) on the C/C++ library, and part of the discovery module within the [Python package](#). The Discovery API provides methods to find connection details for a specific module, as well as methods to list all connected modules. The Discovery methods return Spec objects which represent the required connection details for the device, as well as the Device's [model number](#). The list of model numbers is provided on the [C API](#) page and the [Python API](#) page.

4.6 USB Drivers

Acroname has removed the need for kernel drivers for BrainStem devices across all of the platforms that we support. There is no longer a need to install drivers. However, both Linux and Windows 7 have steps that are required to allow BrainStem devices to work properly.

¹³⁹ https://acroname.com/store-grid/field_manufacturer/acroname

4.6.1 Mac OS X

On Mac OSX there are no installation procedures required.

4.6.2 Linux Ubuntu

On Linux systems, users must run a script to properly set ownership and permissions for BrainStem devices. The script is located within the `brainstem_linux_driverless` folder, and is called `udev.sh`.

```
$> cd /path/to/brainstem_linux_driverless
$> ./udev.sh
```

Note: Executing the commands within `udev.sh` requires `sudo` privileges. You will be prompted for your login password when you execute the script. Once you execute the script, you may have to log out and back in.

In addition to setting the `udev` rules, Linux operating systems require system packages to be installed:

- Ubuntu LTS 18.04, 20.04, 22.04**
 - Required dependencies: *apt install xcb**
- Ubuntu 24.04 (and up)**
 - Required dependencies: *apt install qt6-base-dev*
- Red Hat 8**
 - Required dependencies: *dnf install qt5-qtbase-gui*
- Red Hat 9 (and up)**
 - Required dependencies: *dnf install qt6-qtbase-gui*

4.6.3 Windows 7 USB Driverless Installation

On the Windows 7 OS an installation is required to allow BrainStem devices to be recognized by the system. BrainStem devices use the Microsoft provided WinUSB device drivers to communicate with the brainstem. On Windows 7 operating systems the WinUSB driver is not installed automatically. On more modern versions of Windows newer than 7 this process is automatic and BrainStem devices need no install.

There is a `windows_driver_installation.pdf`, within the Drivers folder of the [BrainStem Development Kit \(BDK\)](#)¹⁴⁰, that describes the process for installing the WinUSB Driver on Windows 7 OS.

4.7 Appendix

¹⁴⁰ <https://acroname.com/api>

4.7.1 Appendix I: BrainStem Universal Entity Interface (UEI)

Most of the BrainStem 2.0 functionality is represented by abstract entities. These entities are things like battery voltage, the module address, or an analog voltage. These entities are accessed in a common command interface called a UEI. These UEIs allow various clients to access module entities both locally and over the BrainStem's network. Clients include the Host, and Reflex code running on the module or on another module in the network.

How UEI's Work

UEI's allow the setting and getting of entity information. This information can be in empty, byte, 2-byte, 4-byte, or N-byte data sizes and the UEI's allow a common mechanism for either reading or writing these entity values. UEI Values of 2-byte and 4-byte are stored in big endian format. Some entities are limited to strictly reading (getting) or writing (setting), based on the underlying entity properties. For instance, an A2D input on a BrainStem module can be queried (read) but not written. An empty write is used to trigger an entity on the module, much like a void parameter to a routine in C.

The mechanism for both reads and writes is performed with an exchange of two UEI's. Reads use a GET/VAL pair where the entity value is requested (GET) and a value is sent back as the reply (VAL). Writes use a SET/ACK pair where the write employs a SET UEI and then an optional ACK response can be sent to learn the status of the SET operation.

There is one other mode of operation which is called streaming. When streaming has been enabled, the device will automatically send new values to the requestor by asynchronously generating and sending a VAL payload.

UEI Classes

Each UEI is part of a group or class of UEI's that share a common subsystem within the BrainStem 2.0 architecture. These classes directly correspond to underlying command structures within the protocols on the link and BrainStem network. The classes also logically group common functionality for various entities.

Example UEI Classes

- System - These are system global values like the module's serial number, I2C rate, etc. Not all modules will have all possible system UEI's available, based upon functionality.
- Servo - This class collects all the common functionality around a servo input/output for modules supporting servos. These may include things like enable, reverse, and position UEI's along with others.
- Analog - Both A2D and DAC channels are grouped in this class of UEI's as they often share functionality or are conceptually similar.
- Port - The class used for USB Port manipulation, enabling/disabling data or power.

The GET/VAL UEI Transaction

The UEI GET/VAL transaction is a back-and-forth between the requestor (client) and the BrainStem 2.0 module (server) where the entity being read is located. There are several client types including the host, another module, a virtual machine running on the network's modules, or a third-party device on the BrainStem network. The requestor first identifies the full entity and specifies a GET operation. The requestor also is responsible for identifying where the response (VAL operation) should be sent. Both of these operations are asynchronous commands.

Breaking this down further, lets first consider the GET UEI from the client requestor. This specifies 5 or 6 specific pieces of information:

UEI GET Request

- Command - The command that groups the class of UEI's in which the entity is included (This is part of BrainStem 2.0 command structure).
- Operation - the operation of the UEI which is GET in the case where the possible operations are VAL (0), GET (1), SET (2), or ACK (3)
- Option - The specific option code within the class of UEI's
- Reply - Identifies the requestor so the response returns to that requestor where the possible options include Host (1), I2C (2), or Reflex (3).
- Index - The array index of the class. This allows for multiple groups of classes like ports, servos, motion channels, etc.
- Subindex (optional) - The subindex of a specific index of the class. This allows for there to be sub-components of a class like setting/getting the configuration for a channel of an equalizer class.
- ReplyID (optional) - For reply values that require more information, this information is include such as the Reflex Machine thread identifier or the remote module's address over I2C where the response will be sent. If the reply specifies the host, no additional replyID information is needed so it will not be present.

The above information is packed into a sequence of bytes for transmission to the module from the requestor. These packed bytes overlay the normal BrainStem 2.0 command structure so they are essentially a generalized set of commands for accessing entities.

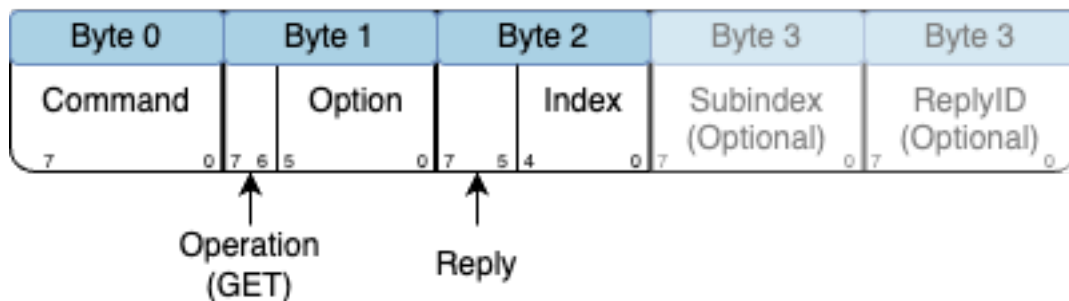


Fig. 1: Packed Byte GET UEI Structure

Once this UEI command payload is received by the module where the entity resides, the bytes are converted to the UEI information which is then validated. Provided all the information checks out and the entity can be read, a response is constructed and returned to the requestor as specified in the GET UEI information. The value may be a single byte, 2-byte, 4-byte, or N-byte value, depending on the entities native size. There are then four possible return packet structures, one for each size.

The information returned in the VAL response is identical to the GET with the following exceptions. First, the responder address identifies the entity where the entity lives which is not necessarily that of the requestor. Second, the operation is VAL. Third, subindex is not transmitted back to the requestor in the VAL. Finally, the data (1-N bytes) follows the index and there is no replyID.

UEI VAL Reply

- Command - The command that groups the class of UEI's in which the entity is included (This is part of BrainStem 2.0 command structure).
- Operation - The operation of the UEI which is VAL in this case where the possible operations are VAL (0), GET (1), SET (2), or ACK (3).
- Option - The specific option code within the class of UEI's
- State - Identifies the response state. If no errors occurred, this value is zero. If an error occurred the high bit (7) of this byte will be set.
- Stream - Identifies if this is a streaming payload or not. If the payload is a normal GET/VAL, this value is zero. If the payload is streamed asynchronously bit (6) of this byte will be set.
- Index - The array index of the class. This allows for multiple groups of classes like ports, servos, motion channels, etc.
- Data - 1, 2, 4, or N bytes for the entity value that was read. If an error occurred, the error bit is set in the state and the data is always a 1-byte error value describing the error.
- Continue - Identifies if there are subsequent payloads after this one that should be used in conjunction or if this is the last payload for the value.
- Sequence - Used to identify the payload index for a multi-payload response.
- Responder Address - The responding entity's module address so the requestor can potentially match the initial request with this response. For replies to the host, this responder address is implicit in the inbound packet protocol so the responder address is not sent in host responses.

UEI VAL Error Handling

In some cases, the UEI GET request may be incorrect, refer to a non-existent entity, or have some type of mode error like reading from a write-only entity. If the request cannot be fulfilled for some such reason, a response is still sent but the response simply contains an error state and *error code*.

The SET/ACK UEI Transaction

Much like GET/VAL transactions which are request/response, the SET/ACK is a request/acknowledge pair of commands. The SET sends a payload of data to write to the entity and the ACK offers acknowledgment and possibly an error code. The ACK is optional so a requestor can "set and forget" if the acknowledgement is not desired. Typically the ACK is used to synchronize behavior on the requesting side to ensure that the value has been written before proceeding. When this synchronizing is not needed, the ACK needn't be requested.

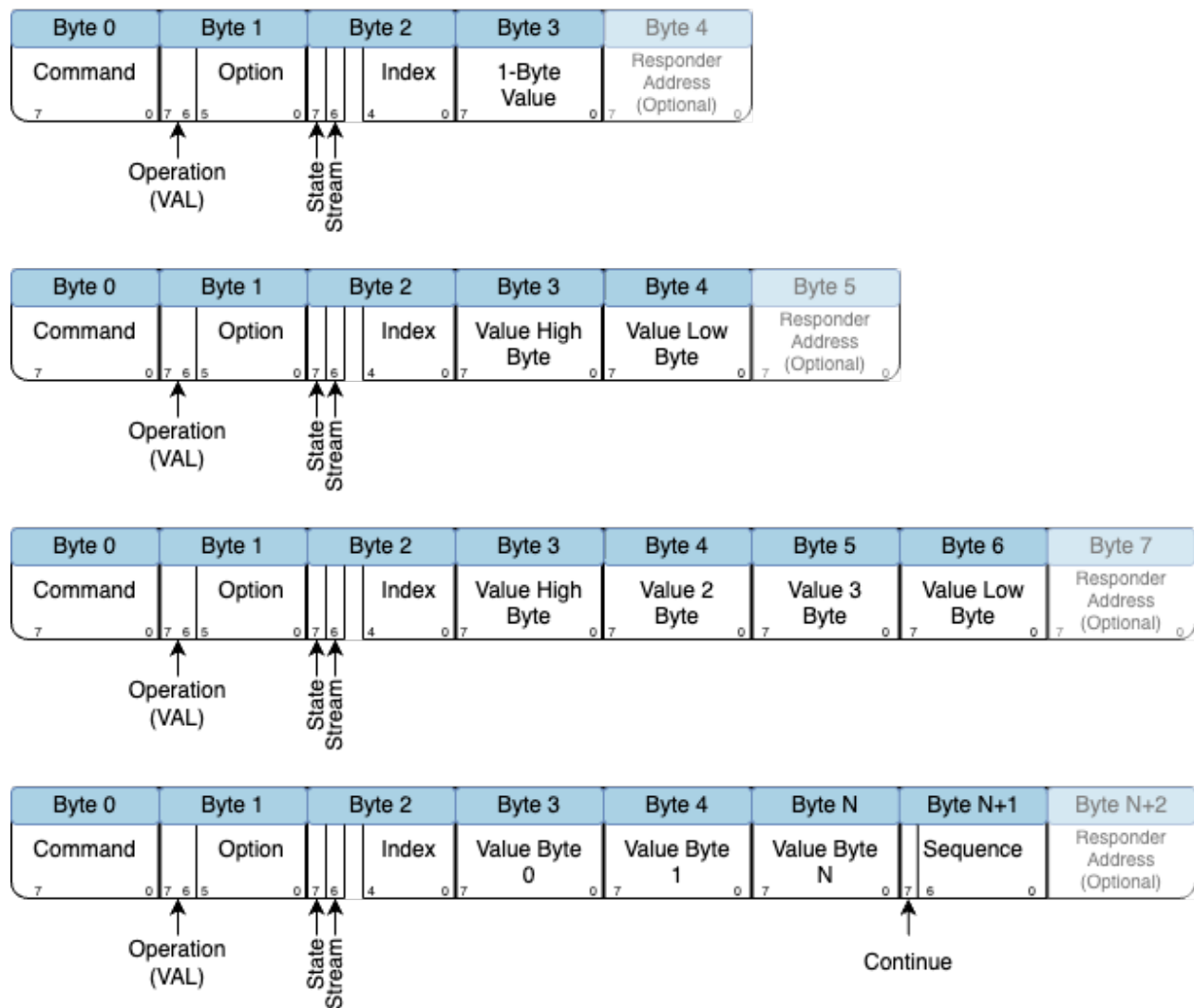


Fig. 2: Packed Byte VAL UEI Structure 1, 2, 4, and N Byte Values

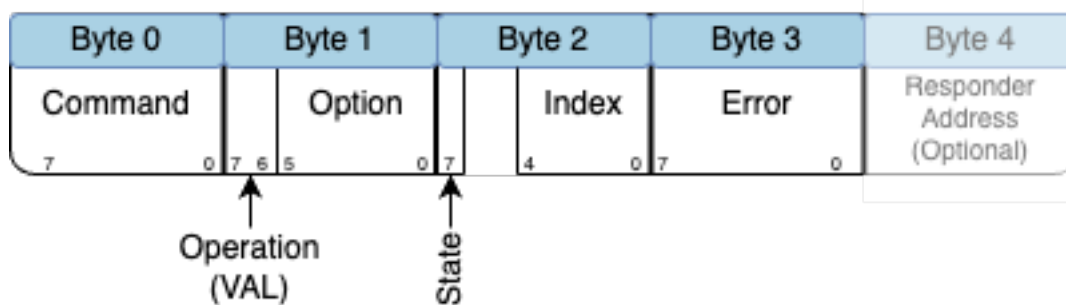


Fig. 3: Packed Byte VAL UEI Error Structure

UEI SET Request

- Command - The command that groups the class of UEI's in which the entity is included (This is part of BrainStem 2.0 command structure).
- Operation - The operation of the UEI which is SET in this case where the possible operations are VAL (0), GET (1), SET (2), or ACK (3).
- Option - The specific option code within the class of UEI's
- Reply - Identifies the requestor so the acknowledgement returns to that requestor where the possible options include none (no acknowledgement), Host (1), I2C (2), or Reflex (3).
- Index - The array index of the class. This allows for multiple groups of classes like ports, servos, motion channels, etc.
- Subindex (optional) - The subindex of a specific index of the class. This allows for there to be sub-components of a class like setting/getting the configuration for a channel of an equalizer class.
- Data - empty, 1, 2, 4, or N bytes for the entity value or trigger being written.
- Continue - Identifies if there are subsequent payloads after this one that should be used in conjunction or if this is the last payload for the value.
- Sequence - Used to identify the payload number for a multi-payload response. If the value is greater than what can fit in 1 payload, the continue bit will be set and there will be multiple payloads each with an incrementing sequence number until the last payload in which the continue bit will be set low.
- ReplyID (optional) - For reply values that require more information, this information is include such as the Reflex Machine thread identifier or the remote module's address over I2C where the response will be sent. If the reply specifies the host, no additional replyID information is needed so it will not be present.

Once a SET is performed, the module responds to the reply location if one was specified. The response is an ACK operation which indicates success or an error. Again, if a reply of "none" was specified, there is no ACK sent anywhere.

UEI ACK Reply

- Command - The command that groups the class of UEI's in which the entity is included (This is part of BrainStem 2.0 command structure).
- Operation - The operation of the UIE which is ACK in this case where the possible operations are VAL (0), GET (1), SET (2), or ACK (3).
- Option - The specific option code within the class of UEI's
- State - Identifies the response state. If no errors occurred, this value is zero. If an error occurred the high bit (7) of this byte will be set.
- Index - The array index of the class. This allows for multiple groups of classes like ports, servos, motion channels, etc.
- Responder Address - The module address from where the ACK is being sent. This helps the requestor verify the completion status of the SET command. When replies are sent to the host, the responder address is implicit in the link protocol so it is not included host acknowledgements.

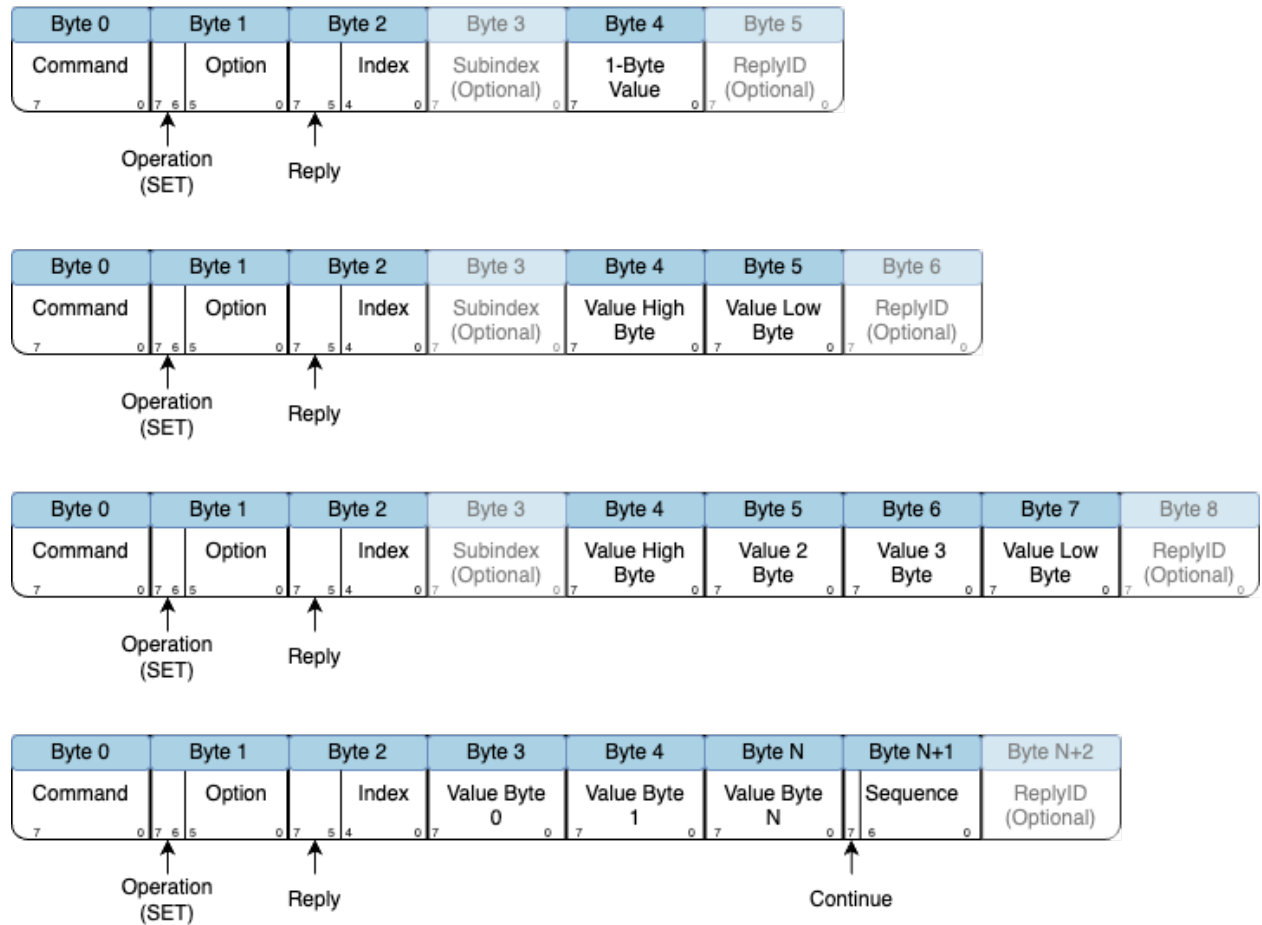


Fig. 4: Packed Byte SET UEI Structure 1, 2, 4, and N Byte Values

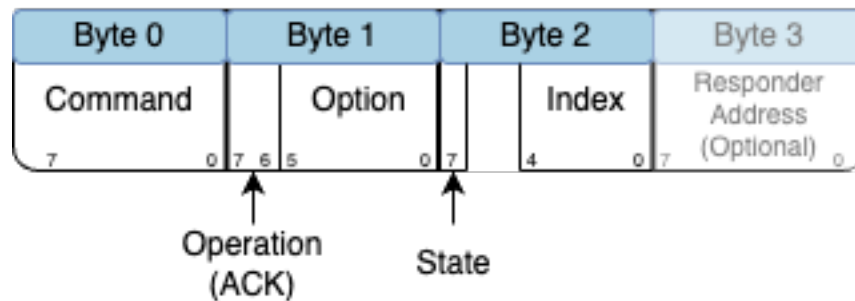


Fig. 5: Packed Byte ACK UEI Structure

UEI ACK Error Handling

In the event of an error such as invalid entity index, configuration, etc., the error bit is set in the ACK state and an *error code* is added to the ACK further describing the error to the SET requestor.



Fig. 6: Packed Byte ACK UEI Error Structure

The Streaming VAL UEI Transaction

There is also a way to configure the device to asynchronously create UEI update payloads that will stream to the requestor upon change of the value. This is very useful for capturing all the voltage measurements on a USB port or being notified of a status change so polling isn't required. The appropriate command, index, option combo is configured via the *Link Class* using the Stream Command.

UEI VAL Streaming

- Command - The command that groups the class of UEI's in which the entity is included (This is part of BrainStem 2.0 command structure).
- Operation - The operation of the UIE which is VAL in this case where the possible operations are VAL (0), GET (1), SET (2), or ACK (3).
- Option - The specific option code within the class of UEI's
- State - Identifies the response state. If no errors occurred, this value is zero. If an error occurred the high bit (7) of this byte will be set.
- Stream - Identifies if this is a streaming payload or not. If the payload is a normal GET/VAL, this value is zero. If the payload is streamed asynchronously bit (6) of this byte will be set.
- Index - The array index of the class. This allows for multiple groups of classes like ports, servos, motion channels, etc.
- Stream Type - Identifies the type of value that will be coming back, 1, 2, 4, or N byte value, with/without Subindex.
- Subindex (optional) - The subindex of a specific index of the class. This allows for there to be sub-components of a class like setting/getting the configuration for a channel of an equalizer class.
- Seconds - The 4 byte value of seconds of uptime for the device.
- uSeconds - The 4 byte value of microseconds of uptime for the device. (To be used in conjunction with seconds of uptime above)
- Data - 1, 2, 4, or N bytes for the entity value that was read. If an error occurred, the error bit is set in the state and the data is always a 1-byte error value describing the error.

- Continue - Identifies if there are subsequent payloads after this one that should be used in conjunction or if this is the last payload for the value.
- Sequence - Used to identify the payload index for a multi-payload response.
- Responder Address - The responding entity's module address so the requestor can potentially match the initial request with this response. For replies to the host, this responder address is implicit in the inbound packet protocol so the responder address is not sent in host responses.

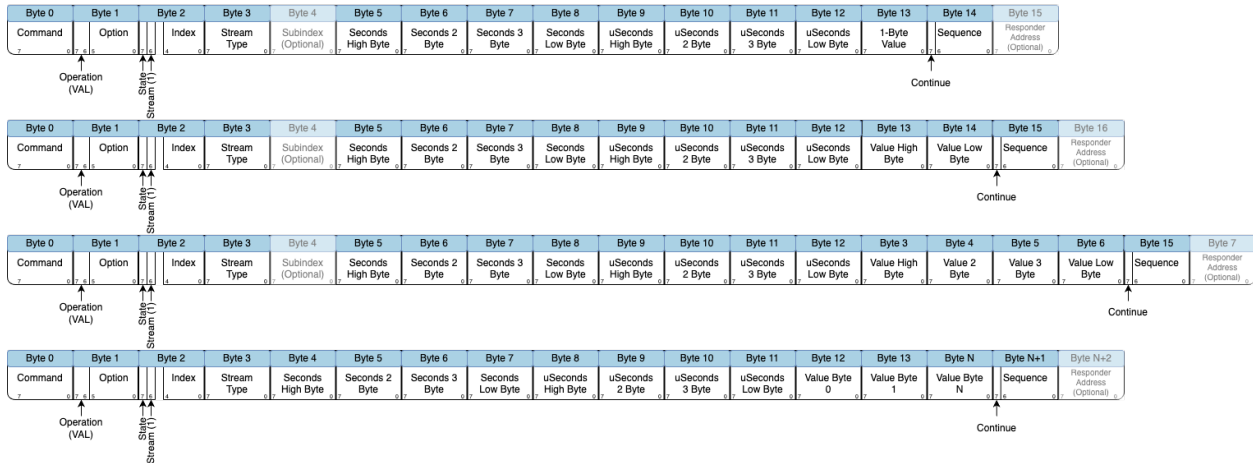


Fig. 7: Packed Byte VAL UEI Stream Structure 1, 2, 4, and N Byte Values

4.7.2 Appendix II: BrainStem Communication Protocol

The BrainStem Communication Protocol is a very light weight transport independent binary packet protocol. The protocol simply imposes a max packet length restriction, and designates two bytes of the packet as “header” bytes which contain information used to address, and handle packets.

The BrainStem protocol is a command protocol. Commands are the foundational communication mechanism for BrainStem modules. Every BrainStem command has a similar structure shown in the following diagram.

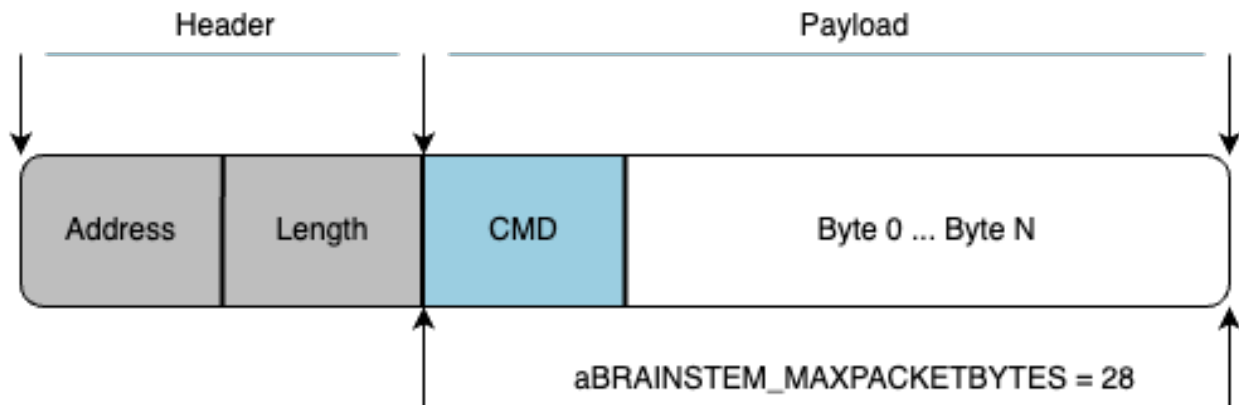


Fig. 8: Typical BrainStem Command Structure

Packet Structure

- **Header** - The packet header consists of the module address and length byte.
- **Payload** - The remainder of the packet is the payload, and consists of the command code followed by data bytes. The packet length cannot exceed aBRAINSTEM_MAXPACKETBYTES (28 bytes).
- **Address** - The BrainStem module address that will receive the packet. This value is an even number that can range from the value 2 to 254. BrainStem module types have different default address values (Check your product Datasheet). The module address can be changed by the user, please see the command reference section on the System command for more information.
- **Length** - The length of the packet Payload in Bytes.
- **Command** - The command code for the BrainStem command. See the reference section on BrainStem commands for more information.
- **Byte 0 .. Byte N** - The command data bytes, a command may impose structure on the data portion of the packet. This is documented in the command reference.

Byte Order

The BrainStem protocol does not specify byte order for the data portion of the packet, but *UEI* datatypes larger than byte are stored in bigEndian byte order.

Command Interaction

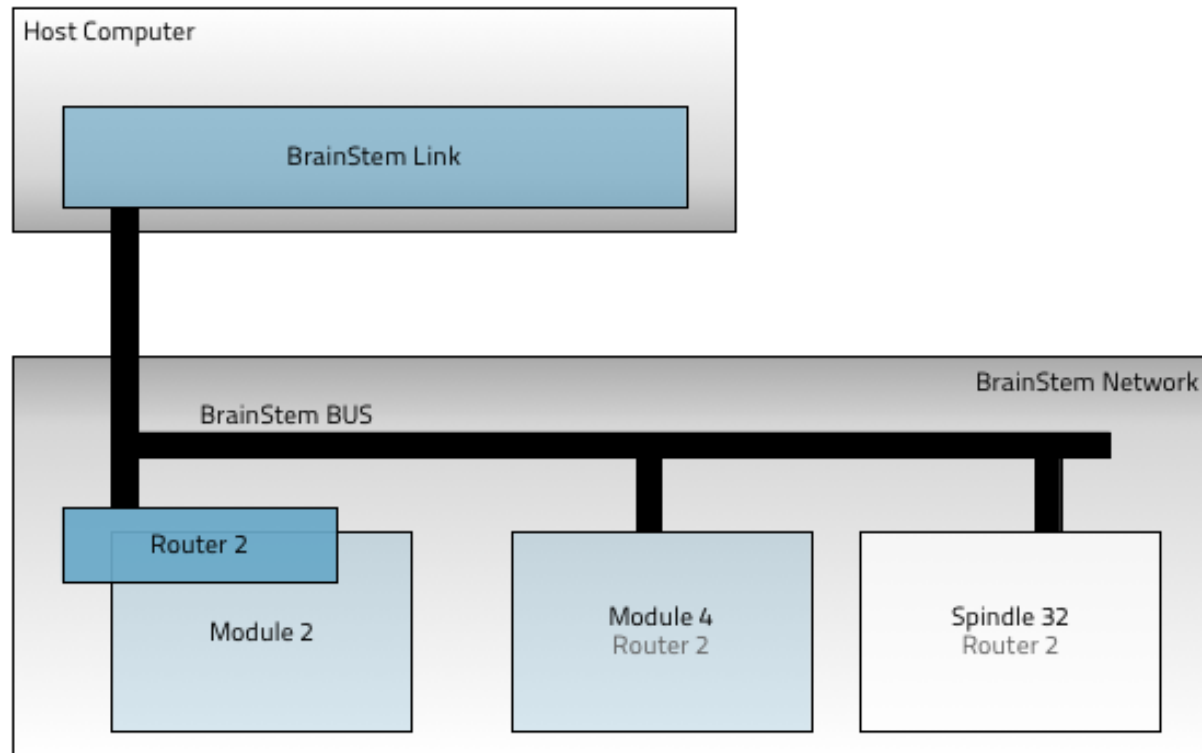
UEI's impose a request response behavior on top of the BrainStem protocol, but the protocol does not itself define a need for a response. Delivery of data is best effort. Some Link transports (TCP/IP) make guarantees about data delivery, this is not part of the BrainStem protocol. Please see specific command documentation to determine whether to expect a response packet.

There are currently two cases when a BrainStem client may receive an error message unrelated to a request. The first case occurs when a module address is given where no such module exists, in this case a cmdMSG packet will be received with a no I2C ack payload. The second case occurs when a Reflex VM exits unexpectedly, a client may receive a cmdMSG packet with a vm exit payload. Please see the Command reference section for more information.

4.7.3 Appendix III: BrainStem Networking

The BrainStem bus is the network backbone of the BrainStem network. Most BrainStem modules, including all MTM modules, use an I2C¹⁴¹ as the hardware transport. The brainstem network is a multiple master I2C fast mode plus (FM+, 1MHz) network. Traffic on this bus generally follows the specification of the BrainStem protocol. Third party devices can be connected to this network, but it is most common to connect I2C peripherals to a BrainStem module's peripheral I2C ports.

BrainStem networks closely mirror standard I2C networks, but aren't necessarily always on an I2C physical network. For example, BrainStem network as described below may use a CAN bus physical network.



¹⁴¹ <http://i2c.info/i2c-bus-specification>

Module Addresses

BrainStem devices rely on having a unique module address on the bus that following I2C conventions. The Brainstem module address is a single unsigned-byte, and can take even (non-odd) values from 2 to 254. Each class of BrainStem module has a specific default base address, listed in the table below. A software offset to this address can be set with the BrainStem API, and MTM modules include a set of hardware offset pins which can be used to modify module addresses with external pin connections.

BrainStem Model	Default Base Address
40Pin BrainStems EtherStem USBStem	2
MTM-EtherStem MTM-USBStem	4
MTM-PM-1 USBHub2x4 USBHub3+ USBHub3c USBCSwitch	6
MTM-IO-Serial	8
MTM-DAQ-2	10
MTM-Relay	12
MTM-Load-1	14
USBCSwitchPro	16

Hardware Offsets

Hardware offset pins are useful when more than one of the same type of module (i.e. modules with the same base address) are installed on a single BrainStem network. Applying a different hardware offset to each module of the same type the modules to seamlessly and automatically be configured on the network for inter-module communication. Further, modules can be simply swapped in and out of the network without needing to pre-configure a module's address before being added to a network. Finally, when a system has more than one of the same type of module in a network, the module's hardware offset can be used to determine the module's physical location and thus its interconnection and intended function.

Each hardware offset pin can be left floating or pulled to ground with a 1k Ω resistor (or smaller) Pins can also simply be shorted to ground. Pin states are only read when the module boots, either from a power cycle, hardware reset or software reset. The hardware offset pins are treated as an inverted binary number which is multiplied by 2 and added the to the module's base address. The hardware offset calculation is detailed in the following table.

Pin0	Pin1	Pin2	Pin3	Address Offset	Base Address	Final Address
NC	NC	NC	NC	0	4	4
0	NC	NC	NC	2	4	6
NC	0	NC	NC	4	4	8
NC	NC	0	NC	8	4	12
NC	NC	NC	0	16	4	20
0	NC	NC	0	2+16=18	4	22

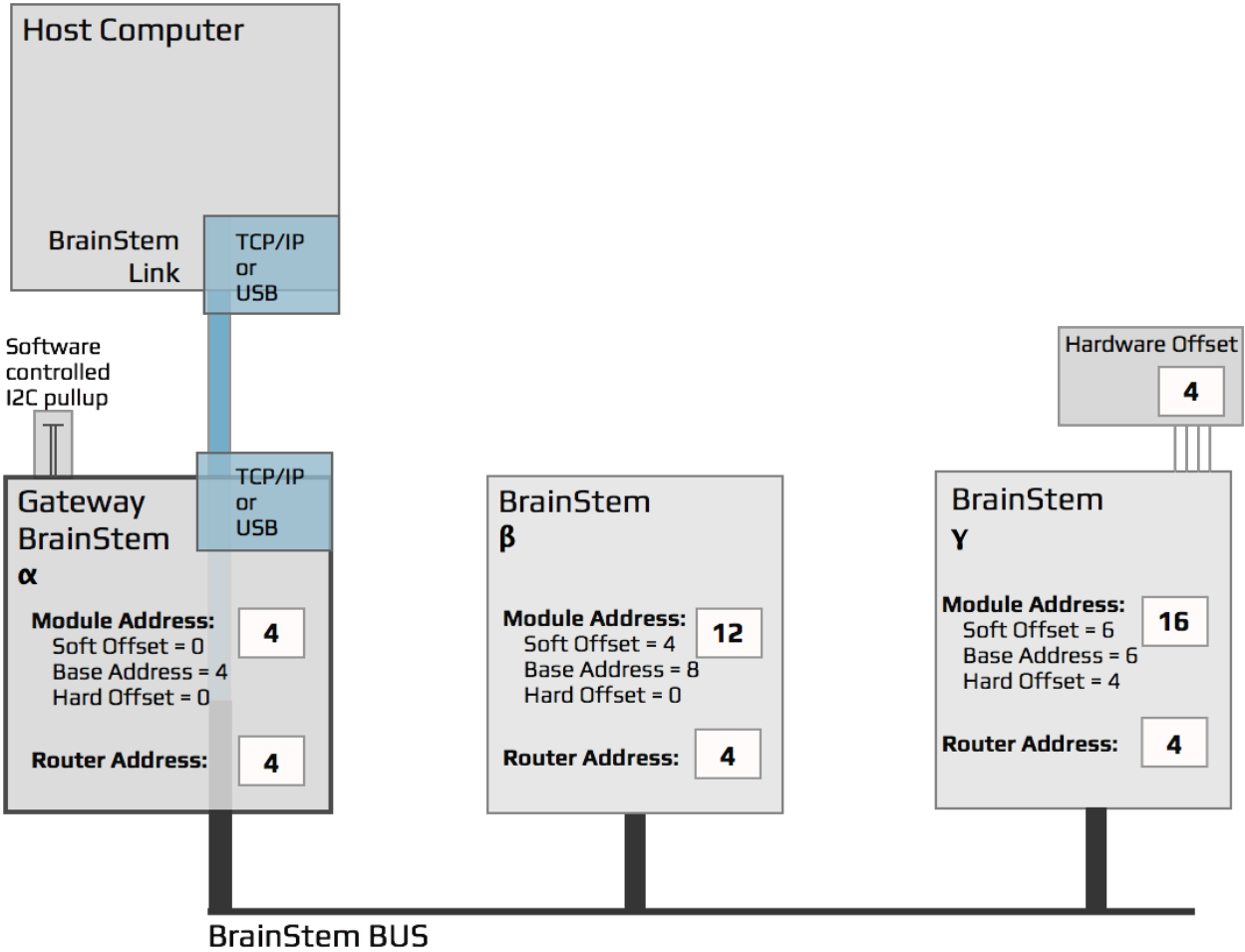
Router Addresses

In addition to the module address, each BrainStem device has a network router address. The router address tells the module which BrainStem in the network is connected to the host; i.e. which BrainStem device is acting as the network gateway. If the router address of the module is set to its own module address, then it will send heartbeat traffic and route host-bound traffic from the BrainStem network through its transport link (e.g. USB or Ethernet). If a module's router address is different from it's module address, it will route any communication intended for the host computer to its router's address on the BrainStem network.

By default, each BrainStem's router address is set to its default base address plus any offset. In this way, each module will communicate over its transport link out of the box. In order to have multiple BrainStem modules communicate across a BrainStem network over one transport link, each module needs have its router address configured. The BrainStem API provides a simple mechanism to quickly configure the router address of all modules on the same BrainStem network: `routeToMe`.

Setting up a BrainStem Network

This section of the appendix will walk you through setting up the network shown in the figure below. This is fairly typical network containing three BrainStem devices since it represents a fully populated MTM development board.



Out of the Box

In the example above the Routing or Gateway BrainStem (α) is set to route through itself; i.e. its module and router addresses are equal. This is the setup that comes with the BrainStem out of the box. To complete the example, two more BrainStem devices are needed and they will have their router and address software offsets changed (described below).

I2C Pull-ups

Most BrainStem use an I2C physical network. I2C relies on bus pull-ups resistors. BrainStem modules have built-in 330 Ω pull-ups to 3.3V which should allow for communication at 1Mbps. There should be no other pull-ups on the network.

Configuring module routers: the quick way

The *system entity* contains the entity `routeToMe`. Calling this from a linked module will temporarily configure all modules on the same BrainStem network to route to the linked module. For example, using the setup from above, we can simply connect to the α module:

```
>>> import brainstem
>>> alpha = brainstem.stem.MTMUSBStem() # or the appropriate module type
>>> alpha.discoverAndConnect(brainstemlink.Spec.USB)
```

Then we tell all other modules to route their traffic to the α module:

```
>>> alpha.system.routeToMe(1)
```

After this, all modules in the network will start receiving and sending heartbeat traffic, and can be connected from the host:

```
>>> beta = brainstem.stem.MTMIOSerial()
>>> beta.connectThroughLinkModule(alpha)
```

Similarly, the γ module can be constructed and connected. All features and abilities of the networked modules are now instantly available to the host software as if they were directly linked the host. This powerful networking allows large networks of modules to be controlled from a single host link.

Setting and saving address offsets and router: the hard(er) way

If it is desirable to configure modules to change their module address or router setting even through power cycles or resets, the BrainStem API provide an interface for directly setting and saving the address offset and the router of each module. This method is more complicated than the `routeToMe` interface, but is available to provide flexibility for complex network setups. Never worry, if a device's router address is saved to a non-default value simply creating a link directly to that module (e.g. via USB) will temporarily re-configure its route to itself so the link can be functional.

The *system entity* contains the options for getting and setting the module offsets and router address of the brainstem module. Module offsets and the router address are applied after a system save and reset. The python interpreter code below sets the module and router addresses for the two modules (β and γ). The same exercise can be done in C++ with almost the same code. For this example, we will assume that both β (beta) and γ (gamma) are new modules and that are directly connected via a USB cable. To simplify this process,

the modules can be connected and configured one at a time. Importing a few key modules makes the following commands bit shorter:

```
>>> import brainstem
>>> from brainstem import link
>>> from brainstem import discover
>>> from brainstem.stem import MTMIOSerial # or other modules needed
```

Then, connecting to the β module is done with:

```
>>> beta = MTMIOSerial() # or the appropriate module type
>>> beta.connect(discover.findFirstModule(link.Spec.USB)) # connect to beta.
```

Then set and save the router and module software offsets with:

```
>>> beta.system.setModuleSoftwareOffset(4)
>>> beta.system.setRouter(4)
>>> beta.system.save()
>>> beta.system.reset() # beta stops communicating here, and will return a timeout on
↳this call.
>>> beta.disconnect() # good practice to always call disconnect
```

After calling reset, the module will be trying to connect and communicate via the router address we defined. This router address is the module address of α , the gateway BrainStem. As such, all host-bound communication will be routed to address 4 on the BrainStem network, instead of going through the module's transport link connector.

The following code sets the module software offset and the router settings on γ to match the diagram. Connecting to γ is done in same way as shown above for β , simply changing the module type to the appropriate module being used.

```
>>> gamma.system.setModuleSoftwareOffset(6)
>>> gamma.system.setRouter(4)
>>> gamma.system.save()
>>> gamma.system.reset() # similarly gamma stops communicating.
>>> gamma.disconnect() # good practice to always call disconnect
```

Now the BrainStem network is configured as described in the diagram above. Moving the link cable to be connected to α will allow the entire network to show up via a single link cable. We can continue to use the same objects created earlier in this process by simply setting the module address to what was configured. This tells the host software what address the module can be reached at:

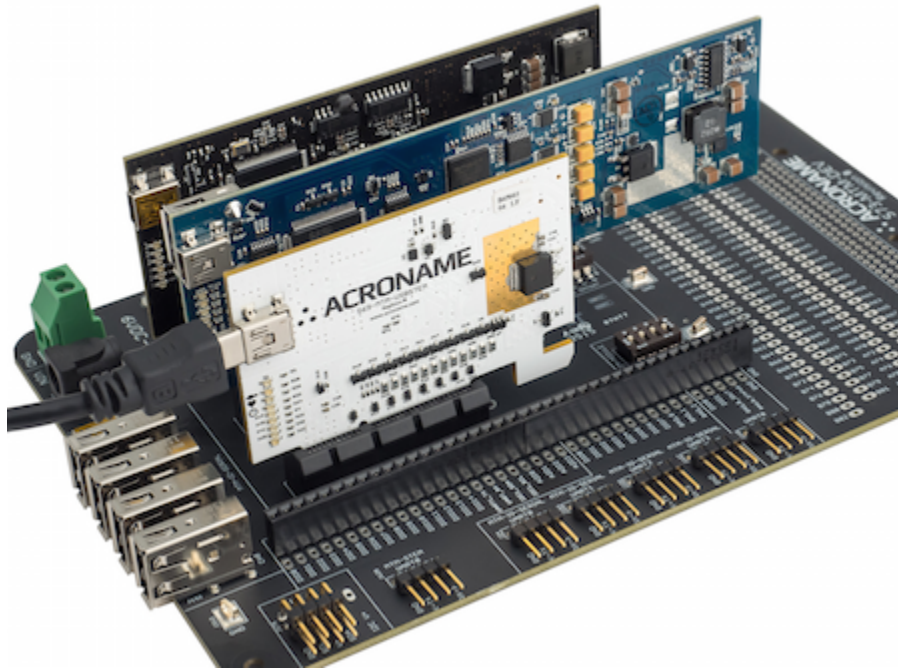
```
>>> beta.setModuleAddress(12) # Set the Module object's address so that we
↳communicate correctly when we reconnect.
>>> gamma.setModuleAddress(16)
```

Finally we connect directly to the Gateway BrainStem α , and then connect β and γ through α 's link. We should now be able to successfully send commands and receive responses on all three brainstems through the single link connection to α .

```
# Connect to the gateway BrainStem. All three stem heartbeats should be active.
>>> alpha.connect(discover.findFirstModule(link.Spec.USB))
>>> beta.connectThroughLinkModule(alpha) # Now reconnect the beta and gamma through
↳the gateway module.
>>> gamma.connectThroughLinkModule(alpha)
```

In Practice

The above example is intentionally complex in order to show the interaction of hardware offsets and software offsets within a BrainStem network. In most applications and with the MTM development board, usually the user will prefer to make minimal changes in order to form the brainstem network.



In a real life use case of an MTM-USBStem acting as the Gateway, and an MTM-IO-Serial and MTM-PM-1 boards acting as the other two devices, the only changes that need to be made are to set the router address of the MTM-IO-Serial, and the MTM-PM-1 to match the MTM-USBStem base address of 4. Each of the modules have unique base addresses so there are no software or hardware offsets to apply. Also, the default object instantiation can be used without having to set the module address.

```
>>> beta.setRouter(4)
>>> gamma.setRouter(4)
>>> beta.save()
>>> gamma.save()
>>> beta.reset()
>>> gamma.reset()
>>> beta.disconnect()
>>> gamma.disconnect()
>>> beta.connectThroughLinkModule(alpha)
>>> gamma.connectThroughLinkModule(alpha)
```


4.7.4 Appendix IV: Updater File Structure

As we discussed in the “*BrainStem Firmware Management*” section, Updater is our new tool updating and recovering your Brainstem modules. After playing around with it a bit you might of noticed that it keeps a history of all the devices it interacts with. In the following appendix I will be explaining the file structure Updater creates on your in or to make things easier on you.

Locating Updater’s files on your machine

Updater stores all its files in your home directory under a hidden folder named “.acroname”. In Mac if you would like to find this directory you can open up your terminal window and type the following.

```
$> cd ~/          #Change to your home directory
$> ls -la         #List files, including hidden files
```

Now lets enter the folder and look around.

```
$> cd ~/.acroname/updater
```

You may have noticed that I skipped over a directory and went straight to the updater folder. Currently there are not any other files/folders in this folder; however, feel free to explore around.

Lets have a look at what is in the updater folder.

```
$> ls -lah        #List files, including hidden files and permissions
```

```
total 0
drwxr-xr-x  7 Mitch  staff   238B Dec  4 10:34 .
drwxr-xr-x  3 Mitch  staff   102B Nov 30 11:08 ..
-rw-r--r--  1 Mitch  staff    0B Nov 30 11:10 .history
drwxr-xr-x  5 Mitch  staff   170B Dec  2 11:40 3797E6F8
drwxr-xr-x  4 Mitch  staff   136B Dec  4 10:34 66F4859B
drwxr-xr-x  5 Mitch  staff   170B Dec  2 11:38 71E3928C
drwxr-xr-x  4 Mitch  staff   136B Nov 30 11:08 856C1C03
```

Exploring the Updater files

Now that we have made it into the Updater file structure lets discuss what we see here. In the example above you will notice that there are 7 items.

- “.” - Standard directory structure: Current directory
- “..” - Standard directory structure: Parent directory
- “.history” - System level history (not current implemented).

The remaining four items are all devices (serial numbers) that the Updater utility has connected too previously.

- 3797E6F8
- 66F4859B
- 71E3928C
- 856C1C03

Exploring the Updater Device files

Lets look into device/directory 66F4859B

```
$> cd 66F4859B
$> ls -lah           #List files, including hidden files and permissions
```

```
total 344
drwxr-xr-x  7 Mitch  staff   238B Dec  4 10:58 .
drwxr-xr-x  7 Mitch  staff   238B Dec  4 11:00 ..
-rw-r--r--  1 Mitch  staff   1.6K Dec  4 10:58 .history
-rw-r--r--  1 Mitch  staff   228B Dec  4 10:58 .settings
-rw-r--r--  1 Mitch  staff    53K Dec  4 10:58 130702287.bird
-rw-r--r--  1 Mitch  staff    52K Dec  4 10:56 75203177.bird
-rw-r--r--  1 Mitch  staff    52K Dec  4 10:57 99528558.bird
```

You will notice a similar layout from the previous example, but there are a few new items we will dig into. It is important to remember that we are now inside a folder for a specific device. Thus all the items in this folder are related to that device only.

“.history”

As you would imagine the “.history” file includes a history of the device and all the actions we have preformed on it from within Updater. By default Updater will include basic information; however, you can also add your own messages to the history file by using the “-l” parameter. Please see *“Using Updater via CLI”* for more information. You may also choose to update this file manually. Lets take a look at the “.history” file.

```
$> vi .history
```

```
2015:12:04:17:34:52 | Created device entry
2015:12:04:17:50:56 |      66F4859B  00      02      04 [USBStem  ]    2.1.5
2015:12:04:17:51:48 |      66F4859B  00      02      04 [USBStem  ]    2.1.5
2015:12:04:17:56:36 | Current settings:
2015:12:04:17:56:36 |      Serial#:[66F4859B, 1727301019]
2015:12:04:17:56:36 |      Module#:[02]
2015:12:04:17:56:36 |      Model#:[04, USBStem]
2015:12:04:17:56:36 |      Firmware Version:  2.1.5
2015:12:04:17:56:36 | Updating device from BIRD file: [/Users/Mitch/.acroname/updater/
66F4859B/75203177.bird]
2015:12:04:17:56:37 | Update successful. Transferred 3 blocks, 57664 total bytes
2015:12:04:17:56:50 |      66F4859B  00      06      04 [USBStem  ]    2.1.3
2015:12:04:17:57:39 | Current settings:
2015:12:04:17:57:39 |      Serial#:[66F4859B, 1727301019]
2015:12:04:17:57:39 |      Module#:[06]
2015:12:04:17:57:39 |      Model#:[04, USBStem]
2015:12:04:17:57:39 |      Firmware Version:  2.1.3
2015:12:04:17:57:39 | Updating device from BIRD file: [/Users/Mitch/.acroname/updater/
66F4859B/99528558.bird]
2015:12:04:17:57:42 | Update successful. Transferred 3 blocks, 57844 total bytes
2015:12:04:17:57:52 |      66F4859B  00      02      04 [USBStem  ]    2.1.4
2015:12:04:17:58:03 | Current settings:
2015:12:04:17:58:03 |      Serial#:[66F4859B, 1727301019]
2015:12:04:17:58:03 |      Module#:[02]
2015:12:04:17:58:03 |      Model#:[04, USBStem]
2015:12:04:17:58:03 |      Firmware Version:  2.1.4
```

(continues on next page)

(continued from previous page)

```

2015:12:04:17:58:03 | Updating device from BIRD file: [/Users/Mitch/.acroname/updater/
66F4859B/130702287.bird]
2015:12:04:17:58:05 | Update successful. Transferred 3 blocks, 59292 total bytes
2015:12:04:17:58:12 |      66F4859B  00      02      04 [USBStem      ]  2.1.5

```

“.settings”

Just like the “.history” file the “.settings” file is also self explanatory. Here Updater information in which it needs to communication with the brainstem module. Although you have read/write access to this file it is recommended that you do not make any changes to this file. Lets take a look at what type of information is stored in the “.settings” file.

```
$> vi .history
```

```

FIRMWARE= 2.1.5
LAST_TRANSFER_DATE=Fri Dec 4 10:58:05 2015
LAST_TRANSFER_VERSION=/Users/Mitch/.acroname/updater/66F4859B/130702287.bird
LINK_TYPE=USB
MODEL_NUMBER=4
MODULE_NAME=USBStem
MODULE_NUMBER=2
SERIAL_NUMBER=0x66F4859B

```

“.bird” Files

The remaining 4 items from the *device files* listed above are called “.bird” files. This is the file type in which we store our firmware. For this particular device I have updated the firmware 3 times and thus have 3 “.bird” files stored under this device.

A

aBaudRate (*C++ enum*), 962
 aBaudRate::aBAUD_115200 (*C++ enumerator*), 962
 aBaudRate::aBAUD_19200 (*C++ enumerator*), 962
 aBaudRate::aBAUD_230400 (*C++ enumerator*), 962
 aBaudRate::aBAUD_2400 (*C++ enumerator*), 962
 aBaudRate::aBAUD_38400 (*C++ enumerator*), 962
 aBaudRate::aBAUD_4800 (*C++ enumerator*), 962
 aBaudRate::aBAUD_57600 (*C++ enumerator*), 962
 aBaudRate::aBAUD_9600 (*C++ enumerator*), 962
 aBRAINSTEM_MAXPACKETBYTES (*C macro*), 892
 Acroname::BrainStem2CLI::aErr (*C++ enum*), 1175
 Acroname::BrainStem2CLI::aErr::aErrAsyncReturn (*C++ enumerator*), 1177
 Acroname::BrainStem2CLI::aErr::aErrBusy (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrCancel (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrConfiguration (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrConnection (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrDuplicate (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrEOF (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrFileNameLength (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrIndexRange (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrInitialization (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrInvalidEntity (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrInvalidOption (*C++ enumerator*), 1177
 Acroname::BrainStem2CLI::aErr::aErrIO (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrMedia (*C++ enumerator*), 1177
 Acroname::BrainStem2CLI::aErr::aErrMemory (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrMode (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrNone (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrNotFound (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrNotReady (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrOverrun (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrPacket (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrParam (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrParse (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrPermission (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrRange (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrRead (*C++ enumerator*), 1175
 Acroname::BrainStem2CLI::aErr::aErrResource (*C++ enumerator*), 1177
 Acroname::BrainStem2CLI::aErr::aErrShortCommand (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrSize (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrStreamStale (*C++ enumerator*), 1177
 Acroname::BrainStem2CLI::aErr::aErrTimeout (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrUnimplemented (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrUnknown (*C++ enumerator*), 1177
 Acroname::BrainStem2CLI::aErr::aErrVersion (*C++ enumerator*), 1176
 Acroname::BrainStem2CLI::aErr::aErrWrite (*C++ enumerator*), 1175

Acroname::BrainStem2CLI::AnalogClass (C++ class), 1185
Acroname::BrainStem2CLI::AnalogClass::~~AnalogClass (C++ function), 1185
Acroname::BrainStem2CLI::AnalogClass::AnalogClass (C++ function), 1185
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureNumberOfSamples (C++ function), 1188
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureSampleRate (C++ function), 1187
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureState (C++ function), 1188
Acroname::BrainStem2CLI::AnalogClass::getConfiguration (C++ function), 1187
Acroname::BrainStem2CLI::AnalogClass::getEnable (C++ function), 1186
Acroname::BrainStem2CLI::AnalogClass::getRange (C++ function), 1186
Acroname::BrainStem2CLI::AnalogClass::getValue (C++ function), 1185
Acroname::BrainStem2CLI::AnalogClass::getVoltage (C++ function), 1185
Acroname::BrainStem2CLI::AnalogClass::initiateBulkCapture (C++ function), 1188
Acroname::BrainStem2CLI::AnalogClass::setBulkCaptureNumberOfSamples (C++ function), 1187
Acroname::BrainStem2CLI::AnalogClass::setBulkCaptureSampleRate (C++ function), 1187
Acroname::BrainStem2CLI::AnalogClass::setConfiguration (C++ function), 1187
Acroname::BrainStem2CLI::AnalogClass::setEnable (C++ function), 1187
Acroname::BrainStem2CLI::AnalogClass::setRange (C++ function), 1186
Acroname::BrainStem2CLI::AnalogClass::setValue (C++ function), 1186
Acroname::BrainStem2CLI::AnalogClass::setVoltage (C++ function), 1186
Acroname::BrainStem2CLI::AppClass (C++ class), 1188
Acroname::BrainStem2CLI::AppClass::~~AppClass (C++ function), 1189
Acroname::BrainStem2CLI::AppClass::AppClass (C++ function), 1189
Acroname::BrainStem2CLI::AppClass::execute (C++ function), 1189
Acroname::BrainStem2CLI::ClockClass (C++ class), 1190
Acroname::BrainStem2CLI::ClockClass::~~ClockClass (C++ function), 1190
Acroname::BrainStem2CLI::ClockClass::ClockClass (C++ function), 1190
Acroname::BrainStem2CLI::ClockClass::getDay (C++ function), 1191
Acroname::BrainStem2CLI::ClockClass::getHour (C++ function), 1191
Acroname::BrainStem2CLI::ClockClass::getMinute (C++ function), 1191
Acroname::BrainStem2CLI::ClockClass::getMonth (C++ function), 1191
Acroname::BrainStem2CLI::ClockClass::getSecond (C++ function), 1192
Acroname::BrainStem2CLI::ClockClass::getYear (C++ function), 1190
Acroname::BrainStem2CLI::ClockClass::setDay (C++ function), 1191
Acroname::BrainStem2CLI::ClockClass::setHour (C++ function), 1191
Acroname::BrainStem2CLI::ClockClass::setMinute (C++ function), 1192
Acroname::BrainStem2CLI::ClockClass::setMonth (C++ function), 1191
Acroname::BrainStem2CLI::ClockClass::setSecond (C++ function), 1192
Acroname::BrainStem2CLI::ClockClass::setYear (C++ function), 1190
Acroname::BrainStem2CLI::DeviceNode (C++ class), 1177
Acroname::BrainStem2CLI::DeviceNode::~~DeviceNode (C++ function), 1178
Acroname::BrainStem2CLI::DeviceNode::DeviceNode (C++ function), 1178
Acroname::BrainStem2CLI::DeviceNode::hubPort (C++ member), 1178
Acroname::BrainStem2CLI::DeviceNode::hubSerialNumber (C++ member), 1178
Acroname::BrainStem2CLI::DeviceNode::idProduct (C++ member), 1178
Acroname::BrainStem2CLI::DeviceNode::idVendor (C++ member), 1178
Acroname::BrainStem2CLI::DeviceNode::speed (C++ member), 1178
Acroname::BrainStem2CLI::DigitalClass (C++ class), 1192
Acroname::BrainStem2CLI::DigitalClass::~~DigitalClass (C++ function), 1192
Acroname::BrainStem2CLI::DigitalClass::DigitalClass (C++ function), 1192
Acroname::BrainStem2CLI::DigitalClass::getConfiguration (C++ function), 1193
Acroname::BrainStem2CLI::DigitalClass::getLinkChannel (C++ function), 1194
Acroname::BrainStem2CLI::DigitalClass::getState (C++ function), 1193
Acroname::BrainStem2CLI::DigitalClass::getStateAll (C++ function), 1194
Acroname::BrainStem2CLI::DigitalClass::getValue (C++ function), 1194
Acroname::BrainStem2CLI::DigitalClass::setConfiguration (C++ function), 1193
Acroname::BrainStem2CLI::DigitalClass::setLinkChannel (C++ function), 1194
Acroname::BrainStem2CLI::DigitalClass::setState (C++ function), 1193
Acroname::BrainStem2CLI::DigitalClass::setStateAll (C++ function), 1194
Acroname::BrainStem2CLI::DigitalClass::setValue (C++ function), 1194
Acroname::BrainStem2CLI::EqualizerClass (C++ class), 1195
Acroname::BrainStem2CLI::EqualizerClass::~~EqualizerClass (C++ function), 1195
Acroname::BrainStem2CLI::EqualizerClass::EqualizerClass (C++ function), 1195
Acroname::BrainStem2CLI::EqualizerClass::getReceiverConfig (C++ function), 1195
Acroname::BrainStem2CLI::EqualizerClass::getTransmitterConfig (C++ function), 1196
Acroname::BrainStem2CLI::EqualizerClass::setReceiverConfig (C++ function), 1195
Acroname::BrainStem2CLI::EqualizerClass::setTransmitterConfig (C++ function), 1195
Acroname::BrainStem2CLI::EthernetClass (C++ class), 1196
Acroname::BrainStem2CLI::EthernetClass::~~EthernetClass (C++ function), 1196

Acroname::BrainStem2CLI::EthernetClass::EthernetClass (C++ function), 1196
 Acroname::BrainStem2CLI::EthernetClass::getEnabled (C++ function), 1196
 Acroname::BrainStem2CLI::EthernetClass::getHostname (C++ function), 1200
 Acroname::BrainStem2CLI::EthernetClass::getInterfacePort (C++ function), 1201
 Acroname::BrainStem2CLI::EthernetClass::getIPv4Address (C++ function), 1198
 Acroname::BrainStem2CLI::EthernetClass::getIPv4DNSAddress (C++ function), 1200
 Acroname::BrainStem2CLI::EthernetClass::getIPv4Gateway (C++ function), 1199
 Acroname::BrainStem2CLI::EthernetClass::getIPv4Netmask (C++ function), 1199
 Acroname::BrainStem2CLI::EthernetClass::getMACAddress (C++ function), 1200
 Acroname::BrainStem2CLI::EthernetClass::getNetworkConfiguration (C++ function), 1197
 Acroname::BrainStem2CLI::EthernetClass::getStaticIPv4Address (C++ function), 1197
 Acroname::BrainStem2CLI::EthernetClass::getStaticIPv4DNSAddress (C++ function), 1199
 Acroname::BrainStem2CLI::EthernetClass::getStaticIPv4Gateway (C++ function), 1198
 Acroname::BrainStem2CLI::EthernetClass::getStaticIPv4Netmask (C++ function), 1198
 Acroname::BrainStem2CLI::EthernetClass::setEnabled (C++ function), 1196
 Acroname::BrainStem2CLI::EthernetClass::setHostname (C++ function), 1200
 Acroname::BrainStem2CLI::EthernetClass::setInterfacePort (C++ function), 1200
 Acroname::BrainStem2CLI::EthernetClass::setNetworkConfiguration (C++ function), 1197
 Acroname::BrainStem2CLI::EthernetClass::setStaticIPv4Address (C++ function), 1197
 Acroname::BrainStem2CLI::EthernetClass::setStaticIPv4DNSAddress (C++ function), 1199
 Acroname::BrainStem2CLI::EthernetClass::setStaticIPv4Gateway (C++ function), 1198
 Acroname::BrainStem2CLI::EthernetClass::setStaticIPv4Netmask (C++ function), 1198
 Acroname::BrainStem2CLI::HDBaseTClass (C++ class), 1201
 Acroname::BrainStem2CLI::HDBaseTClass::~HDBaseTClass (C++ function), 1201
 Acroname::BrainStem2CLI::HDBaseTClass::getCableLength (C++ function), 1202
 Acroname::BrainStem2CLI::HDBaseTClass::getEncodingState (C++ function), 1203
 Acroname::BrainStem2CLI::HDBaseTClass::getFirmwareVersion (C++ function), 1202
 Acroname::BrainStem2CLI::HDBaseTClass::getLinkRole (C++ function), 1203
 Acroname::BrainStem2CLI::HDBaseTClass::getLinkUtilization (C++ function), 1203
 Acroname::BrainStem2CLI::HDBaseTClass::getMSEA (C++ function), 1202
 Acroname::BrainStem2CLI::HDBaseTClass::getMSEB (C++ function), 1202
 Acroname::BrainStem2CLI::HDBaseTClass::getRetransmissionRate (C++ function), 1202
 Acroname::BrainStem2CLI::HDBaseTClass::getSerialNumber (C++ function), 1201
 Acroname::BrainStem2CLI::HDBaseTClass::getState (C++ function), 1202
 Acroname::BrainStem2CLI::HDBaseTClass::getUSB2DeviceTree (C++ function), 1203
 Acroname::BrainStem2CLI::HDBaseTClass::getUSB3DeviceTree (C++ function), 1203
 Acroname::BrainStem2CLI::HDBaseTClass::HDBaseTClass (C++ function), 1201
 Acroname::BrainStem2CLI::HDBaseTClass::setLinkRole (C++ function), 1203
 Acroname::BrainStem2CLI::I2CClass (C++ class), 1204
 Acroname::BrainStem2CLI::I2CClass::~I2CClass (C++ function), 1204
 Acroname::BrainStem2CLI::I2CClass::getSpeed (C++ function), 1205
 Acroname::BrainStem2CLI::I2CClass::I2CClass (C++ function), 1204
 Acroname::BrainStem2CLI::I2CClass::read (C++ function), 1204
 Acroname::BrainStem2CLI::I2CClass::setPullup (C++ function), 1205
 Acroname::BrainStem2CLI::I2CClass::setSpeed (C++ function), 1205
 Acroname::BrainStem2CLI::I2CClass::write (C++ function), 1204
 Acroname::BrainStem2CLI::ModuleClass (C++ class), 1179
 Acroname::BrainStem2CLI::ModuleClass::~ModuleClass (C++ function), 1179
 Acroname::BrainStem2CLI::ModuleClass::connect (C++ function), 1180, 1181
 Acroname::BrainStem2CLI::ModuleClass::connectFromSpec (C++ function), 1180
 Acroname::BrainStem2CLI::ModuleClass::connectThroughLinkModule (C++ function), 1181
 Acroname::BrainStem2CLI::ModuleClass::disconnect (C++ function), 1181
 Acroname::BrainStem2CLI::ModuleClass::discoverAndConnect (C++ function), 1179, 1180
 Acroname::BrainStem2CLI::ModuleClass::getConfig (C++ function), 1182
 Acroname::BrainStem2CLI::ModuleClass::getIPv4Interfaces (C++ function), 1182
 Acroname::BrainStem2CLI::ModuleClass::getModuleAddress (C++ function), 1182
 Acroname::BrainStem2CLI::ModuleClass::getStatus (C++ function), 1181
 Acroname::BrainStem2CLI::ModuleClass::hasUEI (C++ function), 1182
 Acroname::BrainStem2CLI::ModuleClass::isConnected (C++ function), 1181
 Acroname::BrainStem2CLI::ModuleClass::ModuleClass (C++ function), 1179
 Acroname::BrainStem2CLI::ModuleClass::reconnect (C++ function), 1181
 Acroname::BrainStem2CLI::ModuleClass::setConfig (C++ function), 1182
 Acroname::BrainStem2CLI::ModuleClass::setModuleAddress (C++ function), 1182
 Acroname::BrainStem2CLI::ModuleClass::setNetworkingMode (C++ function), 1183
 Acroname::BrainStem2CLI::MuxClass (C++ class), 1205
 Acroname::BrainStem2CLI::MuxClass::~MuxClass (C++ function), 1206
 Acroname::BrainStem2CLI::MuxClass::getChannel (C++ function), 1206
 Acroname::BrainStem2CLI::MuxClass::getChannelVoltage (C++ function), 1206

Acroname::BrainStem2CLI::MuxClass::getConfiguration (C++ function), 1207
Acroname::BrainStem2CLI::MuxClass::getEnable (C++ function), 1206
Acroname::BrainStem2CLI::MuxClass::getSplitMode (C++ function), 1207
Acroname::BrainStem2CLI::MuxClass::MuxClass (C++ function), 1206
Acroname::BrainStem2CLI::MuxClass::setChannel (C++ function), 1206
Acroname::BrainStem2CLI::MuxClass::setConfiguration (C++ function), 1207
Acroname::BrainStem2CLI::MuxClass::setEnable (C++ function), 1206
Acroname::BrainStem2CLI::MuxClass::setSplitMode (C++ function), 1207
Acroname::BrainStem2CLI::PoEClass (C++ class), 1208
Acroname::BrainStem2CLI::PoEClass::~~PoEClass (C++ function), 1208
Acroname::BrainStem2CLI::PoEClass::getPairAccumulatedPower (C++ function), 1211
Acroname::BrainStem2CLI::PoEClass::getPairCapacitance (C++ function), 1211
Acroname::BrainStem2CLI::PoEClass::getPairCurrent (C++ function), 1210
Acroname::BrainStem2CLI::PoEClass::getPairDetectionStatus (C++ function), 1210
Acroname::BrainStem2CLI::PoEClass::getPairDiscoveredClass (C++ function), 1210
Acroname::BrainStem2CLI::PoEClass::getPairEnabled (C++ function), 1208
Acroname::BrainStem2CLI::PoEClass::getPairPower (C++ function), 1211
Acroname::BrainStem2CLI::PoEClass::getPairRequestedClass (C++ function), 1209
Acroname::BrainStem2CLI::PoEClass::getPairResistance (C++ function), 1211
Acroname::BrainStem2CLI::PoEClass::getPairSourcingClass (C++ function), 1209
Acroname::BrainStem2CLI::PoEClass::getPairVoltage (C++ function), 1210
Acroname::BrainStem2CLI::PoEClass::getPowerMode (C++ function), 1208
Acroname::BrainStem2CLI::PoEClass::getPowerState (C++ function), 1209
Acroname::BrainStem2CLI::PoEClass::getTotalAccumulatedPower (C++ function), 1212
Acroname::BrainStem2CLI::PoEClass::getTotalPower (C++ function), 1211
Acroname::BrainStem2CLI::PoEClass::PoEClass (C++ function), 1208
Acroname::BrainStem2CLI::PoEClass::setPairAccumulatedPower (C++ function), 1212
Acroname::BrainStem2CLI::PoEClass::setPairEnabled (C++ function), 1208
Acroname::BrainStem2CLI::PoEClass::setPairSourcingClass (C++ function), 1209
Acroname::BrainStem2CLI::PoEClass::setPowerMode (C++ function), 1209
Acroname::BrainStem2CLI::PoEClass::setTotalAccumulatedPower (C++ function), 1212
Acroname::BrainStem2CLI::PointerClass (C++ class), 1212
Acroname::BrainStem2CLI::PointerClass::~~PointerClass (C++ function), 1213
Acroname::BrainStem2CLI::PointerClass::getChar (C++ function), 1214
Acroname::BrainStem2CLI::PointerClass::getInt (C++ function), 1215
Acroname::BrainStem2CLI::PointerClass::getMode (C++ function), 1213
Acroname::BrainStem2CLI::PointerClass::getOffset (C++ function), 1213
Acroname::BrainStem2CLI::PointerClass::getShort (C++ function), 1215
Acroname::BrainStem2CLI::PointerClass::getTransferStore (C++ function), 1214
Acroname::BrainStem2CLI::PointerClass::initiateTransferFromStore (C++ function), 1214
Acroname::BrainStem2CLI::PointerClass::initiateTransferToStore (C++ function), 1214
Acroname::BrainStem2CLI::PointerClass::PointerClass (C++ function), 1213
Acroname::BrainStem2CLI::PointerClass::setChar (C++ function), 1214
Acroname::BrainStem2CLI::PointerClass::setInt (C++ function), 1215
Acroname::BrainStem2CLI::PointerClass::setMode (C++ function), 1213
Acroname::BrainStem2CLI::PointerClass::setOffset (C++ function), 1213
Acroname::BrainStem2CLI::PointerClass::setShort (C++ function), 1215
Acroname::BrainStem2CLI::PointerClass::setTransferStore (C++ function), 1214
Acroname::BrainStem2CLI::PORT_SPEED (C++ enum), 1177
Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_FULL (C++ enumerator), 1177
Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_HIGH (C++ enumerator), 1177
Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_LOW (C++ enumerator), 1177
Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_SUPER (C++ enumerator), 1177
Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_SUPER_PLUS (C++ enumerator), 1177
Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_UNKNOWN (C++ enumerator), 1177
Acroname::BrainStem2CLI::PortClass (C++ class), 1215
Acroname::BrainStem2CLI::PortClass::~~PortClass (C++ function), 1216
Acroname::BrainStem2CLI::PortClass::getAllocatedPower (C++ function), 1223
Acroname::BrainStem2CLI::PortClass::getAvailablePower (C++ function), 1223
Acroname::BrainStem2CLI::PortClass::getCC1AccumulatedPower (C++ function), 1229
Acroname::BrainStem2CLI::PortClass::getCC1Current (C++ function), 1229
Acroname::BrainStem2CLI::PortClass::getCC1Enabled (C++ function), 1221
Acroname::BrainStem2CLI::PortClass::getCC1State (C++ function), 1227
Acroname::BrainStem2CLI::PortClass::getCC1Voltage (C++ function), 1228
Acroname::BrainStem2CLI::PortClass::getCC2AccumulatedPower (C++ function), 1229
Acroname::BrainStem2CLI::PortClass::getCC2Current (C++ function), 1229
Acroname::BrainStem2CLI::PortClass::getCC2Enabled (C++ function), 1221
Acroname::BrainStem2CLI::PortClass::getCC2State (C++ function), 1227

Acroname::BrainStem2CLI::PortClass::getCC2Voltage (C++ function), 1228
 Acroname::BrainStem2CLI::PortClass::getCCCurrentLimit (C++ function), 1224
 Acroname::BrainStem2CLI::PortClass::getCCEnabled (C++ function), 1220
 Acroname::BrainStem2CLI::PortClass::getCurrentLimit (C++ function), 1222
 Acroname::BrainStem2CLI::PortClass::getCurrentLimitMode (C++ function), 1223
 Acroname::BrainStem2CLI::PortClass::getDataEnabled (C++ function), 1217
 Acroname::BrainStem2CLI::PortClass::getDataHS1Enabled (C++ function), 1218
 Acroname::BrainStem2CLI::PortClass::getDataHS2Enabled (C++ function), 1218
 Acroname::BrainStem2CLI::PortClass::getDataHSEnabled (C++ function), 1217
 Acroname::BrainStem2CLI::PortClass::getDataHSRoutingBehavior (C++ function), 1225
 Acroname::BrainStem2CLI::PortClass::getDataRole (C++ function), 1219
 Acroname::BrainStem2CLI::PortClass::getDataSpeed (C++ function), 1222
 Acroname::BrainStem2CLI::PortClass::getDataSS1Enabled (C++ function), 1219
 Acroname::BrainStem2CLI::PortClass::getDataSS2Enabled (C++ function), 1219
 Acroname::BrainStem2CLI::PortClass::getDataSSEnabled (C++ function), 1218
 Acroname::BrainStem2CLI::PortClass::getDataSSRoutingBehavior (C++ function), 1225
 Acroname::BrainStem2CLI::PortClass::getEnabled (C++ function), 1217
 Acroname::BrainStem2CLI::PortClass::getErrors (C++ function), 1222
 Acroname::BrainStem2CLI::PortClass::getHSBoost (C++ function), 1227
 Acroname::BrainStem2CLI::PortClass::getMode (C++ function), 1222
 Acroname::BrainStem2CLI::PortClass::getName (C++ function), 1224
 Acroname::BrainStem2CLI::PortClass::getPowerEnabled (C++ function), 1219
 Acroname::BrainStem2CLI::PortClass::getPowerLimit (C++ function), 1223
 Acroname::BrainStem2CLI::PortClass::getPowerLimitMode (C++ function), 1224
 Acroname::BrainStem2CLI::PortClass::getPowerMode (C++ function), 1216
 Acroname::BrainStem2CLI::PortClass::getSBU1Voltage (C++ function), 1228
 Acroname::BrainStem2CLI::PortClass::getSBU2Voltage (C++ function), 1228
 Acroname::BrainStem2CLI::PortClass::getState (C++ function), 1222
 Acroname::BrainStem2CLI::PortClass::getVbusAccumulatedPower (C++ function), 1226
 Acroname::BrainStem2CLI::PortClass::getVbusCurrent (C++ function), 1216
 Acroname::BrainStem2CLI::PortClass::getVbusVoltage (C++ function), 1216
 Acroname::BrainStem2CLI::PortClass::getVconn1Enabled (C++ function), 1220
 Acroname::BrainStem2CLI::PortClass::getVconn2Enabled (C++ function), 1220
 Acroname::BrainStem2CLI::PortClass::getVconnAccumulatedPower (C++ function), 1226
 Acroname::BrainStem2CLI::PortClass::getVconnCurrent (C++ function), 1216
 Acroname::BrainStem2CLI::PortClass::getVconnEnabled (C++ function), 1220
 Acroname::BrainStem2CLI::PortClass::getVconnVoltage (C++ function), 1216
 Acroname::BrainStem2CLI::PortClass::getVoltageSetpoint (C++ function), 1221
 Acroname::BrainStem2CLI::PortClass::PortClass (C++ function), 1216
 Acroname::BrainStem2CLI::PortClass::resetVbusAccumulatedPower (C++ function), 1226
 Acroname::BrainStem2CLI::PortClass::resetVconnAccumulatedPower (C++ function), 1226
 Acroname::BrainStem2CLI::PortClass::setCC1AccumulatedPower (C++ function), 1229
 Acroname::BrainStem2CLI::PortClass::setCC1Enabled (C++ function), 1221
 Acroname::BrainStem2CLI::PortClass::setCC2AccumulatedPower (C++ function), 1229
 Acroname::BrainStem2CLI::PortClass::setCC2Enabled (C++ function), 1221
 Acroname::BrainStem2CLI::PortClass::setCCCurrentLimit (C++ function), 1225
 Acroname::BrainStem2CLI::PortClass::setCCEnabled (C++ function), 1221
 Acroname::BrainStem2CLI::PortClass::setCurrentLimit (C++ function), 1223
 Acroname::BrainStem2CLI::PortClass::setCurrentLimitMode (C++ function), 1223
 Acroname::BrainStem2CLI::PortClass::setDataEnabled (C++ function), 1217
 Acroname::BrainStem2CLI::PortClass::setDataHS1Enabled (C++ function), 1218
 Acroname::BrainStem2CLI::PortClass::setDataHS2Enabled (C++ function), 1218
 Acroname::BrainStem2CLI::PortClass::setDataHSEnabled (C++ function), 1217
 Acroname::BrainStem2CLI::PortClass::setDataHSRoutingBehavior (C++ function), 1225
 Acroname::BrainStem2CLI::PortClass::setDataSS1Enabled (C++ function), 1219
 Acroname::BrainStem2CLI::PortClass::setDataSS2Enabled (C++ function), 1219
 Acroname::BrainStem2CLI::PortClass::setDataSSEnabled (C++ function), 1218
 Acroname::BrainStem2CLI::PortClass::setDataSSRoutingBehavior (C++ function), 1226
 Acroname::BrainStem2CLI::PortClass::setEnabled (C++ function), 1217
 Acroname::BrainStem2CLI::PortClass::setHSBoost (C++ function), 1227
 Acroname::BrainStem2CLI::PortClass::setMode (C++ function), 1222
 Acroname::BrainStem2CLI::PortClass::setName (C++ function), 1224
 Acroname::BrainStem2CLI::PortClass::setPowerEnabled (C++ function), 1219
 Acroname::BrainStem2CLI::PortClass::setPowerLimit (C++ function), 1224
 Acroname::BrainStem2CLI::PortClass::setPowerLimitMode (C++ function), 1224
 Acroname::BrainStem2CLI::PortClass::setPowerMode (C++ function), 1217
 Acroname::BrainStem2CLI::PortClass::setVbusAccumulatedPower (C++ function), 1226
 Acroname::BrainStem2CLI::PortClass::setVconn1Enabled (C++ function), 1220

Acroname::BrainStem2CLI::PortClass::setVconn2Enabled (*C++ function*), 1220
Acroname::BrainStem2CLI::PortClass::setVconnAccumulatedPower (*C++ function*), 1226
Acroname::BrainStem2CLI::PortClass::setVconnEnabled (*C++ function*), 1220
Acroname::BrainStem2CLI::PortClass::setVoltageSetpoint (*C++ function*), 1221
Acroname::BrainStem2CLI::PortMapping (*C++ class*), 1178
Acroname::BrainStem2CLI::PortMapping::~~PortMapping (*C++ function*), 1179
Acroname::BrainStem2CLI::PortMapping::lastError (*C++ member*), 1179
Acroname::BrainStem2CLI::PortMapping::PortMapping (*C++ function*), 1179
Acroname::BrainStem2CLI::PortMapping::update (*C++ function*), 1179
Acroname::BrainStem2CLI::PowerDeliveryClass (*C++ class*), 1230
Acroname::BrainStem2CLI::PowerDeliveryClass::~~PowerDeliveryClass (*C++ function*), 1230
Acroname::BrainStem2CLI::PowerDeliveryClass::getAttachTimeElapsed (*C++ function*), 1233
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableCurrentMax (*C++ function*), 1236
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableOrientation (*C++ function*), 1236
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableSpeedMax (*C++ function*), 1236
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableType (*C++ function*), 1236
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableVoltageMax (*C++ function*), 1235
Acroname::BrainStem2CLI::PowerDeliveryClass::getConnectionState (*C++ function*), 1230
Acroname::BrainStem2CLI::PowerDeliveryClass::getDataRoleCapabilities (*C++ function*), 1235
Acroname::BrainStem2CLI::PowerDeliveryClass::getFastRoleSwapCurrent (*C++ function*), 1239
Acroname::BrainStem2CLI::PowerDeliveryClass::getFlagMode (*C++ function*), 1238
Acroname::BrainStem2CLI::PowerDeliveryClass::getLinkState (*C++ function*), 1233
Acroname::BrainStem2CLI::PowerDeliveryClass::getNumberOfPowerDataObjects (*C++ function*), 1230
Acroname::BrainStem2CLI::PowerDeliveryClass::getOverride (*C++ function*), 1238
Acroname::BrainStem2CLI::PowerDeliveryClass::getPeakCurrentConfiguration (*C++ function*), 1239
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObject (*C++ function*), 1231
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectEnabled (*C++ function*), 1232
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectEnabledList (*C++ function*), 1233
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectList (*C++ function*), 1231
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerRole (*C++ function*), 1234
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerRoleCapabilities (*C++ function*), 1234
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerRolePreferred (*C++ function*), 1235
Acroname::BrainStem2CLI::PowerDeliveryClass::getProtocolRevisionMax (*C++ function*), 1238
Acroname::BrainStem2CLI::PowerDeliveryClass::getRequestDataObject (*C++ function*), 1233
Acroname::BrainStem2CLI::PowerDeliveryClass::packDataObjectAttributes (*C++ function*), 1240
Acroname::BrainStem2CLI::PowerDeliveryClass::PowerDeliveryClass (*C++ function*), 1230
Acroname::BrainStem2CLI::PowerDeliveryClass::request (*C++ function*), 1236
Acroname::BrainStem2CLI::PowerDeliveryClass::requestStatus (*C++ function*), 1237
Acroname::BrainStem2CLI::PowerDeliveryClass::resetPowerDataObjectToDefault (*C++ function*), 1231
Acroname::BrainStem2CLI::PowerDeliveryClass::setFastRoleSwapCurrent (*C++ function*), 1240
Acroname::BrainStem2CLI::PowerDeliveryClass::setFlagMode (*C++ function*), 1239
Acroname::BrainStem2CLI::PowerDeliveryClass::setOverride (*C++ function*), 1238
Acroname::BrainStem2CLI::PowerDeliveryClass::setPeakCurrentConfiguration (*C++ function*), 1239
Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerDataObject (*C++ function*), 1231
Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerDataObjectEnabled (*C++ function*), 1232
Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerRole (*C++ function*), 1234
Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerRolePreferred (*C++ function*), 1235
Acroname::BrainStem2CLI::PowerDeliveryClass::setProtocolRevisionMax (*C++ function*), 1238
Acroname::BrainStem2CLI::PowerDeliveryClass::setRequestDataObject (*C++ function*), 1233
Acroname::BrainStem2CLI::PowerDeliveryClass::unpackDataObjectAttributes (*C++ function*), 1240
Acroname::BrainStem2CLI::RailClass (*C++ class*), 1242
Acroname::BrainStem2CLI::RailClass::~~RailClass (*C++ function*), 1243
Acroname::BrainStem2CLI::RailClass::clearFaults (*C++ function*), 1248
Acroname::BrainStem2CLI::RailClass::getCurrent (*C++ function*), 1243
Acroname::BrainStem2CLI::RailClass::getCurrentLimit (*C++ function*), 1244
Acroname::BrainStem2CLI::RailClass::getCurrentSetpoint (*C++ function*), 1243
Acroname::BrainStem2CLI::RailClass::getEnable (*C++ function*), 1244
Acroname::BrainStem2CLI::RailClass::getKelvinSensingEnable (*C++ function*), 1247
Acroname::BrainStem2CLI::RailClass::getKelvinSensingState (*C++ function*), 1247
Acroname::BrainStem2CLI::RailClass::getOperationalMode (*C++ function*), 1248
Acroname::BrainStem2CLI::RailClass::getOperationalState (*C++ function*), 1248
Acroname::BrainStem2CLI::RailClass::getPower (*C++ function*), 1245
Acroname::BrainStem2CLI::RailClass::getPowerLimit (*C++ function*), 1246
Acroname::BrainStem2CLI::RailClass::getPowerSetpoint (*C++ function*), 1246
Acroname::BrainStem2CLI::RailClass::getResistance (*C++ function*), 1246
Acroname::BrainStem2CLI::RailClass::getResistanceSetpoint (*C++ function*), 1247
Acroname::BrainStem2CLI::RailClass::getTemperature (*C++ function*), 1244
Acroname::BrainStem2CLI::RailClass::getVoltage (*C++ function*), 1244

Acroname::BrainStem2CLI::RailClass::getVoltageMaxLimit (C++ function), 1245
 Acroname::BrainStem2CLI::RailClass::getVoltageMinLimit (C++ function), 1245
 Acroname::BrainStem2CLI::RailClass::getVoltageSetpoint (C++ function), 1245
 Acroname::BrainStem2CLI::RailClass::RailClass (C++ function), 1243
 Acroname::BrainStem2CLI::RailClass::setCurrentLimit (C++ function), 1243
 Acroname::BrainStem2CLI::RailClass::setCurrentSetpoint (C++ function), 1243
 Acroname::BrainStem2CLI::RailClass::setEnabled (C++ function), 1244
 Acroname::BrainStem2CLI::RailClass::setKelvinSensingEnable (C++ function), 1247
 Acroname::BrainStem2CLI::RailClass::setOperationalMode (C++ function), 1247
 Acroname::BrainStem2CLI::RailClass::setPowerLimit (C++ function), 1246
 Acroname::BrainStem2CLI::RailClass::setPowerSetpoint (C++ function), 1246
 Acroname::BrainStem2CLI::RailClass::setResistanceSetpoint (C++ function), 1247
 Acroname::BrainStem2CLI::RailClass::setVoltageMaxLimit (C++ function), 1245
 Acroname::BrainStem2CLI::RailClass::setVoltageMinLimit (C++ function), 1245
 Acroname::BrainStem2CLI::RailClass::setVoltageSetpoint (C++ function), 1244
 Acroname::BrainStem2CLI::RCServoClass (C++ class), 1241
 Acroname::BrainStem2CLI::RCServoClass::~RCServoClass (C++ function), 1241
 Acroname::BrainStem2CLI::RCServoClass::setEnabled (C++ function), 1241
 Acroname::BrainStem2CLI::RCServoClass::getPosition (C++ function), 1242
 Acroname::BrainStem2CLI::RCServoClass::getReverse (C++ function), 1242
 Acroname::BrainStem2CLI::RCServoClass::RCServoClass (C++ function), 1241
 Acroname::BrainStem2CLI::RCServoClass::setEnabled (C++ function), 1241
 Acroname::BrainStem2CLI::RCServoClass::setPosition (C++ function), 1242
 Acroname::BrainStem2CLI::RCServoClass::setReverse (C++ function), 1242
 Acroname::BrainStem2CLI::RelayClass (C++ class), 1248
 Acroname::BrainStem2CLI::RelayClass::~RelayClass (C++ function), 1248
 Acroname::BrainStem2CLI::RelayClass::setEnabled (C++ function), 1249
 Acroname::BrainStem2CLI::RelayClass::getVoltage (C++ function), 1249
 Acroname::BrainStem2CLI::RelayClass::RelayClass (C++ function), 1248
 Acroname::BrainStem2CLI::RelayClass::setEnabled (C++ function), 1249
 Acroname::BrainStem2CLI::SignalClass (C++ class), 1249
 Acroname::BrainStem2CLI::SignalClass::~SignalClass (C++ function), 1250
 Acroname::BrainStem2CLI::SignalClass::setEnabled (C++ function), 1250
 Acroname::BrainStem2CLI::SignalClass::getInvert (C++ function), 1250
 Acroname::BrainStem2CLI::SignalClass::getT2Time (C++ function), 1251
 Acroname::BrainStem2CLI::SignalClass::getT3Time (C++ function), 1251
 Acroname::BrainStem2CLI::SignalClass::setEnabled (C++ function), 1250
 Acroname::BrainStem2CLI::SignalClass::SignalClass (C++ function), 1250
 Acroname::BrainStem2CLI::SignalClass::setInvert (C++ function), 1250
 Acroname::BrainStem2CLI::SignalClass::setT2Time (C++ function), 1251
 Acroname::BrainStem2CLI::SignalClass::setT3Time (C++ function), 1250
 Acroname::BrainStem2CLI::SignalClass::SignalClass (C++ function), 1250
 Acroname::BrainStem2CLI::StoreClass (C++ class), 1251
 Acroname::BrainStem2CLI::StoreClass::~StoreClass (C++ function), 1252
 Acroname::BrainStem2CLI::StoreClass::getSlotCapacity (C++ function), 1253
 Acroname::BrainStem2CLI::StoreClass::getSlotLocked (C++ function), 1253
 Acroname::BrainStem2CLI::StoreClass::getSlotSize (C++ function), 1253
 Acroname::BrainStem2CLI::StoreClass::getSlotState (C++ function), 1252
 Acroname::BrainStem2CLI::StoreClass::loadSlot (C++ function), 1252
 Acroname::BrainStem2CLI::StoreClass::setSlotLocked (C++ function), 1253
 Acroname::BrainStem2CLI::StoreClass::slotDisable (C++ function), 1253
 Acroname::BrainStem2CLI::StoreClass::slotEnable (C++ function), 1253
 Acroname::BrainStem2CLI::StoreClass::StoreClass (C++ function), 1252
 Acroname::BrainStem2CLI::StoreClass::unloadSlot (C++ function), 1252
 Acroname::BrainStem2CLI::SystemClass (C++ class), 1254
 Acroname::BrainStem2CLI::SystemClass::~SystemClass (C++ function), 1254
 Acroname::BrainStem2CLI::SystemClass::getBootSlot (C++ function), 1256
 Acroname::BrainStem2CLI::SystemClass::getBuild (C++ function), 1256
 Acroname::BrainStem2CLI::SystemClass::getErrors (C++ function), 1262
 Acroname::BrainStem2CLI::SystemClass::getHardwareVersion (C++ function), 1257
 Acroname::BrainStem2CLI::SystemClass::getHBInterval (C++ function), 1255
 Acroname::BrainStem2CLI::SystemClass::getInputCurrent (C++ function), 1258
 Acroname::BrainStem2CLI::SystemClass::getInputPowerBehavior (C++ function), 1261
 Acroname::BrainStem2CLI::SystemClass::getInputPowerBehaviorConfig (C++ function), 1261
 Acroname::BrainStem2CLI::SystemClass::getInputPowerSource (C++ function), 1260
 Acroname::BrainStem2CLI::SystemClass::getInputVoltage (C++ function), 1258
 Acroname::BrainStem2CLI::SystemClass::getLED (C++ function), 1255
 Acroname::BrainStem2CLI::SystemClass::getLEDMaxBrightness (C++ function), 1256
 Acroname::BrainStem2CLI::SystemClass::getLinkInterface (C++ function), 1262

Acroname::BrainStem2CLI::SystemClass::getMaximumTemperature (C++ function), 1258
Acroname::BrainStem2CLI::SystemClass::getMinimumTemperature (C++ function), 1258
Acroname::BrainStem2CLI::SystemClass::getModel (C++ function), 1257
Acroname::BrainStem2CLI::SystemClass::getModule (C++ function), 1254
Acroname::BrainStem2CLI::SystemClass::getModuleBaseAddress (C++ function), 1254
Acroname::BrainStem2CLI::SystemClass::getModuleHardwareOffset (C++ function), 1258
Acroname::BrainStem2CLI::SystemClass::getModuleSoftwareOffset (C++ function), 1259
Acroname::BrainStem2CLI::SystemClass::getName (C++ function), 1261
Acroname::BrainStem2CLI::SystemClass::getPowerLimit (C++ function), 1259
Acroname::BrainStem2CLI::SystemClass::getPowerLimitMax (C++ function), 1260
Acroname::BrainStem2CLI::SystemClass::getPowerLimitState (C++ function), 1260
Acroname::BrainStem2CLI::SystemClass::getProtocolFeatures (C++ function), 1263
Acroname::BrainStem2CLI::SystemClass::getRouter (C++ function), 1255
Acroname::BrainStem2CLI::SystemClass::getRouterAddressSetting (C++ function), 1259
Acroname::BrainStem2CLI::SystemClass::getSerialNumber (C++ function), 1257
Acroname::BrainStem2CLI::SystemClass::getTemperature (C++ function), 1258
Acroname::BrainStem2CLI::SystemClass::getUnregulatedCurrent (C++ function), 1260
Acroname::BrainStem2CLI::SystemClass::getUnregulatedVoltage (C++ function), 1260
Acroname::BrainStem2CLI::SystemClass::getUptime (C++ function), 1258
Acroname::BrainStem2CLI::SystemClass::getVersion (C++ function), 1256
Acroname::BrainStem2CLI::SystemClass::logEvents (C++ function), 1257
Acroname::BrainStem2CLI::SystemClass::reset (C++ function), 1257
Acroname::BrainStem2CLI::SystemClass::resetDeviceToFactoryDefaults (C++ function), 1262
Acroname::BrainStem2CLI::SystemClass::routeToMe (C++ function), 1259
Acroname::BrainStem2CLI::SystemClass::save (C++ function), 1257
Acroname::BrainStem2CLI::SystemClass::setBootSlot (C++ function), 1256
Acroname::BrainStem2CLI::SystemClass::setHBInterval (C++ function), 1255
Acroname::BrainStem2CLI::SystemClass::setInputPowerBehavior (C++ function), 1261
Acroname::BrainStem2CLI::SystemClass::setInputPowerBehaviorConfig (C++ function), 1261
Acroname::BrainStem2CLI::SystemClass::setLED (C++ function), 1255
Acroname::BrainStem2CLI::SystemClass::setLEDMaxBrightness (C++ function), 1256
Acroname::BrainStem2CLI::SystemClass::setLinkInterface (C++ function), 1262
Acroname::BrainStem2CLI::SystemClass::setModuleSoftwareOffset (C++ function), 1259
Acroname::BrainStem2CLI::SystemClass::setName (C++ function), 1262
Acroname::BrainStem2CLI::SystemClass::setPowerLimitMax (C++ function), 1260
Acroname::BrainStem2CLI::SystemClass::setRouter (C++ function), 1254
Acroname::BrainStem2CLI::SystemClass::SystemClass (C++ function), 1254
Acroname::BrainStem2CLI::TemperatureClass (C++ class), 1263
Acroname::BrainStem2CLI::TemperatureClass::~TemperatureClass (C++ function), 1263
Acroname::BrainStem2CLI::TemperatureClass::getValue (C++ function), 1263
Acroname::BrainStem2CLI::TemperatureClass::getValueMax (C++ function), 1264
Acroname::BrainStem2CLI::TemperatureClass::getValueMin (C++ function), 1263
Acroname::BrainStem2CLI::TemperatureClass::TemperatureClass (C++ function), 1263
Acroname::BrainStem2CLI::TimerClass (C++ class), 1264
Acroname::BrainStem2CLI::TimerClass::~TimerClass (C++ function), 1264
Acroname::BrainStem2CLI::TimerClass::getExpiration (C++ function), 1264
Acroname::BrainStem2CLI::TimerClass::getMode (C++ function), 1265
Acroname::BrainStem2CLI::TimerClass::TimerClass::setExpiration (C++ function), 1264
Acroname::BrainStem2CLI::TimerClass::setMode (C++ function), 1265
Acroname::BrainStem2CLI::TimerClass::TimerClass (C++ function), 1264
Acroname::BrainStem2CLI::UARTClass (C++ class), 1265
Acroname::BrainStem2CLI::UARTClass::~UARTClass (C++ function), 1265
Acroname::BrainStem2CLI::UARTClass::getAvailableProtocols (C++ function), 1269
Acroname::BrainStem2CLI::UARTClass::getBaudRate (C++ function), 1266
Acroname::BrainStem2CLI::UARTClass::getCapableProtocols (C++ function), 1269
Acroname::BrainStem2CLI::UARTClass::getDataBits (C++ function), 1268
Acroname::BrainStem2CLI::UARTClass::UARTClass::getEnable (C++ function), 1266
Acroname::BrainStem2CLI::UARTClass::getFlowControl (C++ function), 1268
Acroname::BrainStem2CLI::UARTClass::getLinkChannel (C++ function), 1267
Acroname::BrainStem2CLI::UARTClass::getParity (C++ function), 1267
Acroname::BrainStem2CLI::UARTClass::getProtocol (C++ function), 1266
Acroname::BrainStem2CLI::UARTClass::UARTClass::getStopBits (C++ function), 1267
Acroname::BrainStem2CLI::UARTClass::setBaudRate (C++ function), 1266
Acroname::BrainStem2CLI::UARTClass::setDataBits (C++ function), 1268
Acroname::BrainStem2CLI::UARTClass::setEnable (C++ function), 1266
Acroname::BrainStem2CLI::UARTClass::setFlowControl (C++ function), 1268
Acroname::BrainStem2CLI::UARTClass::setLinkChannel (C++ function), 1266
Acroname::BrainStem2CLI::UARTClass::setParity (C++ function), 1267

Acroname::BrainStem2CLI::UARTClass::setProtocol (*C++ function*), 1266
 Acroname::BrainStem2CLI::UARTClass::setStopBits (*C++ function*), 1267
 Acroname::BrainStem2CLI::UARTClass::UARTClass (*C++ function*), 1265
 Acroname::BrainStem2CLI::USBClass (*C++ class*), 1269
 Acroname::BrainStem2CLI::USBClass::~USBClass (*C++ function*), 1269
 Acroname::BrainStem2CLI::USBClass::clearPortErrorStatus (*C++ function*), 1272
 Acroname::BrainStem2CLI::USBClass::getAltModeConfig (*C++ function*), 1278
 Acroname::BrainStem2CLI::USBClass::getCableFlip (*C++ function*), 1278
 Acroname::BrainStem2CLI::USBClass::getCC1Current (*C++ function*), 1276
 Acroname::BrainStem2CLI::USBClass::getCC1Enable (*C++ function*), 1276
 Acroname::BrainStem2CLI::USBClass::getCC1Voltage (*C++ function*), 1277
 Acroname::BrainStem2CLI::USBClass::getCC2Current (*C++ function*), 1276
 Acroname::BrainStem2CLI::USBClass::getCC2Enable (*C++ function*), 1276
 Acroname::BrainStem2CLI::USBClass::getCC2Voltage (*C++ function*), 1277
 Acroname::BrainStem2CLI::USBClass::getConnectMode (*C++ function*), 1275
 Acroname::BrainStem2CLI::USBClass::getDownstreamBoostMode (*C++ function*), 1274
 Acroname::BrainStem2CLI::USBClass::getDownstreamDataSpeed (*C++ function*), 1275
 Acroname::BrainStem2CLI::USBClass::getEnumerationDelay (*C++ function*), 1272
 Acroname::BrainStem2CLI::USBClass::getHubMode (*C++ function*), 1271
 Acroname::BrainStem2CLI::USBClass::getPortCurrent (*C++ function*), 1271
 Acroname::BrainStem2CLI::USBClass::getPortCurrentLimit (*C++ function*), 1273
 Acroname::BrainStem2CLI::USBClass::getPortError (*C++ function*), 1274
 Acroname::BrainStem2CLI::USBClass::getPortMode (*C++ function*), 1273
 Acroname::BrainStem2CLI::USBClass::getPortState (*C++ function*), 1273
 Acroname::BrainStem2CLI::USBClass::getPortVoltage (*C++ function*), 1271
 Acroname::BrainStem2CLI::USBClass::getSBU1Voltage (*C++ function*), 1278
 Acroname::BrainStem2CLI::USBClass::getSBU2Voltage (*C++ function*), 1278
 Acroname::BrainStem2CLI::USBClass::getSBUEnable (*C++ function*), 1277
 Acroname::BrainStem2CLI::USBClass::getUpstreamBoostMode (*C++ function*), 1274
 Acroname::BrainStem2CLI::USBClass::getUpstreamMode (*C++ function*), 1272
 Acroname::BrainStem2CLI::USBClass::getUpstreamState (*C++ function*), 1272
 Acroname::BrainStem2CLI::USBClass::setAltModeConfig (*C++ function*), 1278
 Acroname::BrainStem2CLI::USBClass::setCableFlip (*C++ function*), 1277
 Acroname::BrainStem2CLI::USBClass::setCC1Enable (*C++ function*), 1275
 Acroname::BrainStem2CLI::USBClass::setCC2Enable (*C++ function*), 1276
 Acroname::BrainStem2CLI::USBClass::setConnectMode (*C++ function*), 1275
 Acroname::BrainStem2CLI::USBClass::setDataDisable (*C++ function*), 1270
 Acroname::BrainStem2CLI::USBClass::setDataEnable (*C++ function*), 1270
 Acroname::BrainStem2CLI::USBClass::setDownstreamBoostMode (*C++ function*), 1274
 Acroname::BrainStem2CLI::USBClass::setEnumerationDelay (*C++ function*), 1272
 Acroname::BrainStem2CLI::USBClass::setHiSpeedDataDisable (*C++ function*), 1270
 Acroname::BrainStem2CLI::USBClass::setHiSpeedDataEnable (*C++ function*), 1270
 Acroname::BrainStem2CLI::USBClass::setHubMode (*C++ function*), 1272
 Acroname::BrainStem2CLI::USBClass::setPortCurrentLimit (*C++ function*), 1273
 Acroname::BrainStem2CLI::USBClass::setPortDisable (*C++ function*), 1270
 Acroname::BrainStem2CLI::USBClass::setPortEnable (*C++ function*), 1269
 Acroname::BrainStem2CLI::USBClass::setPortMode (*C++ function*), 1273
 Acroname::BrainStem2CLI::USBClass::setPowerDisable (*C++ function*), 1271
 Acroname::BrainStem2CLI::USBClass::setPowerEnable (*C++ function*), 1271
 Acroname::BrainStem2CLI::USBClass::setSBUEnable (*C++ function*), 1277
 Acroname::BrainStem2CLI::USBClass::setSuperSpeedDataDisable (*C++ function*), 1271
 Acroname::BrainStem2CLI::USBClass::setSuperSpeedDataEnable (*C++ function*), 1270
 Acroname::BrainStem2CLI::USBClass::setUpstreamBoostMode (*C++ function*), 1274
 Acroname::BrainStem2CLI::USBClass::setUpstreamMode (*C++ function*), 1272
 Acroname::BrainStem2CLI::USBClass::USBClass (*C++ function*), 1269
 Acroname::BrainStem2CLI::USBSystemClass (*C++ class*), 1279
 Acroname::BrainStem2CLI::USBSystemClass::~USBSystemClass (*C++ function*), 1279
 Acroname::BrainStem2CLI::USBSystemClass::getDataHSMMaxDataRate (*C++ function*), 1284
 Acroname::BrainStem2CLI::USBSystemClass::getDataRoleBehavior (*C++ function*), 1282
 Acroname::BrainStem2CLI::USBSystemClass::getDataRoleBehaviorConfig (*C++ function*), 1282
 Acroname::BrainStem2CLI::USBSystemClass::getDataRoleList (*C++ function*), 1280
 Acroname::BrainStem2CLI::USBSystemClass::getDataSSMaxDataRate (*C++ function*), 1284
 Acroname::BrainStem2CLI::USBSystemClass::getEnabledList (*C++ function*), 1280
 Acroname::BrainStem2CLI::USBSystemClass::getEnumerationDelay (*C++ function*), 1280
 Acroname::BrainStem2CLI::USBSystemClass::getModeList (*C++ function*), 1280
 Acroname::BrainStem2CLI::USBSystemClass::getOverride (*C++ function*), 1283
 Acroname::BrainStem2CLI::USBSystemClass::getPowerBehavior (*C++ function*), 1281
 Acroname::BrainStem2CLI::USBSystemClass::getPowerBehaviorConfig (*C++ function*), 1281

Acroname::BrainStem2CLI::USBSystemClass::getSelectorMode (C++ function), 1283
Acroname::BrainStem2CLI::USBSystemClass::getStateList (C++ function), 1281
Acroname::BrainStem2CLI::USBSystemClass::getUpstream (C++ function), 1279
Acroname::BrainStem2CLI::USBSystemClass::getUpstreamHS (C++ function), 1283
Acroname::BrainStem2CLI::USBSystemClass::getUpstreamSS (C++ function), 1283
Acroname::BrainStem2CLI::USBSystemClass::setDataHSMaxDataRate (C++ function), 1284
Acroname::BrainStem2CLI::USBSystemClass::setDataRoleBehavior (C++ function), 1282
Acroname::BrainStem2CLI::USBSystemClass::setDataRoleBehaviorConfig (C++ function), 1282
Acroname::BrainStem2CLI::USBSystemClass::setDataSSMaxDataRate (C++ function), 1284
Acroname::BrainStem2CLI::USBSystemClass::setEnabledList (C++ function), 1280
Acroname::BrainStem2CLI::USBSystemClass::setEnumerationDelay (C++ function), 1280
Acroname::BrainStem2CLI::USBSystemClass::setModeList (C++ function), 1281
Acroname::BrainStem2CLI::USBSystemClass::setOverride (C++ function), 1284
Acroname::BrainStem2CLI::USBSystemClass::setPowerBehavior (C++ function), 1281
Acroname::BrainStem2CLI::USBSystemClass::setPowerBehaviorConfig (C++ function), 1282
Acroname::BrainStem2CLI::USBSystemClass::setSelectorMode (C++ function), 1283
Acroname::BrainStem2CLI::USBSystemClass::setUpstream (C++ function), 1279
Acroname::BrainStem2CLI::USBSystemClass::setUpstreamHS (C++ function), 1283
Acroname::BrainStem2CLI::USBSystemClass::setUpstreamSS (C++ function), 1283
Acroname::BrainStem2CLI::USBSystemClass::USBSystemClass (C++ function), 1279
Acroname::BrainStem::AnalogClass (C++ class), 759
Acroname::BrainStem::AnalogClass::~AnalogClass (C++ function), 760
Acroname::BrainStem::AnalogClass::AnalogClass (C++ function), 760
Acroname::BrainStem::AnalogClass::getBulkCaptureNumberOfSamples (C++ function), 762
Acroname::BrainStem::AnalogClass::getBulkCaptureSampleRate (C++ function), 762
Acroname::BrainStem::AnalogClass::getBulkCaptureState (C++ function), 763
Acroname::BrainStem::AnalogClass::getConfiguration (C++ function), 762
Acroname::BrainStem::AnalogClass::getEnable (C++ function), 760
Acroname::BrainStem::AnalogClass::getRange (C++ function), 760
Acroname::BrainStem::AnalogClass::getValue (C++ function), 760
Acroname::BrainStem::AnalogClass::getVoltage (C++ function), 760
Acroname::BrainStem::AnalogClass::init (C++ function), 760
Acroname::BrainStem::AnalogClass::initiateBulkCapture (C++ function), 762
Acroname::BrainStem::AnalogClass::setBulkCaptureNumberOfSamples (C++ function), 762
Acroname::BrainStem::AnalogClass::setBulkCaptureSampleRate (C++ function), 762
Acroname::BrainStem::AnalogClass::setConfiguration (C++ function), 761
Acroname::BrainStem::AnalogClass::setEnable (C++ function), 761
Acroname::BrainStem::AnalogClass::setRange (C++ function), 761
Acroname::BrainStem::AnalogClass::setValue (C++ function), 761
Acroname::BrainStem::AnalogClass::setVoltage (C++ function), 761
Acroname::BrainStem::AppClass (C++ class), 763
Acroname::BrainStem::AppClass::~AppClass (C++ function), 763
Acroname::BrainStem::AppClass::AppClass (C++ function), 763
Acroname::BrainStem::AppClass::execute (C++ function), 763, 764
Acroname::BrainStem::AppClass::init (C++ function), 763
Acroname::BrainStem::ClockClass (C++ class), 764
Acroname::BrainStem::ClockClass::~ClockClass (C++ function), 765
Acroname::BrainStem::ClockClass::ClockClass (C++ function), 765
Acroname::BrainStem::ClockClass::getDay (C++ function), 765
Acroname::BrainStem::ClockClass::getHour (C++ function), 766
Acroname::BrainStem::ClockClass::getMinute (C++ function), 766
Acroname::BrainStem::ClockClass::getMonth (C++ function), 765
Acroname::BrainStem::ClockClass::getSecond (C++ function), 766
Acroname::BrainStem::ClockClass::getYear (C++ function), 765
Acroname::BrainStem::ClockClass::init (C++ function), 765
Acroname::BrainStem::ClockClass::setDay (C++ function), 765
Acroname::BrainStem::ClockClass::setHour (C++ function), 766
Acroname::BrainStem::ClockClass::setMinute (C++ function), 766
Acroname::BrainStem::ClockClass::setMonth (C++ function), 765
Acroname::BrainStem::ClockClass::setSecond (C++ function), 766
Acroname::BrainStem::ClockClass::setYear (C++ function), 765
Acroname::BrainStem::DigitalClass (C++ class), 767
Acroname::BrainStem::DigitalClass::~DigitalClass (C++ function), 767
Acroname::BrainStem::DigitalClass::DigitalClass (C++ function), 767
Acroname::BrainStem::DigitalClass::getConfiguration (C++ function), 767
Acroname::BrainStem::DigitalClass::getLinkChannel (C++ function), 769
Acroname::BrainStem::DigitalClass::getState (C++ function), 768
Acroname::BrainStem::DigitalClass::getStateAll (C++ function), 768

Acroname::BrainStem::DigitalClass::getValue (C++ function), 768
 Acroname::BrainStem::DigitalClass::init (C++ function), 767
 Acroname::BrainStem::DigitalClass::setConfiguration (C++ function), 767
 Acroname::BrainStem::DigitalClass::setLinkChannel (C++ function), 769
 Acroname::BrainStem::DigitalClass::setState (C++ function), 768
 Acroname::BrainStem::DigitalClass::setStateAll (C++ function), 768
 Acroname::BrainStem::DigitalClass::setValue (C++ function), 768
 Acroname::BrainStem::EntityClass (C++ class), 754
 Acroname::BrainStem::EntityClass::~EntityClass (C++ function), 755
 Acroname::BrainStem::EntityClass::callUEI (C++ function), 755
 Acroname::BrainStem::EntityClass::drainUEI (C++ function), 757
 Acroname::BrainStem::EntityClass::EntityClass (C++ function), 755
 Acroname::BrainStem::EntityClass::getIndex (C++ function), 757
 Acroname::BrainStem::EntityClass::getStreamStatus (C++ function), 758
 Acroname::BrainStem::EntityClass::getUEI16 (C++ function), 756
 Acroname::BrainStem::EntityClass::getUEI32 (C++ function), 757
 Acroname::BrainStem::EntityClass::getUEI8 (C++ function), 755
 Acroname::BrainStem::EntityClass::getUEIBytes (C++ function), 757
 Acroname::BrainStem::EntityClass::getUEIBytesCheck (C++ function), 757
 Acroname::BrainStem::EntityClass::getUEIBytesContinue (C++ function), 759
 Acroname::BrainStem::EntityClass::getUEIBytesSequence (C++ function), 759
 Acroname::BrainStem::EntityClass::impl (C++ class), 759
 Acroname::BrainStem::EntityClass::init (C++ function), 755
 Acroname::BrainStem::EntityClass::loadEntityFromStore (C++ function), 758
 Acroname::BrainStem::EntityClass::registerOptionCallback (C++ function), 758
 Acroname::BrainStem::EntityClass::resetEntityToFactoryDefaults (C++ function), 758
 Acroname::BrainStem::EntityClass::saveEntityToStore (C++ function), 758
 Acroname::BrainStem::EntityClass::setStreamEnabled (C++ function), 758
 Acroname::BrainStem::EntityClass::setUEI16 (C++ function), 756
 Acroname::BrainStem::EntityClass::setUEI32 (C++ function), 756
 Acroname::BrainStem::EntityClass::setUEI8 (C++ function), 755
 Acroname::BrainStem::EntityClass::setUEIBytes (C++ function), 757
 Acroname::BrainStem::EntityClass::sUEIBytesFilter (C++ function), 759
 Acroname::BrainStem::EqualizerClass (C++ class), 769
 Acroname::BrainStem::EqualizerClass::~EqualizerClass (C++ function), 769
 Acroname::BrainStem::EqualizerClass::EqualizerClass (C++ function), 769
 Acroname::BrainStem::EqualizerClass::getReceiverConfig (C++ function), 770
 Acroname::BrainStem::EqualizerClass::getTransmitterConfig (C++ function), 770
 Acroname::BrainStem::EqualizerClass::init (C++ function), 769
 Acroname::BrainStem::EqualizerClass::setReceiverConfig (C++ function), 769
 Acroname::BrainStem::EqualizerClass::setTransmitterConfig (C++ function), 770
 Acroname::BrainStem::EthernetClass (C++ class), 770
 Acroname::BrainStem::EthernetClass::~EthernetClass (C++ function), 770
 Acroname::BrainStem::EthernetClass::EthernetClass (C++ function), 770
 Acroname::BrainStem::EthernetClass::getEnabled (C++ function), 771
 Acroname::BrainStem::EthernetClass::getHostname (C++ function), 774
 Acroname::BrainStem::EthernetClass::getInterfacePort (C++ function), 775
 Acroname::BrainStem::EthernetClass::getIPv4Address (C++ function), 773
 Acroname::BrainStem::EthernetClass::getIPv4DNSAddress (C++ function), 774
 Acroname::BrainStem::EthernetClass::getIPv4Gateway (C++ function), 773
 Acroname::BrainStem::EthernetClass::getIPv4Netmask (C++ function), 773
 Acroname::BrainStem::EthernetClass::getMACAddress (C++ function), 774
 Acroname::BrainStem::EthernetClass::getNetworkConfiguration (C++ function), 771
 Acroname::BrainStem::EthernetClass::getStaticIPv4Address (C++ function), 771
 Acroname::BrainStem::EthernetClass::getStaticIPv4DNSAddress (C++ function), 773
 Acroname::BrainStem::EthernetClass::getStaticIPv4Gateway (C++ function), 772
 Acroname::BrainStem::EthernetClass::getStaticIPv4Netmask (C++ function), 772
 Acroname::BrainStem::EthernetClass::init (C++ function), 770
 Acroname::BrainStem::EthernetClass::setEnabled (C++ function), 770
 Acroname::BrainStem::EthernetClass::setHostname (C++ function), 774
 Acroname::BrainStem::EthernetClass::setInterfacePort (C++ function), 774
 Acroname::BrainStem::EthernetClass::setNetworkConfiguration (C++ function), 771
 Acroname::BrainStem::EthernetClass::setStaticIPv4Address (C++ function), 771
 Acroname::BrainStem::EthernetClass::setStaticIPv4DNSAddress (C++ function), 773
 Acroname::BrainStem::EthernetClass::setStaticIPv4Gateway (C++ function), 772
 Acroname::BrainStem::EthernetClass::setStaticIPv4Netmask (C++ function), 772
 Acroname::BrainStem::HDBaseTClass (C++ class), 775
 Acroname::BrainStem::HDBaseTClass::~HDBaseTClass (C++ function), 775

Acroname::BrainStem::HDBaseTClass::getCableLength (C++ function), 776
Acroname::BrainStem::HDBaseTClass::getEncodingState (C++ function), 776
Acroname::BrainStem::HDBaseTClass::getFirmwareVersion (C++ function), 775
Acroname::BrainStem::HDBaseTClass::getLinkRole (C++ function), 777
Acroname::BrainStem::HDBaseTClass::getLinkUtilization (C++ function), 776
Acroname::BrainStem::HDBaseTClass::getMSEA (C++ function), 776
Acroname::BrainStem::HDBaseTClass::getMSEB (C++ function), 776
Acroname::BrainStem::HDBaseTClass::getRetransmissionRate (C++ function), 776
Acroname::BrainStem::HDBaseTClass::getSerialNumber (C++ function), 775
Acroname::BrainStem::HDBaseTClass::getState (C++ function), 776
Acroname::BrainStem::HDBaseTClass::getUSB2DeviceTree (C++ function), 777
Acroname::BrainStem::HDBaseTClass::getUSB3DeviceTree (C++ function), 777
Acroname::BrainStem::HDBaseTClass::HDBaseTClass (C++ function), 775
Acroname::BrainStem::HDBaseTClass::init (C++ function), 775
Acroname::BrainStem::HDBaseTClass::setLinkRole (C++ function), 777
Acroname::BrainStem::I2CClass (C++ class), 777
Acroname::BrainStem::I2CClass::~I2CClass (C++ function), 778
Acroname::BrainStem::I2CClass::getSpeed (C++ function), 779
Acroname::BrainStem::I2CClass::I2CClass (C++ function), 778
Acroname::BrainStem::I2CClass::init (C++ function), 778
Acroname::BrainStem::I2CClass::read (C++ function), 778
Acroname::BrainStem::I2CClass::setPullup (C++ function), 778
Acroname::BrainStem::I2CClass::setSpeed (C++ function), 778
Acroname::BrainStem::I2CClass::write (C++ function), 778
Acroname::BrainStem::Link (C++ class), 856
Acroname::BrainStem::Link::~Link (C++ function), 858
Acroname::BrainStem::Link::connect (C++ function), 859
Acroname::BrainStem::Link::connectThroughLinkModule (C++ function), 859
Acroname::BrainStem::Link::disablePacketLog (C++ function), 865
Acroname::BrainStem::Link::disconnect (C++ function), 859
Acroname::BrainStem::Link::discoverAndConnect (C++ function), 858
Acroname::BrainStem::Link::dropMatchingUEIPackets (C++ function), 861
Acroname::BrainStem::Link::enablePacketLog (C++ function), 865
Acroname::BrainStem::Link::enablePooledPackets (C++ function), 865
Acroname::BrainStem::Link::enableStream (C++ function), 863
Acroname::BrainStem::Link::filterActiveStreamKeys (C++ function), 865
Acroname::BrainStem::Link::getConfig (C++ function), 858
Acroname::BrainStem::Link::getFactoryData (C++ function), 865
Acroname::BrainStem::Link::getLinkSpecifier (C++ function), 860
Acroname::BrainStem::Link::getModuleAddress (C++ function), 860
Acroname::BrainStem::Link::getName (C++ function), 859
Acroname::BrainStem::Link::getStatus (C++ function), 859
Acroname::BrainStem::Link::getStreamKeyElement (C++ function), 867
Acroname::BrainStem::Link::getStreamPacketType (C++ function), 866
Acroname::BrainStem::Link::getStreamSample (C++ function), 867
Acroname::BrainStem::Link::getStreamStatus (C++ function), 864
Acroname::BrainStem::Link::getStreamValue (C++ function), 864
Acroname::BrainStem::Link::getTimestampParts (C++ function), 867
Acroname::BrainStem::Link::impl (C++ class), 868
Acroname::BrainStem::Link::isConnected (C++ function), 859
Acroname::BrainStem::Link::isLinkStreaming (C++ function), 864
Acroname::BrainStem::Link::isStreamPacket (C++ function), 867
Acroname::BrainStem::Link::isStreamSample (C++ function), 867
Acroname::BrainStem::Link::isSubindexType (C++ function), 867
Acroname::BrainStem::Link::Link (C++ function), 858
Acroname::BrainStem::Link::linkStreamFilter (C++ function), 868
Acroname::BrainStem::Link::loadStoreSlot (C++ function), 862
Acroname::BrainStem::Link::receivePacket (C++ function), 862
Acroname::BrainStem::Link::receiveUEI (C++ function), 860, 861
Acroname::BrainStem::Link::registerStreamCallback (C++ function), 864
Acroname::BrainStem::Link::reset (C++ function), 859
Acroname::BrainStem::Link::sDiscover (C++ function), 866
Acroname::BrainStem::Link::sendPacket (C++ function), 861
Acroname::BrainStem::Link::sendUEI (C++ function), 860
Acroname::BrainStem::Link::setConfig (C++ function), 858
Acroname::BrainStem::Link::setFactoryData (C++ function), 865
Acroname::BrainStem::Link::setLinkSpecifier (C++ function), 860
Acroname::BrainStem::Link::sFindAll (C++ function), 866

Acroname::BrainStem::Link::storeSlotCapacity (C++ function), 863
 Acroname::BrainStem::Link::storeSlotSize (C++ function), 863
 Acroname::BrainStem::Link::STREAM_KEY (C++ enum), 857
 Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_CMD (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_INDEX (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_LAST (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_MODULE_ADDRESS (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_OPTION (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_SUBINDEX (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_KEY_t (C++ type), 858
 Acroname::BrainStem::Link::STREAM_PACKET (C++ enum), 856
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_BYTES (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_LAST (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_SUBINDEX_U16 (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_SUBINDEX_U32 (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_SUBINDEX_U8 (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_U16 (C++ enumerator), 856
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_U32 (C++ enumerator), 857
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_U8 (C++ enumerator), 856
 Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_UNKNOWN (C++ enumerator), 856
 Acroname::BrainStem::Link::STREAM_PACKET_t (C++ type), 857
 Acroname::BrainStem::Link::streamCallback_t (C++ type), 857
 Acroname::BrainStem::Link::StreamStatusEntry (C++ struct), 868
 Acroname::BrainStem::Link::StreamStatusEntry::key (C++ member), 868
 Acroname::BrainStem::Link::StreamStatusEntry::value (C++ member), 868
 Acroname::BrainStem::Link::StreamStatusEntry_t (C++ type), 857
 Acroname::BrainStem::Link::unloadStoreSlot (C++ function), 862
 Acroname::BrainStem::Module (C++ class), 868
 Acroname::BrainStem::Module::~Module (C++ function), 869
 Acroname::BrainStem::Module::classQuantity (C++ function), 871
 Acroname::BrainStem::Module::connect (C++ function), 869
 Acroname::BrainStem::Module::connectFromSpec (C++ function), 869
 Acroname::BrainStem::Module::connectThroughLinkModule (C++ function), 869
 Acroname::BrainStem::Module::debug (C++ function), 872
 Acroname::BrainStem::Module::disconnect (C++ function), 870
 Acroname::BrainStem::Module::discoverAndConnect (C++ function), 869
 Acroname::BrainStem::Module::entityGroup (C++ function), 872
 Acroname::BrainStem::Module::getBuild (C++ function), 871
 Acroname::BrainStem::Module::getConfig (C++ function), 870
 Acroname::BrainStem::Module::getLink (C++ function), 870
 Acroname::BrainStem::Module::getLinkSpecifier (C++ function), 871
 Acroname::BrainStem::Module::getModuleAddress (C++ function), 870
 Acroname::BrainStem::Module::getStatus (C++ function), 870
 Acroname::BrainStem::Module::hasUEI (C++ function), 871
 Acroname::BrainStem::Module::isConnected (C++ function), 870
 Acroname::BrainStem::Module::Module (C++ function), 869
 Acroname::BrainStem::Module::reconnect (C++ function), 870
 Acroname::BrainStem::Module::setConfig (C++ function), 870
 Acroname::BrainStem::Module::setModuleAddress (C++ function), 871
 Acroname::BrainStem::Module::setNetworkingMode (C++ function), 872
 Acroname::BrainStem::Module::subClassQuantity (C++ function), 872
 Acroname::BrainStem::MuxClass (C++ class), 779
 Acroname::BrainStem::MuxClass::~MuxClass (C++ function), 779
 Acroname::BrainStem::MuxClass::getChannel (C++ function), 780
 Acroname::BrainStem::MuxClass::getChannelVoltage (C++ function), 780
 Acroname::BrainStem::MuxClass::getConfiguration (C++ function), 780
 Acroname::BrainStem::MuxClass::getEnable (C++ function), 779
 Acroname::BrainStem::MuxClass::getSplitMode (C++ function), 781
 Acroname::BrainStem::MuxClass::init (C++ function), 779
 Acroname::BrainStem::MuxClass::MuxClass (C++ function), 779
 Acroname::BrainStem::MuxClass::setChannel (C++ function), 780
 Acroname::BrainStem::MuxClass::setConfiguration (C++ function), 780
 Acroname::BrainStem::MuxClass::setEnable (C++ function), 779
 Acroname::BrainStem::MuxClass::setSplitMode (C++ function), 781
 Acroname::BrainStem::PoEClass (C++ class), 781
 Acroname::BrainStem::PoEClass::~PoEClass (C++ function), 781
 Acroname::BrainStem::PoEClass::getPairAccumulatedPower (C++ function), 785
 Acroname::BrainStem::PoEClass::getPairCapacitance (C++ function), 784

Acroname::BrainStem::PoEClass::getPairCurrent (*C++ function*), 784
Acroname::BrainStem::PoEClass::getPairDetectionStatus (*C++ function*), 783
Acroname::BrainStem::PoEClass::getPairDiscoveredClass (*C++ function*), 783
Acroname::BrainStem::PoEClass::getPairEnabled (*C++ function*), 781
Acroname::BrainStem::PoEClass::getPairPower (*C++ function*), 784
Acroname::BrainStem::PoEClass::getPairRequestedClass (*C++ function*), 783
Acroname::BrainStem::PoEClass::getPairResistance (*C++ function*), 784
Acroname::BrainStem::PoEClass::getPairSourcingClass (*C++ function*), 782
Acroname::BrainStem::PoEClass::getPairVoltage (*C++ function*), 783
Acroname::BrainStem::PoEClass::getPowerMode (*C++ function*), 782
Acroname::BrainStem::PoEClass::getPowerState (*C++ function*), 782
Acroname::BrainStem::PoEClass::getTotalAccumulatedPower (*C++ function*), 785
Acroname::BrainStem::PoEClass::getTotalPower (*C++ function*), 785
Acroname::BrainStem::PoEClass::init (*C++ function*), 781
Acroname::BrainStem::PoEClass::PoEClass (*C++ function*), 781
Acroname::BrainStem::PoEClass::setPairAccumulatedPower (*C++ function*), 785
Acroname::BrainStem::PoEClass::setPairEnabled (*C++ function*), 782
Acroname::BrainStem::PoEClass::setPairSourcingClass (*C++ function*), 782
Acroname::BrainStem::PoEClass::setPowerMode (*C++ function*), 782
Acroname::BrainStem::PoEClass::setTotalAccumulatedPower (*C++ function*), 786
Acroname::BrainStem::PointerClass (*C++ class*), 786
Acroname::BrainStem::PointerClass::~~PointerClass (*C++ function*), 786
Acroname::BrainStem::PointerClass::getChar (*C++ function*), 788
Acroname::BrainStem::PointerClass::getInt (*C++ function*), 788
Acroname::BrainStem::PointerClass::getMode (*C++ function*), 787
Acroname::BrainStem::PointerClass::getOffset (*C++ function*), 786
Acroname::BrainStem::PointerClass::getShort (*C++ function*), 788
Acroname::BrainStem::PointerClass::getTransferStore (*C++ function*), 787
Acroname::BrainStem::PointerClass::init (*C++ function*), 786
Acroname::BrainStem::PointerClass::initiateTransferFromStore (*C++ function*), 787
Acroname::BrainStem::PointerClass::initiateTransferToStore (*C++ function*), 787
Acroname::BrainStem::PointerClass::PointerClass (*C++ function*), 786
Acroname::BrainStem::PointerClass::setChar (*C++ function*), 788
Acroname::BrainStem::PointerClass::setInt (*C++ function*), 788
Acroname::BrainStem::PointerClass::setMode (*C++ function*), 787
Acroname::BrainStem::PointerClass::setOffset (*C++ function*), 786
Acroname::BrainStem::PointerClass::setShort (*C++ function*), 788
Acroname::BrainStem::PointerClass::setTransferStore (*C++ function*), 787
Acroname::BrainStem::PortClass (*C++ class*), 789
Acroname::BrainStem::PortClass::~~PortClass (*C++ function*), 789
Acroname::BrainStem::PortClass::getAllocatedPower (*C++ function*), 796
Acroname::BrainStem::PortClass::getAvailablePower (*C++ function*), 796
Acroname::BrainStem::PortClass::getCC1AccumulatedPower (*C++ function*), 802
Acroname::BrainStem::PortClass::getCC1Current (*C++ function*), 802
Acroname::BrainStem::PortClass::getCC1Enabled (*C++ function*), 794
Acroname::BrainStem::PortClass::getCC1State (*C++ function*), 800
Acroname::BrainStem::PortClass::getCC1Voltage (*C++ function*), 801
Acroname::BrainStem::PortClass::getCC2AccumulatedPower (*C++ function*), 802
Acroname::BrainStem::PortClass::getCC2Current (*C++ function*), 802
Acroname::BrainStem::PortClass::getCC2Enabled (*C++ function*), 794
Acroname::BrainStem::PortClass::getCC2State (*C++ function*), 801
Acroname::BrainStem::PortClass::getCC2Voltage (*C++ function*), 801
Acroname::BrainStem::PortClass::getCCCurrentLimit (*C++ function*), 797
Acroname::BrainStem::PortClass::getCCEnabled (*C++ function*), 794
Acroname::BrainStem::PortClass::getCurrentLimit (*C++ function*), 795
Acroname::BrainStem::PortClass::getCurrentLimitMode (*C++ function*), 796
Acroname::BrainStem::PortClass::getDataEnabled (*C++ function*), 790
Acroname::BrainStem::PortClass::getDataHS1Enabled (*C++ function*), 791
Acroname::BrainStem::PortClass::getDataHS2Enabled (*C++ function*), 791
Acroname::BrainStem::PortClass::getDataHSEnabled (*C++ function*), 790
Acroname::BrainStem::PortClass::getDataHSRoutingBehavior (*C++ function*), 798
Acroname::BrainStem::PortClass::getDataRole (*C++ function*), 793
Acroname::BrainStem::PortClass::getDataSpeed (*C++ function*), 795
Acroname::BrainStem::PortClass::getDataSS1Enabled (*C++ function*), 792
Acroname::BrainStem::PortClass::getDataSS2Enabled (*C++ function*), 792
Acroname::BrainStem::PortClass::getDataSSEnabled (*C++ function*), 791
Acroname::BrainStem::PortClass::getDataSSRoutingBehavior (*C++ function*), 798
Acroname::BrainStem::PortClass::getEnabled (*C++ function*), 790

Acroname::BrainStem::PortClass::getErrors (C++ function), 795
 Acroname::BrainStem::PortClass::getHSBoost (C++ function), 800
 Acroname::BrainStem::PortClass::getMode (C++ function), 795
 Acroname::BrainStem::PortClass::getName (C++ function), 797
 Acroname::BrainStem::PortClass::getPowerEnabled (C++ function), 792
 Acroname::BrainStem::PortClass::getPowerLimit (C++ function), 796
 Acroname::BrainStem::PortClass::getPowerLimitMode (C++ function), 797
 Acroname::BrainStem::PortClass::getPowerMode (C++ function), 790
 Acroname::BrainStem::PortClass::getSBU1Voltage (C++ function), 801
 Acroname::BrainStem::PortClass::getSBU2Voltage (C++ function), 801
 Acroname::BrainStem::PortClass::getState (C++ function), 795
 Acroname::BrainStem::PortClass::getVbusAccumulatedPower (C++ function), 799
 Acroname::BrainStem::PortClass::getVbusCurrent (C++ function), 789
 Acroname::BrainStem::PortClass::getVbusVoltage (C++ function), 789
 Acroname::BrainStem::PortClass::getVconn1Enabled (C++ function), 793
 Acroname::BrainStem::PortClass::getVconn2Enabled (C++ function), 793
 Acroname::BrainStem::PortClass::getVconnAccumulatedPower (C++ function), 799
 Acroname::BrainStem::PortClass::getVconnCurrent (C++ function), 789
 Acroname::BrainStem::PortClass::getVconnEnabled (C++ function), 793
 Acroname::BrainStem::PortClass::getVconnVoltage (C++ function), 789
 Acroname::BrainStem::PortClass::getVoltageSetpoint (C++ function), 794
 Acroname::BrainStem::PortClass::init (C++ function), 789
 Acroname::BrainStem::PortClass::PortClass (C++ function), 789
 Acroname::BrainStem::PortClass::resetVbusAccumulatedPower (C++ function), 799
 Acroname::BrainStem::PortClass::resetVconnAccumulatedPower (C++ function), 800
 Acroname::BrainStem::PortClass::setCC1AccumulatedPower (C++ function), 802
 Acroname::BrainStem::PortClass::setCC1Enabled (C++ function), 794
 Acroname::BrainStem::PortClass::setCC2AccumulatedPower (C++ function), 802
 Acroname::BrainStem::PortClass::setCC2Enabled (C++ function), 794
 Acroname::BrainStem::PortClass::setCCCCurrentLimit (C++ function), 798
 Acroname::BrainStem::PortClass::setCCEnabled (C++ function), 794
 Acroname::BrainStem::PortClass::setCurrentLimit (C++ function), 796
 Acroname::BrainStem::PortClass::setCurrentLimitMode (C++ function), 796
 Acroname::BrainStem::PortClass::setDataEnabled (C++ function), 790
 Acroname::BrainStem::PortClass::setDataHS1Enabled (C++ function), 791
 Acroname::BrainStem::PortClass::setDataHS2Enabled (C++ function), 791
 Acroname::BrainStem::PortClass::setDataHSEnabled (C++ function), 791
 Acroname::BrainStem::PortClass::setDataHSRoutingBehavior (C++ function), 798
 Acroname::BrainStem::PortClass::setDataSS1Enabled (C++ function), 792
 Acroname::BrainStem::PortClass::setDataSS2Enabled (C++ function), 792
 Acroname::BrainStem::PortClass::setDataSSEnabled (C++ function), 791
 Acroname::BrainStem::PortClass::setDataSSRoutingBehavior (C++ function), 799
 Acroname::BrainStem::PortClass::setEnabled (C++ function), 790
 Acroname::BrainStem::PortClass::setHSBoost (C++ function), 800
 Acroname::BrainStem::PortClass::setMode (C++ function), 795
 Acroname::BrainStem::PortClass::setName (C++ function), 797
 Acroname::BrainStem::PortClass::setPowerEnabled (C++ function), 792
 Acroname::BrainStem::PortClass::setPowerLimit (C++ function), 797
 Acroname::BrainStem::PortClass::setPowerLimitMode (C++ function), 797
 Acroname::BrainStem::PortClass::setPowerMode (C++ function), 790
 Acroname::BrainStem::PortClass::setVbusAccumulatedPower (C++ function), 799
 Acroname::BrainStem::PortClass::setVconn1Enabled (C++ function), 793
 Acroname::BrainStem::PortClass::setVconn2Enabled (C++ function), 793
 Acroname::BrainStem::PortClass::setVconnAccumulatedPower (C++ function), 799
 Acroname::BrainStem::PortClass::setVconnEnabled (C++ function), 793
 Acroname::BrainStem::PortClass::setVoltageSetpoint (C++ function), 795
 Acroname::BrainStem::PowerDeliveryClass (C++ class), 803
 Acroname::BrainStem::PowerDeliveryClass::~PowerDeliveryClass (C++ function), 803
 Acroname::BrainStem::PowerDeliveryClass::getAttachTimeElapsed (C++ function), 806
 Acroname::BrainStem::PowerDeliveryClass::getCableCurrentMax (C++ function), 808
 Acroname::BrainStem::PowerDeliveryClass::getCableOrientation (C++ function), 809
 Acroname::BrainStem::PowerDeliveryClass::getCableSpeedMax (C++ function), 809
 Acroname::BrainStem::PowerDeliveryClass::getCableType (C++ function), 809
 Acroname::BrainStem::PowerDeliveryClass::getCableVoltageMax (C++ function), 808
 Acroname::BrainStem::PowerDeliveryClass::getConnectionState (C++ function), 803
 Acroname::BrainStem::PowerDeliveryClass::getDataRoleCapabilities (C++ function), 808
 Acroname::BrainStem::PowerDeliveryClass::getFastRoleSwapCurrent (C++ function), 812
 Acroname::BrainStem::PowerDeliveryClass::getFlagMode (C++ function), 811

Acroname::BrainStem::PowerDeliveryClass::getLinkState (C++ function), 806
Acroname::BrainStem::PowerDeliveryClass::getNumberOfPowerDataObjects (C++ function), 803
Acroname::BrainStem::PowerDeliveryClass::getOverride (C++ function), 810
Acroname::BrainStem::PowerDeliveryClass::getPeakCurrentConfiguration (C++ function), 812
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObject (C++ function), 803
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObjectEnabled (C++ function), 805
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObjectEnabledList (C++ function), 805
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObjectList (C++ function), 804
Acroname::BrainStem::PowerDeliveryClass::getPowerRole (C++ function), 807
Acroname::BrainStem::PowerDeliveryClass::getPowerRoleCapabilities (C++ function), 807
Acroname::BrainStem::PowerDeliveryClass::getPowerRolePreferred (C++ function), 807
Acroname::BrainStem::PowerDeliveryClass::getProtocolRevisionMax (C++ function), 811
Acroname::BrainStem::PowerDeliveryClass::getRequestDataObject (C++ function), 806
Acroname::BrainStem::PowerDeliveryClass::init (C++ function), 803
Acroname::BrainStem::PowerDeliveryClass::packDataObjectAttributes (C++ function), 813
Acroname::BrainStem::PowerDeliveryClass::PowerDeliveryClass (C++ function), 803
Acroname::BrainStem::PowerDeliveryClass::request (C++ function), 809
Acroname::BrainStem::PowerDeliveryClass::requestStatus (C++ function), 810
Acroname::BrainStem::PowerDeliveryClass::resetPowerDataObjectToDefault (C++ function), 804
Acroname::BrainStem::PowerDeliveryClass::setFastRoleSwapCurrent (C++ function), 812
Acroname::BrainStem::PowerDeliveryClass::setFlagMode (C++ function), 811
Acroname::BrainStem::PowerDeliveryClass::setOverride (C++ function), 810
Acroname::BrainStem::PowerDeliveryClass::setPeakCurrentConfiguration (C++ function), 812
Acroname::BrainStem::PowerDeliveryClass::setPowerDataObject (C++ function), 804
Acroname::BrainStem::PowerDeliveryClass::setPowerDataObjectEnabled (C++ function), 805
Acroname::BrainStem::PowerDeliveryClass::setPowerRole (C++ function), 807
Acroname::BrainStem::PowerDeliveryClass::setPowerRolePreferred (C++ function), 808
Acroname::BrainStem::PowerDeliveryClass::setProtocolRevisionMax (C++ function), 811
Acroname::BrainStem::PowerDeliveryClass::setRequestDataObject (C++ function), 806
Acroname::BrainStem::PowerDeliveryClass::unpackDataObjectAttributes (C++ function), 813
Acroname::BrainStem::RailClass (C++ class), 815
Acroname::BrainStem::RailClass::~RailClass (C++ function), 815
Acroname::BrainStem::RailClass::clearFaults (C++ function), 820
Acroname::BrainStem::RailClass::getCurrent (C++ function), 815
Acroname::BrainStem::RailClass::getCurrentLimit (C++ function), 816
Acroname::BrainStem::RailClass::getCurrentSetpoint (C++ function), 816
Acroname::BrainStem::RailClass::getEnable (C++ function), 816
Acroname::BrainStem::RailClass::getKelvinSensingEnable (C++ function), 820
Acroname::BrainStem::RailClass::getKelvinSensingState (C++ function), 820
Acroname::BrainStem::RailClass::getOperationalMode (C++ function), 820
Acroname::BrainStem::RailClass::getOperationalState (C++ function), 820
Acroname::BrainStem::RailClass::getPower (C++ function), 818
Acroname::BrainStem::RailClass::getPowerLimit (C++ function), 819
Acroname::BrainStem::RailClass::getPowerSetpoint (C++ function), 818
Acroname::BrainStem::RailClass::getResistance (C++ function), 819
Acroname::BrainStem::RailClass::getResistanceSetpoint (C++ function), 819
Acroname::BrainStem::RailClass::getTemperature (C++ function), 816
Acroname::BrainStem::RailClass::getVoltage (C++ function), 817
Acroname::BrainStem::RailClass::getVoltageMaxLimit (C++ function), 818
Acroname::BrainStem::RailClass::getVoltageMinLimit (C++ function), 818
Acroname::BrainStem::RailClass::getVoltageSetpoint (C++ function), 817
Acroname::BrainStem::RailClass::init (C++ function), 815
Acroname::BrainStem::RailClass::RailClass (C++ function), 815
Acroname::BrainStem::RailClass::setCurrentLimit (C++ function), 816
Acroname::BrainStem::RailClass::setCurrentSetpoint (C++ function), 816
Acroname::BrainStem::RailClass::setEnable (C++ function), 817
Acroname::BrainStem::RailClass::setKelvinSensingEnable (C++ function), 819
Acroname::BrainStem::RailClass::setOperationalMode (C++ function), 820
Acroname::BrainStem::RailClass::setPowerLimit (C++ function), 819
Acroname::BrainStem::RailClass::setPowerSetpoint (C++ function), 818
Acroname::BrainStem::RailClass::setResistanceSetpoint (C++ function), 819
Acroname::BrainStem::RailClass::setVoltageMaxLimit (C++ function), 818
Acroname::BrainStem::RailClass::setVoltageMinLimit (C++ function), 817
Acroname::BrainStem::RailClass::setVoltageSetpoint (C++ function), 817
Acroname::BrainStem::RCServoClass (C++ class), 814
Acroname::BrainStem::RCServoClass::~RCServoClass (C++ function), 814
Acroname::BrainStem::RCServoClass::getEnable (C++ function), 814
Acroname::BrainStem::RCServoClass::getPosition (C++ function), 814

Acroname::BrainStem::RCServoClass::getReverse (C++ function), 815
 Acroname::BrainStem::RCServoClass::init (C++ function), 814
 Acroname::BrainStem::RCServoClass::RCServoClass (C++ function), 814
 Acroname::BrainStem::RCServoClass::setEnabled (C++ function), 814
 Acroname::BrainStem::RCServoClass::setPosition (C++ function), 814
 Acroname::BrainStem::RCServoClass::setReverse (C++ function), 815
 Acroname::BrainStem::RelayClass (C++ class), 821
 Acroname::BrainStem::RelayClass::~~RelayClass (C++ function), 821
 Acroname::BrainStem::RelayClass::getEnable (C++ function), 821
 Acroname::BrainStem::RelayClass::getVoltage (C++ function), 821
 Acroname::BrainStem::RelayClass::init (C++ function), 821
 Acroname::BrainStem::RelayClass::RelayClass (C++ function), 821
 Acroname::BrainStem::RelayClass::setEnabled (C++ function), 821
 Acroname::BrainStem::SignalClass (C++ class), 822
 Acroname::BrainStem::SignalClass::~~SignalClass (C++ function), 822
 Acroname::BrainStem::SignalClass::getEnable (C++ function), 822
 Acroname::BrainStem::SignalClass::getInvert (C++ function), 823
 Acroname::BrainStem::SignalClass::getT2Time (C++ function), 823
 Acroname::BrainStem::SignalClass::getT3Time (C++ function), 823
 Acroname::BrainStem::SignalClass::init (C++ function), 822
 Acroname::BrainStem::SignalClass::setEnabled (C++ function), 822
 Acroname::BrainStem::SignalClass::setInvert (C++ function), 822
 Acroname::BrainStem::SignalClass::setT2Time (C++ function), 823
 Acroname::BrainStem::SignalClass::setT3Time (C++ function), 823
 Acroname::BrainStem::SignalClass::SignalClass (C++ function), 822
 Acroname::BrainStem::StoreClass (C++ class), 824
 Acroname::BrainStem::StoreClass::~~StoreClass (C++ function), 824
 Acroname::BrainStem::StoreClass::getSlotCapacity (C++ function), 825
 Acroname::BrainStem::StoreClass::getSlotLocked (C++ function), 825
 Acroname::BrainStem::StoreClass::getSlotSize (C++ function), 825
 Acroname::BrainStem::StoreClass::getSlotState (C++ function), 824
 Acroname::BrainStem::StoreClass::init (C++ function), 824
 Acroname::BrainStem::StoreClass::loadSlot (C++ function), 824
 Acroname::BrainStem::StoreClass::setSlotLocked (C++ function), 826
 Acroname::BrainStem::StoreClass::slotDisable (C++ function), 825
 Acroname::BrainStem::StoreClass::slotEnable (C++ function), 825
 Acroname::BrainStem::StoreClass::StoreClass (C++ function), 824
 Acroname::BrainStem::StoreClass::unloadSlot (C++ function), 824
 Acroname::BrainStem::SystemClass (C++ class), 826
 Acroname::BrainStem::SystemClass::~~SystemClass (C++ function), 826
 Acroname::BrainStem::SystemClass::getBootSlot (C++ function), 828
 Acroname::BrainStem::SystemClass::getBuild (C++ function), 829
 Acroname::BrainStem::SystemClass::getErrors (C++ function), 834
 Acroname::BrainStem::SystemClass::getHardwareVersion (C++ function), 829
 Acroname::BrainStem::SystemClass::getHBInterval (C++ function), 827
 Acroname::BrainStem::SystemClass::getInputCurrent (C++ function), 830
 Acroname::BrainStem::SystemClass::getInputPowerBehavior (C++ function), 833
 Acroname::BrainStem::SystemClass::getInputPowerBehaviorConfig (C++ function), 833
 Acroname::BrainStem::SystemClass::getInputPowerSource (C++ function), 832
 Acroname::BrainStem::SystemClass::getInputVoltage (C++ function), 830
 Acroname::BrainStem::SystemClass::getLED (C++ function), 827
 Acroname::BrainStem::SystemClass::getLEDMaxBrightness (C++ function), 828
 Acroname::BrainStem::SystemClass::getLinkInterface (C++ function), 834
 Acroname::BrainStem::SystemClass::getMaximumTemperature (C++ function), 830
 Acroname::BrainStem::SystemClass::getMinimumTemperature (C++ function), 830
 Acroname::BrainStem::SystemClass::getModel (C++ function), 829
 Acroname::BrainStem::SystemClass::getModule (C++ function), 826
 Acroname::BrainStem::SystemClass::getModuleBaseAddress (C++ function), 826
 Acroname::BrainStem::SystemClass::getModuleHardwareOffset (C++ function), 830
 Acroname::BrainStem::SystemClass::getModuleSoftwareOffset (C++ function), 831
 Acroname::BrainStem::SystemClass::getName (C++ function), 833
 Acroname::BrainStem::SystemClass::getPowerLimit (C++ function), 831
 Acroname::BrainStem::SystemClass::getPowerLimitMax (C++ function), 832
 Acroname::BrainStem::SystemClass::getPowerLimitState (C++ function), 832
 Acroname::BrainStem::SystemClass::getProtocolFeatures (C++ function), 835
 Acroname::BrainStem::SystemClass::getRouter (C++ function), 827
 Acroname::BrainStem::SystemClass::getRouterAddressSetting (C++ function), 831
 Acroname::BrainStem::SystemClass::getSerialNumber (C++ function), 829

Acroname::BrainStem::SystemClass::getTemperature (C++ function), 830
Acroname::BrainStem::SystemClass::getUnregulatedCurrent (C++ function), 832
Acroname::BrainStem::SystemClass::getUnregulatedVoltage (C++ function), 832
Acroname::BrainStem::SystemClass::getUptime (C++ function), 830
Acroname::BrainStem::SystemClass::getVersion (C++ function), 828
Acroname::BrainStem::SystemClass::init (C++ function), 826
Acroname::BrainStem::SystemClass::logEvents (C++ function), 829
Acroname::BrainStem::SystemClass::reset (C++ function), 829
Acroname::BrainStem::SystemClass::resetDeviceToFactoryDefaults (C++ function), 834
Acroname::BrainStem::SystemClass::routeToMe (C++ function), 831
Acroname::BrainStem::SystemClass::save (C++ function), 829
Acroname::BrainStem::SystemClass::setBootSlot (C++ function), 828
Acroname::BrainStem::SystemClass::setHBInterval (C++ function), 827
Acroname::BrainStem::SystemClass::setInputPowerBehavior (C++ function), 833
Acroname::BrainStem::SystemClass::setInputPowerBehaviorConfig (C++ function), 833
Acroname::BrainStem::SystemClass::setLED (C++ function), 827
Acroname::BrainStem::SystemClass::setLEDMaxBrightness (C++ function), 828
Acroname::BrainStem::SystemClass::setLinkInterface (C++ function), 834
Acroname::BrainStem::SystemClass::setModuleSoftwareOffset (C++ function), 831
Acroname::BrainStem::SystemClass::setName (C++ function), 834
Acroname::BrainStem::SystemClass::setPowerLimitMax (C++ function), 832
Acroname::BrainStem::SystemClass::setRouter (C++ function), 827
Acroname::BrainStem::SystemClass::SystemClass (C++ function), 826
Acroname::BrainStem::TemperatureClass (C++ class), 835
Acroname::BrainStem::TemperatureClass::~TemperatureClass (C++ function), 835
Acroname::BrainStem::TemperatureClass::getValue (C++ function), 835
Acroname::BrainStem::TemperatureClass::getValueMax (C++ function), 835
Acroname::BrainStem::TemperatureClass::getValueMin (C++ function), 835
Acroname::BrainStem::TemperatureClass::init (C++ function), 835
Acroname::BrainStem::TemperatureClass::TemperatureClass (C++ function), 835
Acroname::BrainStem::TimerClass (C++ class), 836
Acroname::BrainStem::TimerClass::~TimerClass (C++ function), 836
Acroname::BrainStem::TimerClass::getExpiration (C++ function), 836
Acroname::BrainStem::TimerClass::getMode (C++ function), 836
Acroname::BrainStem::TimerClass::init (C++ function), 836
Acroname::BrainStem::TimerClass::setExpiration (C++ function), 836
Acroname::BrainStem::TimerClass::setMode (C++ function), 837
Acroname::BrainStem::TimerClass::TimerClass (C++ function), 836
Acroname::BrainStem::UARTClass (C++ class), 837
Acroname::BrainStem::UARTClass::~UARTClass (C++ function), 837
Acroname::BrainStem::UARTClass::getAvailableProtocols (C++ function), 840
Acroname::BrainStem::UARTClass::getBaudRate (C++ function), 838
Acroname::BrainStem::UARTClass::getCapableProtocols (C++ function), 840
Acroname::BrainStem::UARTClass::getDataBits (C++ function), 840
Acroname::BrainStem::UARTClass::getEnable (C++ function), 837
Acroname::BrainStem::UARTClass::getFlowControl (C++ function), 840
Acroname::BrainStem::UARTClass::getLinkChannel (C++ function), 838
Acroname::BrainStem::UARTClass::getParity (C++ function), 839
Acroname::BrainStem::UARTClass::getProtocol (C++ function), 838
Acroname::BrainStem::UARTClass::getStopBits (C++ function), 839
Acroname::BrainStem::UARTClass::init (C++ function), 837
Acroname::BrainStem::UARTClass::setBaudRate (C++ function), 838
Acroname::BrainStem::UARTClass::setDataBits (C++ function), 840
Acroname::BrainStem::UARTClass::setEnable (C++ function), 837
Acroname::BrainStem::UARTClass::setFlowControl (C++ function), 840
Acroname::BrainStem::UARTClass::setLinkChannel (C++ function), 838
Acroname::BrainStem::UARTClass::setParity (C++ function), 839
Acroname::BrainStem::UARTClass::setProtocol (C++ function), 838
Acroname::BrainStem::UARTClass::setStopBits (C++ function), 839
Acroname::BrainStem::UARTClass::UARTClass (C++ function), 837
Acroname::BrainStem::USBCClass (C++ class), 841
Acroname::BrainStem::USBCClass::~USBCClass (C++ function), 841
Acroname::BrainStem::USBCClass::clearPortErrorStatus (C++ function), 843
Acroname::BrainStem::USBCClass::getAltModeConfig (C++ function), 850
Acroname::BrainStem::USBCClass::getCableFlip (C++ function), 849
Acroname::BrainStem::USBCClass::getCC1Current (C++ function), 848
Acroname::BrainStem::USBCClass::getCC1Enable (C++ function), 847
Acroname::BrainStem::USBCClass::getCC1Voltage (C++ function), 848

Acroname::BrainStem::USBClass::getCC2Current (C++ function), 848
 Acroname::BrainStem::USBClass::getCC2Enable (C++ function), 848
 Acroname::BrainStem::USBClass::getCC2Voltage (C++ function), 849
 Acroname::BrainStem::USBClass::getConnectMode (C++ function), 847
 Acroname::BrainStem::USBClass::getDownstreamBoostMode (C++ function), 846
 Acroname::BrainStem::USBClass::getDownstreamDataSpeed (C++ function), 846
 Acroname::BrainStem::USBClass::getEnumerationDelay (C++ function), 844
 Acroname::BrainStem::USBClass::getHubMode (C++ function), 843
 Acroname::BrainStem::USBClass::getPortCurrent (C++ function), 843
 Acroname::BrainStem::USBClass::getPortCurrentLimit (C++ function), 845
 Acroname::BrainStem::USBClass::getPortError (C++ function), 845
 Acroname::BrainStem::USBClass::getPortMode (C++ function), 845
 Acroname::BrainStem::USBClass::getPortState (C++ function), 845
 Acroname::BrainStem::USBClass::getPortVoltage (C++ function), 843
 Acroname::BrainStem::USBClass::getSBU1Voltage (C++ function), 850
 Acroname::BrainStem::USBClass::getSBU2Voltage (C++ function), 850
 Acroname::BrainStem::USBClass::getSBUEnable (C++ function), 849
 Acroname::BrainStem::USBClass::getUpstreamBoostMode (C++ function), 846
 Acroname::BrainStem::USBClass::getUpstreamMode (C++ function), 844
 Acroname::BrainStem::USBClass::getUpstreamState (C++ function), 844
 Acroname::BrainStem::USBClass::init (C++ function), 841
 Acroname::BrainStem::USBClass::setAltModeConfig (C++ function), 850
 Acroname::BrainStem::USBClass::setCableFlip (C++ function), 849
 Acroname::BrainStem::USBClass::setCC1Enable (C++ function), 847
 Acroname::BrainStem::USBClass::setCC2Enable (C++ function), 848
 Acroname::BrainStem::USBClass::setConnectMode (C++ function), 847
 Acroname::BrainStem::USBClass::setDataDisable (C++ function), 842
 Acroname::BrainStem::USBClass::setDataEnable (C++ function), 841
 Acroname::BrainStem::USBClass::setDownstreamBoostMode (C++ function), 846
 Acroname::BrainStem::USBClass::setEnumerationDelay (C++ function), 844
 Acroname::BrainStem::USBClass::setHiSpeedDataDisable (C++ function), 842
 Acroname::BrainStem::USBClass::setHiSpeedDataEnable (C++ function), 842
 Acroname::BrainStem::USBClass::setHubMode (C++ function), 843
 Acroname::BrainStem::USBClass::setPortCurrentLimit (C++ function), 844
 Acroname::BrainStem::USBClass::setPortDisable (C++ function), 841
 Acroname::BrainStem::USBClass::setPortEnable (C++ function), 841
 Acroname::BrainStem::USBClass::setPortMode (C++ function), 845
 Acroname::BrainStem::USBClass::setPowerDisable (C++ function), 843
 Acroname::BrainStem::USBClass::setPowerEnable (C++ function), 842
 Acroname::BrainStem::USBClass::setSBUEnable (C++ function), 849
 Acroname::BrainStem::USBClass::setSuperSpeedDataDisable (C++ function), 842
 Acroname::BrainStem::USBClass::setSuperSpeedDataEnable (C++ function), 842
 Acroname::BrainStem::USBClass::setUpstreamBoostMode (C++ function), 846
 Acroname::BrainStem::USBClass::setUpstreamMode (C++ function), 844
 Acroname::BrainStem::USBClass::USBClass (C++ function), 841
 Acroname::BrainStem::USBSysClass (C++ class), 850
 Acroname::BrainStem::USBSysClass::~USBSysClass (C++ function), 851
 Acroname::BrainStem::USBSysClass::getDataHSMaxDatarate (C++ function), 855
 Acroname::BrainStem::USBSysClass::getDataRoleBehavior (C++ function), 853
 Acroname::BrainStem::USBSysClass::getDataRoleBehaviorConfig (C++ function), 854
 Acroname::BrainStem::USBSysClass::getDataRoleList (C++ function), 851
 Acroname::BrainStem::USBSysClass::getDataSSMaxDatarate (C++ function), 856
 Acroname::BrainStem::USBSysClass::getEnabledList (C++ function), 851
 Acroname::BrainStem::USBSysClass::getEnumerationDelay (C++ function), 851
 Acroname::BrainStem::USBSysClass::getModeList (C++ function), 852
 Acroname::BrainStem::USBSysClass::getOverride (C++ function), 855
 Acroname::BrainStem::USBSysClass::getPowerBehavior (C++ function), 852
 Acroname::BrainStem::USBSysClass::getPowerBehaviorConfig (C++ function), 853
 Acroname::BrainStem::USBSysClass::getSelectorMode (C++ function), 854
 Acroname::BrainStem::USBSysClass::getStateList (C++ function), 852
 Acroname::BrainStem::USBSysClass::getUpstream (C++ function), 851
 Acroname::BrainStem::USBSysClass::getUpstreamHS (C++ function), 854
 Acroname::BrainStem::USBSysClass::getUpstreamSS (C++ function), 855
 Acroname::BrainStem::USBSysClass::init (C++ function), 851
 Acroname::BrainStem::USBSysClass::setDataHSMaxDatarate (C++ function), 855
 Acroname::BrainStem::USBSysClass::setDataRoleBehavior (C++ function), 853
 Acroname::BrainStem::USBSysClass::setDataRoleBehaviorConfig (C++ function), 854
 Acroname::BrainStem::USBSysClass::setDataSSMaxDatarate (C++ function), 856

Acroname::BrainStem::USBSysClass::setEnabledList (C++ function), 852
Acroname::BrainStem::USBSysClass::setEnumerationDelay (C++ function), 851
Acroname::BrainStem::USBSysClass::setModeList (C++ function), 852
Acroname::BrainStem::USBSysClass::setOverride (C++ function), 855
Acroname::BrainStem::USBSysClass::setPowerBehavior (C++ function), 853
Acroname::BrainStem::USBSysClass::setPowerBehaviorConfig (C++ function), 853
Acroname::BrainStem::USBSysClass::setSelectorMode (C++ function), 854
Acroname::BrainStem::USBSysClass::setUpstream (C++ function), 851
Acroname::BrainStem::USBSysClass::setUpstreamHS (C++ function), 855
Acroname::BrainStem::USBSysClass::setUpstreamSS (C++ function), 855
Acroname::BrainStem::USBSysClass::USBSysClass (C++ function), 851
Acroname::BrainStem::Utilities::PDChannelLogger (C++ class), 873
Acroname::BrainStem::Utilities::PDChannelLogger::getIndex (C++ function), 873
Acroname::BrainStem::Utilities::PDChannelLogger::getPacket (C++ function), 873
Acroname::BrainStem::Utilities::PDChannelLogger::getPackets (C++ function), 873
Acroname::BrainStem::Utilities::PDChannelLogger::PDChannelLogger (C++ function), 873
Acroname::BrainStem::Utilities::PDChannelLogger::setEnabled (C++ function), 873
address (brainstem.module.Module property), 582
aDefs_GetModelName (C++ function), 875
aDiscovery_EnumerateModules (C++ function), 877
aDiscovery_FindFirstModule (C++ function), 878
aDiscovery_FindModule (C++ function), 877
aDiscoveryModuleFoundProc (C++ type), 877
aErr (C++ enum), 879
aErr::aErrAsyncReturn (C++ enumerator), 881
aErr::aErrBusy (C++ enumerator), 879
aErr::aErrCancel (C++ enumerator), 880
aErr::aErrConfiguration (C++ enumerator), 880
aErr::aErrConnection (C++ enumerator), 880
aErr::aErrDuplicate (C++ enumerator), 880
aErr::aErrEOF (C++ enumerator), 879
aErr::aErrFileNameLength (C++ enumerator), 879
aErr::aErrIndexRange (C++ enumerator), 880
aErr::aErrInitialization (C++ enumerator), 880
aErr::aErrInvalidEntity (C++ enumerator), 880
aErr::aErrInvalidOption (C++ enumerator), 881
aErr::aErrIO (C++ enumerator), 879
aErr::aErrMedia (C++ enumerator), 881
aErr::aErrMemory (C++ enumerator), 879
aErr::aErrMode (C++ enumerator), 879
aErr::aErrNone (C++ enumerator), 879
aErr::aErrNotFound (C++ enumerator), 879
aErr::aErrNotReady (C++ enumerator), 879
aErr::aErrOverrun (C++ enumerator), 880
aErr::aErrPacket (C++ enumerator), 880
aErr::aErrParam (C++ enumerator), 879
aErr::aErrParse (C++ enumerator), 880
aErr::aErrPermission (C++ enumerator), 879
aErr::aErrRange (C++ enumerator), 879
aErr::aErrRead (C++ enumerator), 879
aErr::aErrResource (C++ enumerator), 881
aErr::aErrShortCommand (C++ enumerator), 880
aErr::aErrSize (C++ enumerator), 880
aErr::aErrStreamStale (C++ enumerator), 881
aErr::aErrTimeout (C++ enumerator), 880
aErr::aErrUnimplemented (C++ enumerator), 880
aErr::aErrUnknown (C++ enumerator), 881
aErr::aErrVersion (C++ enumerator), 880
aErr::aErrWrite (C++ enumerator), 879
aError_GetErrorText (C++ function), 881
AETHER (brainstem.link.Spec attribute), 581
AETHER_SERIAL (brainstem.link.Spec attribute), 581
AETHER_TCPIP (brainstem.link.Spec attribute), 581
AETHER_USB (brainstem.link.Spec attribute), 581
aEtherConfig (class in brainstem.link), 582
aFile_Close (C++ function), 882
aFile_Delete (C++ function), 884
aFile_Exists (C++ function), 882

aFile_GetSize (*C++ function*), 884
aFile_Open (*C++ function*), 882
aFile_Read (*C++ function*), 883
aFile_Seek (*C++ function*), 883
aFile_Write (*C++ function*), 883
aFileMode (*C++ enum*), 881
aFileMode::aFileModeAppend (*C++ enumerator*), 881
aFileMode::aFileModeReadOnly (*C++ enumerator*), 881
aFileMode::aFileModeUnknown (*C++ enumerator*), 882
aFileMode::aFileModeWriteOnly (*C++ enumerator*), 881
aFileRef (*C++ type*), 881
aFileSeekMode (*C++ enum*), 882
aFileSeekMode::aSeekCurrent (*C++ enumerator*), 882
aFileSeekMode::aSeekEnd (*C++ enumerator*), 882
aFileSeekMode::aSeekStart (*C++ enumerator*), 882
aLIBEXPORT (*C macro*), 875
aLink_AwaitFirst (*C++ function*), 887
aLink_AwaitPacket (*C++ function*), 887
aLink_CreateTCPIP (*C++ function*), 886
aLink_CreateUSB (*C++ function*), 885
aLink_Destroy (*C++ function*), 886
aLink_DrainPackets (*C++ function*), 888
aLink_GetFirst (*C++ function*), 887
aLink_GetPacket (*C++ function*), 887
aLink_GetStatus (*C++ function*), 887
aLink_PutPacket (*C++ function*), 888
aLink_Reset (*C++ function*), 886
aLinkRef (*C++ type*), 885
aLinkSpec_Create (*C++ function*), 878
aLinkSpec_Destroy (*C++ function*), 878
aMemPtr (*C macro*), 875
aMTM_ETHERSTEM_BULK_CAPTURE_MAX_HZ (*C macro*), 739
aMTM_ETHERSTEM_BULK_CAPTURE_MIN_HZ (*C macro*), 739
aMTM_ETHERSTEM_MODULE_BASE_ADDRESS (*C macro*), 739
aMTM_ETHERSTEM_NUM_A2D (*C macro*), 739
aMTM_ETHERSTEM_NUM_APPS (*C macro*), 739
aMTM_ETHERSTEM_NUM_CLOCK (*C macro*), 739
aMTM_ETHERSTEM_NUM_DIG (*C macro*), 739
aMTM_ETHERSTEM_NUM_I2C (*C macro*), 740
aMTM_ETHERSTEM_NUM_INPUT_SIGNALS (*C macro*), 740
aMTM_ETHERSTEM_NUM_INTERNAL_SLOTS (*C macro*), 739
aMTM_ETHERSTEM_NUM_OUTPUT_SIGNALS (*C macro*), 740
aMTM_ETHERSTEM_NUM_POINTERS (*C macro*), 740
aMTM_ETHERSTEM_NUM_RAM_SLOTS (*C macro*), 739
aMTM_ETHERSTEM_NUM_SD_SLOTS (*C macro*), 739
aMTM_ETHERSTEM_NUM_SERVOS (*C macro*), 740
aMTM_ETHERSTEM_NUM_SIGNALS (*C macro*), 740
aMTM_ETHERSTEM_NUM_STORES (*C macro*), 739
aMTM_ETHERSTEM_NUM_TIMERS (*C macro*), 740
aMTM_STEM_BULK_CAPTURE_MAX_HZ (*C macro*), 753
aMTM_STEM_BULK_CAPTURE_MIN_HZ (*C macro*), 753
aMTM_STEM_MODULE_BASE_ADDRESS (*C macro*), 753
aMTM_STEM_NUM_A2D (*C macro*), 753
aMTM_STEM_NUM_APPS (*C macro*), 753
aMTM_STEM_NUM_CLOCK (*C macro*), 753
aMTM_STEM_NUM_DIG (*C macro*), 753
aMTM_STEM_NUM_I2C (*C macro*), 753
aMTM_STEM_NUM_INPUT_SIGNALS (*C macro*), 754
aMTM_STEM_NUM_INTERNAL_SLOTS (*C macro*), 754
aMTM_STEM_NUM_OUTPUT_SIGNALS (*C macro*), 754
aMTM_STEM_NUM_POINTERS (*C macro*), 753
aMTM_STEM_NUM_RAM_SLOTS (*C macro*), 754
aMTM_STEM_NUM_SD_SLOTS (*C macro*), 754
aMTM_STEM_NUM_SERVOS (*C macro*), 754
aMTM_STEM_NUM_SIGNALS (*C macro*), 754
aMTM_STEM_NUM_STORES (*C macro*), 754
aMTM_STEM_NUM_TIMERS (*C macro*), 754
aMTM_USBSTEM_BULK_CAPTURE_MAX_HZ (*C macro*), 751

aMTM_USBSTEM_BULK_CAPTURE_MIN_HZ (*C macro*), 751
aMTM_USBSTEM_MODULE_BASE_ADDRESS (*C macro*), 751
aMTM_USBSTEM_NUM_A2D (*C macro*), 751
aMTM_USBSTEM_NUM_APPS (*C macro*), 751
aMTM_USBSTEM_NUM_CLOCK (*C macro*), 751
aMTM_USBSTEM_NUM_DIG (*C macro*), 751
aMTM_USBSTEM_NUM_I2C (*C macro*), 751
aMTM_USBSTEM_NUM_INPUT_SIGNALS (*C macro*), 751
aMTM_USBSTEM_NUM_INTERNAL_SLOTS (*C macro*), 752
aMTM_USBSTEM_NUM_OUTPUT_SIGNALS (*C macro*), 751
aMTM_USBSTEM_NUM_POINTERS (*C macro*), 751
aMTM_USBSTEM_NUM_RAM_SLOTS (*C macro*), 752
aMTM_USBSTEM_NUM_SD_SLOTS (*C macro*), 752
aMTM_USBSTEM_NUM_SERVOS (*C macro*), 751
aMTM_USBSTEM_NUM_SIGNALS (*C macro*), 751
aMTM_USBSTEM_NUM_STORES (*C macro*), 752
aMTM_USBSTEM_NUM_TIMERS (*C macro*), 752
aMTMDAQ2 (*C++ class*), 736
aMTMDAQ2::analog (*C++ member*), 736
aMTMDAQ2::app (*C++ member*), 736
aMTMDAQ2::digital (*C++ member*), 736
aMTMDAQ2::getDifferentialInputRanges (*C++ function*), 737
aMTMDAQ2::getOutputRanges (*C++ function*), 737
aMTMDAQ2::getSingleEndedInputRanges (*C++ function*), 737
aMTMDAQ2::i2c (*C++ member*), 736
aMTMDAQ2::pointer (*C++ member*), 737
aMTMDAQ2::store (*C++ member*), 737
aMTMDAQ2::system (*C++ member*), 737
aMTMDAQ2::timer (*C++ member*), 737
aMTMDAQ2_BULK_CAPTURE_MAX_HZ (*C macro*), 738
aMTMDAQ2_BULK_CAPTURE_MIN_HZ (*C macro*), 738
aMTMDAQ2_MODULE_BASE_ADDRESS (*C macro*), 737
aMTMDAQ2_NUM_ANALOG_INPUTS (*C macro*), 737
aMTMDAQ2_NUM_ANALOG_OUTPUTS (*C macro*), 737
aMTMDAQ2_NUM_ANALOGS (*C macro*), 737
aMTMDAQ2_NUM_APPS (*C macro*), 737
aMTMDAQ2_NUM_DIGITALS (*C macro*), 738
aMTMDAQ2_NUM_I2C (*C macro*), 738
aMTMDAQ2_NUM_INTERNAL_SLOTS (*C macro*), 738
aMTMDAQ2_NUM_POINTERS (*C macro*), 738
aMTMDAQ2_NUM_RAM_SLOTS (*C macro*), 738
aMTMDAQ2_NUM_STORES (*C macro*), 738
aMTMDAQ2_NUM_TIMERS (*C macro*), 738
aMTMEtherStem (*C++ class*), 738
aMTMIOSerial (*C++ class*), 740
aMTMIOSerial::app (*C++ member*), 741
aMTMIOSerial::digital (*C++ member*), 741
aMTMIOSerial::hub (*C++ member*), 741
aMTMIOSerial::HubClass (*C++ class*), 742
aMTMIOSerial::i2c (*C++ member*), 741
aMTMIOSerial::pointer (*C++ member*), 741
aMTMIOSerial::PORT_ID (*C++ enum*), 740
aMTMIOSerial::PORT_ID::kPORT_ID_0 (*C++ enumerator*), 740
aMTMIOSerial::PORT_ID::kPORT_ID_1 (*C++ enumerator*), 740
aMTMIOSerial::PORT_ID::kPORT_ID_2 (*C++ enumerator*), 740
aMTMIOSerial::PORT_ID::kPORT_ID_3 (*C++ enumerator*), 740
aMTMIOSerial::PORT_ID::kPORT_ID_UP0 (*C++ enumerator*), 741
aMTMIOSerial::PORT_ID_t (*C++ type*), 741
aMTMIOSerial::rail (*C++ member*), 741
aMTMIOSerial::servo (*C++ member*), 741
aMTMIOSerial::signal (*C++ member*), 741
aMTMIOSerial::store (*C++ member*), 741
aMTMIOSerial::system (*C++ member*), 741
aMTMIOSerial::temperature (*C++ member*), 741
aMTMIOSerial::timer (*C++ member*), 741
aMTMIOSerial::uart (*C++ member*), 741
aMTMIOSerial::usb (*C++ member*), 742
aMTMIO SERIAL_5VRAIL (*C macro*), 742

aMTMIO SERIAL_ADJR AIL1 (*C macro*), 742
 aMTMIO SERIAL_ADJR AIL2 (*C macro*), 742
 aMTMIO SERIAL_ERROR_VBUS_OVERCURRENT (*C macro*), 744
 aMTMIO SERIAL_MAX_MICROVOLTAGE (*C macro*), 742
 aMTMIO SERIAL_MIN_MICROVOLTAGE (*C macro*), 742
 aMTMIO SERIAL_MODULE_BASE_ADDRESS (*C macro*), 742
 aMTMIO SERIAL_NUM_APPS (*C macro*), 742
 aMTMIO SERIAL_NUM_DIGITALS (*C macro*), 742
 aMTMIO SERIAL_NUM_I2C (*C macro*), 742
 aMTMIO SERIAL_NUM_INPUT_SIGNALS (*C macro*), 743
 aMTMIO SERIAL_NUM_INTERNAL_SLOTS (*C macro*), 743
 aMTMIO SERIAL_NUM_OUTPUT_SIGNALS (*C macro*), 743
 aMTMIO SERIAL_NUM_POINTERS (*C macro*), 742
 aMTMIO SERIAL_NUM_PORTS (*C macro*), 743
 aMTMIO SERIAL_NUM_RAILS (*C macro*), 742
 aMTMIO SERIAL_NUM_RAM_SLOTS (*C macro*), 743
 aMTMIO SERIAL_NUM_SERVOS (*C macro*), 742
 aMTMIO SERIAL_NUM_SIGNALS (*C macro*), 743
 aMTMIO SERIAL_NUM_STORES (*C macro*), 743
 aMTMIO SERIAL_NUM_TIMERS (*C macro*), 743
 aMTMIO SERIAL_NUM_UART (*C macro*), 743
 aMTMIO SERIAL_NUM_USB (*C macro*), 743
 aMTMIO SERIAL_NUM_USB_PORTS (*C macro*), 743
 aMTMIO SERIAL_USB2_BOOST_ENABLED (*C macro*), 744
 aMTMIO SERIAL_USB2_DATA_ENABLED (*C macro*), 744
 aMTMIO SERIAL_USB_ERROR_FLAG (*C macro*), 744
 aMTMIO SERIAL_USB_NUM_CHANNELS (*C macro*), 743
 aMTMIO SERIAL_USB_VBUS_ENABLED (*C macro*), 744
 aMTMLoad1 (*C++ class*), 744
 aMTMLoad1::app (*C++ member*), 745
 aMTMLoad1::digital (*C++ member*), 745
 aMTMLoad1::i2c (*C++ member*), 745
 aMTMLoad1::pointer (*C++ member*), 745
 aMTMLoad1::rail (*C++ member*), 745
 aMTMLoad1::store (*C++ member*), 745
 aMTMLoad1::system (*C++ member*), 745
 aMTMLoad1::temperature (*C++ member*), 745
 aMTMLoad1::timer (*C++ member*), 745
 aMTMLOAD1_MAX_CURRENT_LIMIT_MICROAMPS (*C macro*), 746
 aMTMLOAD1_MAX_MICROAMPS (*C macro*), 746
 aMTMLOAD1_MAX_MICROOHMS (*C macro*), 746
 aMTMLOAD1_MAX_MICROVOLTAGE (*C macro*), 746
 aMTMLOAD1_MAX_MILLIWATTS (*C macro*), 746
 aMTMLOAD1_MAX_POWER_LIMIT_MILLIWATTS (*C macro*), 746
 aMTMLOAD1_MAX_VOLTAGE_LIMIT_MICROVOLTS (*C macro*), 746
 aMTMLOAD1_MIN_CURRENT_LIMIT_MICROAMPS (*C macro*), 746
 aMTMLOAD1_MIN_MICROAMPS (*C macro*), 746
 aMTMLOAD1_MIN_MICROOHMS (*C macro*), 746
 aMTMLOAD1_MIN_MICROVOLTAGE (*C macro*), 746
 aMTMLOAD1_MIN_MILLIWATTS (*C macro*), 746
 aMTMLOAD1_MIN_POWER_LIMIT_MILLIWATTS (*C macro*), 746
 aMTMLOAD1_MIN_VOLTAGE_LIMIT_MICROVOLTS (*C macro*), 746
 aMTMLOAD1_MODULE_BASE_ADDRESS (*C macro*), 745
 aMTMLOAD1_NUM_APPS (*C macro*), 745
 aMTMLOAD1_NUM_DIGITALS (*C macro*), 745
 aMTMLOAD1_NUM_I2C (*C macro*), 745
 aMTMLOAD1_NUM_INTERNAL_SLOTS (*C macro*), 747
 aMTMLOAD1_NUM_POINTERS (*C macro*), 745
 aMTMLOAD1_NUM_RAILS (*C macro*), 745
 aMTMLOAD1_NUM_RAM_SLOTS (*C macro*), 747
 aMTMLOAD1_NUM_STORES (*C macro*), 747
 aMTMLOAD1_NUM_TEMPERATURES (*C macro*), 747
 aMTMLOAD1_NUM_TIMERS (*C macro*), 747
 aMTMLOAD1_RAIL0 (*C macro*), 746
 aMTMPM1 (*C++ class*), 747
 aMTMPM1::app (*C++ member*), 747
 aMTMPM1::digital (*C++ member*), 747
 aMTMPM1::i2c (*C++ member*), 747

aMTMPM1::pointer (C++ member), 747
aMTMPM1::rail (C++ member), 747
aMTMPM1::store (C++ member), 747
aMTMPM1::system (C++ member), 748
aMTMPM1::temperature (C++ member), 748
aMTMPM1::timer (C++ member), 748
aMTMPM1_MAX_CURRENT_LIMIT_MICROAMPS (C macro), 748
aMTMPM1_MAX_MICROVOLTAGE (C macro), 748
aMTMPM1_MIN_CURRENT_LIMIT_MICROAMPS (C macro), 749
aMTMPM1_MIN_MICROVOLTAGE (C macro), 748
aMTMPM1_MODULE_BASE_ADDRESS (C macro), 748
aMTMPM1_NUM_APPS (C macro), 748
aMTMPM1_NUM_DIGITALS (C macro), 748
aMTMPM1_NUM_I2C (C macro), 748
aMTMPM1_NUM_INTERNAL_SLOTS (C macro), 749
aMTMPM1_NUM_POINTERS (C macro), 748
aMTMPM1_NUM_RAILS (C macro), 748
aMTMPM1_NUM_RAM_SLOTS (C macro), 749
aMTMPM1_NUM_STORES (C macro), 749
aMTMPM1_NUM_TEMPERATURES (C macro), 749
aMTMPM1_NUM_TIMERS (C macro), 749
aMTMPM1_RAIL0 (C macro), 748
aMTMPM1_RAIL1 (C macro), 748
aMTMRelay (C++ class), 749
aMTMRelay::app (C++ member), 749
aMTMRelay::digital (C++ member), 749
aMTMRelay::i2c (C++ member), 749
aMTMRelay::pointer (C++ member), 749
aMTMRelay::relay (C++ member), 749
aMTMRelay::store (C++ member), 750
aMTMRelay::system (C++ member), 750
aMTMRelay::timer (C++ member), 750
aMTMRELAY_MODULE_BASE_ADDRESS (C macro), 750
aMTMRELAY_NUM_APPS (C macro), 750
aMTMRELAY_NUM_DIGITALS (C macro), 750
aMTMRELAY_NUM_I2C (C macro), 750
aMTMRELAY_NUM_INTERNAL_SLOTS (C macro), 750
aMTMRELAY_NUM_POINTERS (C macro), 750
aMTMRELAY_NUM_RAM_SLOTS (C macro), 750
aMTMRELAY_NUM_RELAYS (C macro), 750
aMTMRELAY_NUM_STORES (C macro), 750
aMTMRELAY_NUM_TIMERS (C macro), 750
aMTMStemModule (C++ class), 752
aMTMStemModule::analog (C++ member), 752
aMTMStemModule::app (C++ member), 752
aMTMStemModule::clock (C++ member), 752
aMTMStemModule::digital (C++ member), 752
aMTMStemModule::i2c (C++ member), 752
aMTMStemModule::pointer (C++ member), 752
aMTMStemModule::servo (C++ member), 753
aMTMStemModule::signal (C++ member), 753
aMTMStemModule::store (C++ member), 753
aMTMStemModule::system (C++ member), 753
aMTMStemModule::timer (C++ member), 753
aMTMUSBStem (C++ class), 751
aMutex_Create (C++ function), 889
aMutex_Destroy (C++ function), 889
aMutex_Identifier (C++ function), 889
aMutex_Lock (C++ function), 889
aMutex_TryLock (C++ function), 889
aMutex_Unlock (C++ function), 890
aMutexRef (C++ type), 888
Analog (class in brainstem.entity), 592
analog_getBulkCaptureNumberOfSamples (C++ function), 1289
analog_getBulkCaptureSampleRate (C++ function), 1288
analog_getBulkCaptureState (C++ function), 1289
analog_getConfiguration (C++ function), 1288
analog_getEnable (C++ function), 1286

analog_getRange (C++ function), 1285
 analog_getValue (C++ function), 1285
 analog_getVoltage (C++ function), 1285
 analog_Hz_Maximum (C macro), 894
 analog_Hz_Minimum (C macro), 894
 analog_initiateBulkCapture (C++ function), 1289
 analog_setBulkCaptureNumberOfSamples (C++ function), 1288
 analog_setBulkCaptureSampleRate (C++ function), 1288
 analog_setConfiguration (C++ function), 1287
 analog_setEnable (C++ function), 1287
 analog_setRange (C++ function), 1287
 analog_setValue (C++ function), 1286
 analog_setVoltage (C++ function), 1286
 analogBulkCapture (C macro), 894
 analogBulkCaptureNumberOfSamples (C macro), 894
 analogBulkCaptureSampleRate (C macro), 894
 analogBulkCaptureState (C macro), 894
 analogConfiguration (C macro), 894
 analogConfigurationHiZ (C macro), 894
 analogConfigurationInput (C macro), 894
 analogConfigurationOutput (C macro), 894
 analogEnable (C macro), 896
 analogNumberOfOptions (C macro), 896
 analogRange (C macro), 895
 analogRange_P0V064N0V064 (C macro), 895
 analogRange_P0V128N0V128 (C macro), 895
 analogRange_P0V256N0V256 (C macro), 895
 analogRange_P0V512N0V512 (C macro), 895
 analogRange_P0V64N0V64 (C macro), 895
 analogRange_P10V24N0V0 (C macro), 895
 analogRange_P10V24N10V24 (C macro), 895
 analogRange_P1V024N1V024 (C macro), 895
 analogRange_P1V28N0V0 (C macro), 895
 analogRange_P1V28N1V28 (C macro), 895
 analogRange_P2V048N0V0 (C macro), 896
 analogRange_P2V56N0V0 (C macro), 895
 analogRange_P2V56N2V56 (C macro), 895
 analogRange_P4V096N0V0 (C macro), 896
 analogRange_P5V12N0V0 (C macro), 895
 analogRange_P5V12N5V12 (C macro), 895
 analogValue (C macro), 894
 analogVoltage (C macro), 894
 aPacket (C++ struct), 890
 aPacket_AddByte (C++ function), 891
 aPacket_Create (C++ function), 891
 aPacket_CreateWithData (C++ function), 891
 aPacket_Destroy (C++ function), 892
 aPacket_IsComplete (C++ function), 891
 aPacket_Reset (C++ function), 891
 App (class in *brainstem.entity*), 596
 app_execute (C++ function), 1290
 app_executeAndReturn (C++ function), 1290
 appExecute (C macro), 896
 appReturn (C macro), 896
 aSerial_Bits (C++ enum), 963
 aSerial_Bits::aBITS_7 (C++ enumerator), 963
 aSerial_Bits::aBITS_8 (C++ enumerator), 963
 aSerial_Stop_bits (C++ enum), 963
 aSerial_Stop_bits::aSTOP_BITS_1 (C++ enumerator), 963
 aSerial_Stop_bits::aSTOP_BITS_2 (C++ enumerator), 963
 aSHOWERR (C macro), 874
 aSNPRINTF (C macro), 874
 assignedPort (*brainstem.link.aEtherConfig* attribute), 582
 aStream_Create (C++ function), 963
 aStream_CreateFileInput (C++ function), 963
 aStream_CreateFileOutput (C++ function), 964
 aStream_CreateLogStream (C++ function), 967
 aStream_CreateMemory (C++ function), 965

aStream_CreatePipe (C++ function), 966
aStream_CreateSerial (C++ function), 964
aStream_CreateSocket (C++ function), 964
aStream_CreateUSB (C++ function), 965
aStream_Destroy (C++ function), 971
aStream_Flush (C++ function), 971
aStream_Read (C++ function), 967
aStream_ReadCString (C++ function), 969
aStream_ReadCStringRecord (C++ function), 970
aStream_ReadRecord (C++ function), 968
aStream_Write (C++ function), 968
aStream_WriteCString (C++ function), 969
aStream_WriteCStringRecord (C++ function), 970
aStream_WriteRecord (C++ function), 969
aStreamBuffer_Create (C++ function), 966
aStreamBuffer_Flush (C++ function), 967
aStreamBuffer_Get (C++ function), 966
aStreamDeleteProc (C++ type), 961
aStreamGetProc (C++ type), 961
aStreamPutProc (C++ type), 961
aStreamRef (C++ type), 960
aStreamWriteProc (C++ type), 962
aStringCatSafe (C macro), 874
aStringCopySafe (C macro), 874
aSystemBootSlotNone (C macro), 943
aTime_GetMSTicks (C++ function), 971
aTime_MSSleep (C++ function), 971
aUEI_RetrieveInt (C++ function), 973
aUEI_RetrieveShort (C++ function), 973
aUEI_StoreInt (C++ function), 973
aUEI_StoreShort (C++ function), 973
aUSB_UPSTREAM_CONFIG_AUTO (C macro), 743
aUSB_UPSTREAM_CONFIG_EDGE (C macro), 743
aUSB_UPSTREAM_CONFIG_ONBOARD (C macro), 743
aUSB_UPSTREAM_EDGE (C macro), 744
aUSB_UPSTREAM_ONBOARD (C macro), 744
aUSBCSwitch (C++ class), 731
aUSBCSwitch::daughtercard_type (C++ enum), 733
aUSBCSwitch::daughtercard_type::NO_DAUGHTERCARD (C++ enumerator), 733
aUSBCSwitch::daughtercard_type::PASSIVE_DAUGHTERCARD (C++ enumerator), 733
aUSBCSwitch::daughtercard_type::REDRIVER_DAUGHTERCARD (C++ enumerator), 733
aUSBCSwitch::daughtercard_type::UNKNOWN_DAUGHTERCARD (C++ enumerator), 733
aUSBCSwitch::equalizer (C++ member), 734
aUSBCSwitch::EQUALIZER_2P0_RECEIVER_CONFIGS (C++ enum), 733
aUSBCSwitch::EQUALIZER_2P0_RECEIVER_CONFIGS::LEVEL_1_2P0 (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_2P0_RECEIVER_CONFIGS::LEVEL_2_2P0 (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS (C++ enum), 733
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_0mV (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_40mV (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_60mV (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_80mV (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS (C++ enum), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_10_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_11_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_12_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_13_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_14_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_15_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_16_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_1_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_2_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_3_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_4_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_5_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_6_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_7_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_8_3P0 (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_9_3P0 (C++ enumerator), 732

aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS (C++ enum), 731
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_0db_1100mV (C++ enumerator), 731
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_0db_1300mV (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_0db_900mV (C++ enumerator), 731
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_1db_1100mV (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_1db_900mV (C++ enumerator), 731
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_1db_COM_0db_1100mV (C++ enumerator), 731
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_1db_COM_0db_900mV (C++ enumerator), 731
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_1db_COM_1db_900mV (C++ enumerator), 731
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_2db_COM_2db_1100mV (C++ enumerator), 732
aUSBCSwitch::EQUALIZER_CHANNELS (C++ enum), 733
aUSBCSwitch::EQUALIZER_CHANNELS::BOTH (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_CHANNELS::COMMON (C++ enumerator), 733
aUSBCSwitch::EQUALIZER_CHANNELS::MUX (C++ enumerator), 733
aUSBCSwitch::mux (C++ member), 734
aUSBCSwitch::store (C++ member), 734
aUSBCSwitch::system (C++ member), 734
aUSBCSwitch::usb (C++ member), 734
aUSBCSWITCH_MODULE (C macro), 734
aUSBCSWITCH_NUM_EQ (C macro), 734
aUSBCSWITCH_NUM_INTERNAL_SLOTS (C macro), 734
aUSBCSWITCH_NUM_MUX (C macro), 734
aUSBCSWITCH_NUM_MUX_CHANNELS (C macro), 734
aUSBCSWITCH_NUM_RAM_SLOTS (C macro), 734
aUSBCSWITCH_NUM_STORES (C macro), 734
aUSBCSWITCH_NUM_USB (C macro), 734
aUSBCSwitchPro (C++ class), 728
aUSBCSwitchPro::digital (C++ member), 729
aUSBCSwitchPro::i2c (C++ member), 729
aUSBCSwitchPro::mux (C++ member), 729
aUSBCSwitchPro::pd (C++ member), 729
aUSBCSwitchPro::PORT_ID (C++ enum), 728
aUSBCSwitchPro::PORT_ID::kPORT_ID_0 (C++ enumerator), 728
aUSBCSwitchPro::PORT_ID::kPORT_ID_1 (C++ enumerator), 728
aUSBCSwitchPro::PORT_ID::kPORT_ID_2 (C++ enumerator), 728
aUSBCSwitchPro::PORT_ID::kPORT_ID_3 (C++ enumerator), 728
aUSBCSwitchPro::PORT_ID::kPORT_ID_COMMON (C++ enumerator), 728
aUSBCSwitchPro::PORT_ID::kPORT_ID_CONTROL (C++ enumerator), 728
aUSBCSwitchPro::PORT_ID_t (C++ type), 728
aUSBCSwitchPro::rail (C++ member), 729
aUSBCSwitchPro::store (C++ member), 729
aUSBCSwitchPro::system (C++ member), 729
aUSBCSwitchPro::temperature (C++ member), 729
aUSBCSwitchPro::uart (C++ member), 729
aUSBCSwitchPro::usb (C++ member), 729
aUSBCSWITCHPRO_MODULE (C macro), 729
aUSBCSWITCHPRO_NUM_DIGITALS (C macro), 731
aUSBCSWITCHPRO_NUM_EEPROM_SLOTS (C macro), 730
aUSBCSWITCHPRO_NUM_I2C (C macro), 730
aUSBCSWITCHPRO_NUM_INTERNAL_SLOTS (C macro), 729
aUSBCSWITCHPRO_NUM_MUX (C macro), 730
aUSBCSWITCHPRO_NUM_MUX_CHANNELS (C macro), 730
aUSBCSWITCHPRO_NUM_PD_PORTS (C macro), 730
aUSBCSWITCHPRO_NUM_PD_RULES_PER_PORT (C macro), 730
aUSBCSWITCHPRO_NUM_PORTS (C macro), 730
aUSBCSWITCHPRO_NUM_RAILS (C macro), 730
aUSBCSWITCHPRO_NUM_RAM_SLOTS (C macro), 729
aUSBCSWITCHPRO_NUM_STORES (C macro), 729
aUSBCSWITCHPRO_NUM_TEMPERATURES (C macro), 730
aUSBCSWITCHPRO_NUM_UART (C macro), 730
aUSBCSWITCHPRO_NUM_USB (C macro), 730
aUSBCSWITCHPRO_NUM_USB_PORTS (C macro), 730
aUSBCSWITCHPRO_STORE_EEPROM_INDEX (C macro), 730
aUSBCSWITCHPRO_STORE_INTERNAL_INDEX (C macro), 730
aUSBCSWITCHPRO_STORE_RAM_INDEX (C macro), 730
aUSBHub2x4 (C++ class), 725
aUSBHub2x4::app (C++ member), 726
aUSBHub2x4::hub (C++ member), 726

aUSBHub2x4::HubClass (C++ class), 726
aUSBHub2x4::pointer (C++ member), 726
aUSBHub2x4::PORT_ID (C++ enum), 725
aUSBHub2x4::PORT_ID::kPORT_ID_0 (C++ enumerator), 725
aUSBHub2x4::PORT_ID::kPORT_ID_1 (C++ enumerator), 725
aUSBHub2x4::PORT_ID::kPORT_ID_2 (C++ enumerator), 725
aUSBHub2x4::PORT_ID::kPORT_ID_3 (C++ enumerator), 725
aUSBHub2x4::PORT_ID::kPORT_ID_UP0 (C++ enumerator), 725
aUSBHub2x4::PORT_ID::kPORT_ID_UP1 (C++ enumerator), 725
aUSBHub2x4::PORT_ID_t (C++ type), 725
aUSBHub2x4::store (C++ member), 726
aUSBHub2x4::system (C++ member), 726
aUSBHub2x4::temperature (C++ member), 726
aUSBHub2x4::timer (C++ member), 726
aUSBHub2x4::usb (C++ member), 726
aUSBHUB2X4_CONSTANT_CURRENT (C macro), 727
aUSBHUB2X4_DEVICE_ATTACHED (C macro), 727
aUSBHub2X4_ERROR_DISCHARGE (C macro), 728
aUSBHUB2X4_ERROR_OVER_TEMPERATURE (C macro), 728
aUSBHUB2X4_ERROR_VBUS_OVERCURRENT (C macro), 728
aUSBHUB2X4_MODULE (C macro), 726
aUSBHUB2X4_NUM_APPS (C macro), 726
aUSBHUB2X4_NUM_INTERNAL_SLOTS (C macro), 726
aUSBHUB2X4_NUM_POINTERS (C macro), 726
aUSBHUB2x4_NUM_PORTS (C macro), 727
aUSBHUB2X4_NUM_RAM_SLOTS (C macro), 726
aUSBHUB2X4_NUM_STORES (C macro), 726
aUSBHUB2X4_NUM_TIMERS (C macro), 727
aUSBHUB2X4_NUM_USB (C macro), 727
aUSBHUB2x4_NUM_USB_PORTS (C macro), 727
aUSBHUB2X4_USB2_BOOST_ENABLED (C macro), 727
aUSBHUB2X4_USB2_DATA_ENABLED (C macro), 727
aUSBHUB2X4_USB_ERROR_FLAG (C macro), 727
aUSBHUB2X4_USB_VBUS_ENABLED (C macro), 727
aUSBHub3c (C++ class), 719
aUSBHub3c::hub (C++ member), 720
aUSBHub3c::HubClass (C++ class), 720
aUSBHub3c::i2c (C++ member), 720
aUSBHub3c::pd (C++ member), 720
aUSBHub3c::PORT_ID (C++ enum), 719
aUSBHub3c::PORT_ID::kPORT_ID_0 (C++ enumerator), 719
aUSBHub3c::PORT_ID::kPORT_ID_1 (C++ enumerator), 719
aUSBHub3c::PORT_ID::kPORT_ID_2 (C++ enumerator), 719
aUSBHub3c::PORT_ID::kPORT_ID_3 (C++ enumerator), 719
aUSBHub3c::PORT_ID::kPORT_ID_4 (C++ enumerator), 719
aUSBHub3c::PORT_ID::kPORT_ID_5 (C++ enumerator), 719
aUSBHub3c::PORT_ID::kPORT_ID_CONTROL (C++ enumerator), 719
aUSBHub3c::PORT_ID::kPORT_ID_POWER_C (C++ enumerator), 719
aUSBHub3c::PORT_ID_t (C++ type), 719
aUSBHub3c::rail (C++ member), 720
aUSBHub3c::store (C++ member), 720
aUSBHub3c::system (C++ member), 720
aUSBHub3c::temperature (C++ member), 720
aUSBHub3c::uart (C++ member), 720
aUSBHub3c::usb (C++ member), 720
aUSBHUB3C_MODULE (C macro), 720
aUSBHUB3C_NUM_I2C (C macro), 721
aUSBHUB3C_NUM_INTERNAL_SLOTS (C macro), 720
aUSBHUB3C_NUM_PD_PORTS (C macro), 721
aUSBHUB3C_NUM_PD_RULES_PER_PORT (C macro), 721
aUSBHUB3C_NUM_PORTS (C macro), 721
aUSBHUB3C_NUM_RAILS (C macro), 721
aUSBHUB3C_NUM_RAM_SLOTS (C macro), 720
aUSBHUB3C_NUM_STORES (C macro), 720
aUSBHUB3C_NUM_TEMPERATURES (C macro), 721
aUSBHUB3C_NUM_UART (C macro), 721
aUSBHUB3C_NUM_USB (C macro), 721
aUSBHUB3C_NUM_USB_PORTS (C macro), 721

aUSBHUB3C_STORE_EEPROM_INDEX (*C macro*), 721
 aUSBHUB3C_STORE_INTERNAL_INDEX (*C macro*), 720
 aUSBHUB3C_STORE_RAM_INDEX (*C macro*), 721
 aUSBHub3p (*C++ class*), 721
 aUSBHub3p::app (*C++ member*), 722
 aUSBHub3p::hub (*C++ member*), 722
 aUSBHub3p::HubClass (*C++ class*), 723
 aUSBHub3p::pointer (*C++ member*), 722
 aUSBHub3p::PORT_ID (*C++ enum*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_0 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_1 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_2 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_3 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_4 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_5 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_6 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_7 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_CONTROL (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_DWNA (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_UP0 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID::kPORT_ID_UP1 (*C++ enumerator*), 722
 aUSBHub3p::PORT_ID_t (*C++ type*), 722
 aUSBHub3p::store (*C++ member*), 722
 aUSBHub3p::system (*C++ member*), 723
 aUSBHub3p::temperature (*C++ member*), 723
 aUSBHub3p::timer (*C++ member*), 723
 aUSBHub3p::usb (*C++ member*), 723
 aUSBHUB3P_DEVICE_ATTACHED (*C macro*), 724
 aUSBHUB3P_ERROR_DISCHARGE_ERR (*C macro*), 725
 aUSBHUB3P_ERROR_HUB_POWER (*C macro*), 724
 aUSBHUB3P_ERROR_OVER_TEMPERATURE (*C macro*), 725
 aUSBHUB3P_ERROR_SHORT_CIRCUIT (*C macro*), 725
 aUSBHUB3P_ERROR_VBUS_BACKDRIVE (*C macro*), 724
 aUSBHUB3P_ERROR_VBUS_OVERCURRENT (*C macro*), 724
 aUSBHUB3P_MODULE (*C macro*), 723
 aUSBHUB3P_NUM_APPS (*C macro*), 723
 aUSBHUB3P_NUM_INTERNAL_SLOTS (*C macro*), 723
 aUSBHUB3P_NUM_POINTERS (*C macro*), 723
 aUSBHUB3P_NUM_PORTS (*C macro*), 724
 aUSBHUB3P_NUM_RAM_SLOTS (*C macro*), 723
 aUSBHUB3P_NUM_STORES (*C macro*), 723
 aUSBHUB3P_NUM_TIMERS (*C macro*), 723
 aUSBHUB3P_NUM_USB (*C macro*), 723
 aUSBHUB3P_NUM_USB_PORTS (*C macro*), 723
 aUSBHUB3P_USB2_BOOST_ENABLED (*C macro*), 724
 aUSBHUB3P_USB2_DATA_ENABLED (*C macro*), 724
 aUSBHUB3P_USB3_DATA_ENABLED (*C macro*), 724
 aUSBHUB3P_USB_ERROR_FLAG (*C macro*), 724
 aUSBHUB3P_USB_SPEED_USB2 (*C macro*), 724
 aUSBHUB3P_USB_SPEED_USB3 (*C macro*), 724
 aUSBHUB3P_USB_VBUS_ENABLED (*C macro*), 724
 aVALIDPACKET (*C++ function*), 890
 aVersion_DestroyFeatureList (*C++ function*), 976
 aVersion_GetFeatureList (*C++ function*), 976
 aVersion_GetMajor (*C++ function*), 975
 aVersion_GetMinor (*C++ function*), 975
 aVersion_GetPatch (*C++ function*), 975
 aVersion_GetString (*C++ function*), 975
 aVersion_IsAtLeast (*C++ function*), 975
 aVersion_IsLegacyFormat (*C++ function*), 975
 aVERSION_MAJOR (*C macro*), 974
 aVERSION_MINOR (*C macro*), 974
 aVersion_ParseMajor (*C++ function*), 974
 aVersion_ParseMinor (*C++ function*), 974
 aVersion_ParsePatch (*C++ function*), 975
 aVersion_ParseString (*C++ function*), 975
 aVERSION_PATCH (*C macro*), 974

B

bAutoNetworking (*brainstem.module.Module* property), 582
bContinueSearch (*C++ type*), 877
bitI2CAck (*C macro*), 907
bitI2CError (*C macro*), 907
bitSlotError (*C macro*), 940
brainstem.defs
 module, 574
brainstem.discover
 module, 574
brainstem.link
 module, 580
brainstem.module
 module, 582
brainstem.pd_channel_logger
 module, 586
brainstem.result
 module, 588
brainstem.version
 module, 589
BS_PD_Packet (*C++ struct*), 978
BS_PD_Packet (*class in brainstem.pd_channel_logger*), 586
BS_PD_Packet::ccChannel (*C++ member*), 979
BS_PD_Packet::channel (*C++ member*), 979
BS_PD_Packet::crc (*C++ member*), 979
BS_PD_Packet::direction (*C++ member*), 979
BS_PD_Packet::event (*C++ member*), 979
BS_PD_Packet::payload (*C++ member*), 979
BS_PD_Packet::seconds (*C++ member*), 979
BS_PD_Packet::sop (*C++ member*), 979
BS_PD_Packet::uSeconds (*C++ member*), 979
BS_PD_Packet_CCA (*C++ struct*), 1436
BS_PD_Packet_CCA::ccChannel (*C++ member*), 1437
BS_PD_Packet_CCA::channel (*C++ member*), 1437
BS_PD_Packet_CCA::crc (*C++ member*), 1437
BS_PD_Packet_CCA::direction (*C++ member*), 1437
BS_PD_Packet_CCA::event (*C++ member*), 1437
BS_PD_Packet_CCA::payload (*C++ member*), 1437
BS_PD_Packet_CCA::payloadSize (*C++ member*), 1437
BS_PD_Packet_CCA::seconds (*C++ member*), 1437
BS_PD_Packet_CCA::sop (*C++ member*), 1437
BS_PD_Packet_CCA::uSeconds (*C++ member*), 1437
BS_PD_Packet_Direction (*C++ enum*), 978
BS_PD_Packet_Direction::kBS_PD_Packet_Direction_Invalid (*C++ enumerator*), 978
BS_PD_Packet_Direction::kBS_PD_Packet_Direction_LAST (*C++ enumerator*), 978
BS_PD_Packet_Direction::kBS_PD_Packet_Direction_Receive (*C++ enumerator*), 978
BS_PD_Packet_Direction::kBS_PD_Packet_Direction_Sniff (*C++ enumerator*), 978
BS_PD_Packet_Direction::kBS_PD_Packet_Direction_Transmit (*C++ enumerator*), 978
BS_PD_Packet_Direction::kBS_PD_Packet_Direction_UNKNOWN (*C++ enumerator*), 978
BS_PD_Packet_SOP (*C++ enum*), 978
BS_PD_Packet_SOP::kBS_PD_Packet_SOP (*C++ enumerator*), 978
BS_PD_Packet_SOP::kBS_PD_Packet_SOP_LAST (*C++ enumerator*), 978
BS_PD_Packet_SOP::kBS_PD_Packet_SOP_P (*C++ enumerator*), 978
BS_PD_Packet_SOP::kBS_PD_Packet_SOP_P_DEBUG (*C++ enumerator*), 978
BS_PD_Packet_SOP::kBS_PD_Packet_SOP_P_P (*C++ enumerator*), 978
BS_PD_Packet_SOP::kBS_PD_Packet_SOP_P_P_DEBUG (*C++ enumerator*), 978
BS_PD_Packet_SOP::kBS_PD_Packet_SOP_UNKNOWN (*C++ enumerator*), 978
buffer_length (*brainstem.pd_channel_logger.PDChannelLogger* property), 587
bulkCaptureError (*C macro*), 894
bulkCaptureFinished (*C macro*), 894
bulkCaptureIdle (*C macro*), 894
bulkCapturePending (*C macro*), 894

C

call_UEI() (*brainstem.Entity_Entity.Entity* method), 576
capacityBuild (*C macro*), 897
capacityClassQuantity (*C macro*), 897

capacityEntityGroup (C macro), 897
 capacitySubClassQuantity (C macro), 897
 capacitySubClassSize (C macro), 897
 capacityUEI (C macro), 897
 cca_spec_to_python_spec() (brainstem.link.Spec static method), 581
 classQuantity() (brainstem.module.Module method), 583
 clearFaults() (brainstem.entity.Rail method), 664
 clearPortErrorStatus() (brainstem.entity.USB method), 697
 Clock (class in brainstem.entity), 597
 clock_getDay (C++ function), 1292
 clock_getHour (C++ function), 1293
 clock_getMinute (C++ function), 1293
 clock_getMonth (C++ function), 1291
 clock_getSecond (C++ function), 1294
 clock_getYear (C++ function), 1291
 clock_setDay (C++ function), 1292
 clock_setHour (C++ function), 1293
 clock_setMinute (C++ function), 1293
 clock_setMonth (C++ function), 1292
 clock_setSecond (C++ function), 1294
 clock_setYear (C++ function), 1291
 clockDay (C macro), 898
 clockHour (C macro), 898
 clockMinute (C macro), 898
 clockMonth (C macro), 898
 clockNumberOfOptions (C macro), 898
 clockSecond (C macro), 898
 clockYear (C macro), 898
 cmdANALOG (C macro), 893
 cmdAPP (C macro), 896
 cmdCAPACITY (C macro), 897
 cmdCLOCK (C macro), 898
 cmdDEBUG (C macro), 903
 cmdDIGITAL (C macro), 898
 cmdEQUALIZER (C macro), 900
 cmdETHERNET (C macro), 901
 cmdHB (C macro), 905
 cmdHDBASET (C macro), 903
 cmdI2C (C macro), 906
 cmdLAST (C macro), 903
 cmdMUX (C macro), 907
 cmdNOTIFY (C macro), 903
 cmdPOE (C macro), 908
 cmdPOINTER (C macro), 912
 cmdPORT (C macro), 913
 cmdPOWERDELIVERY (C macro), 923
 cmdRAIL (C macro), 933
 cmdRELAY (C macro), 937
 cmdSERVO (C macro), 938
 cmdSIGNAL (C macro), 938
 cmdSLOT (C macro), 939
 cmdSTORE (C macro), 940
 cmdSTREAM (C macro), 942
 cmdSYSTEM (C macro), 942
 cmdTEMPERATURE (C macro), 946
 cmdTIMER (C macro), 947
 cmdUART (C macro), 948
 cmdUSB (C macro), 950
 cmdUSBSYSTEM (C macro), 957
 cmdUSERCONFIG (C macro), 960
 command (brainstem.Entity_Entity.Entity property), 576
 connect() (brainstem.module.Module method), 583
 connect() (brainstem.stem.EtherStem method), 570
 connect() (brainstem.stem.MTMDAQ1 method), 569
 connect() (brainstem.stem.MTMDAQ2 method), 561
 connect() (brainstem.stem.MTMEtherStem method), 562
 connect() (brainstem.stem.MTMIOSerial method), 564
 connect() (brainstem.stem.MTMLoad1 method), 565

connect () (*brainstem.stem.MTMPM1 method*), 566
connect () (*brainstem.stem.MTMRelay method*), 567
connect () (*brainstem.stem.MTMUSBStem method*), 568
connect () (*brainstem.stem.USBCSwitch method*), 560
connect () (*brainstem.stem.USBCSwitchPro method*), 557
connect () (*brainstem.stem.USBHub2x4 method*), 556
connect () (*brainstem.stem.USBHub3c method*), 553
connect () (*brainstem.stem.USBHub3p method*), 555
connect () (*brainstem.stem.USBStem method*), 572
connectFromSpec () (*brainstem.module.Module method*), 583
connectThroughLinkModule () (*brainstem.module.Module method*), 583

D

dataType (*C++ enum*), 972
dataType::aUEI_BYTE (*C++ enumerator*), 972
dataType::aUEI_BYTES (*C++ enumerator*), 972
dataType::aUEI_INT (*C++ enumerator*), 972
dataType::aUEI_SHORT (*C++ enumerator*), 972
dataType::aUEI_VOID (*C++ enumerator*), 972
DefaultOperationalRailMode_Value (*C macro*), 934
DefaultTimerMode (*C macro*), 947
DeviceNode (*C++ struct*), 977
DeviceNode (*class in brainstem.discover*), 574
DeviceNode::hubPort (*C++ member*), 977
DeviceNode::hubSerialNumber (*C++ member*), 977
DeviceNode::idProduct (*C++ member*), 977
DeviceNode::idVendor (*C++ member*), 977
DeviceNode::manufacturer (*C++ member*), 977
DeviceNode::productName (*C++ member*), 977
DeviceNode::serialNumber (*C++ member*), 977
DeviceNode::speed (*C++ member*), 977
DeviceNode_CCA (*C++ struct*), 1439
DeviceNode_CCA::hubPort (*C++ member*), 1439
DeviceNode_CCA::hubSerialNumber (*C++ member*), 1439
DeviceNode_CCA::idProduct (*C++ member*), 1439
DeviceNode_CCA::idVendor (*C++ member*), 1439
DeviceNode_CCA::manufacturer (*C++ member*), 1440
DeviceNode_CCA::productName (*C++ member*), 1439
DeviceNode_CCA::serialNumber (*C++ member*), 1440
DeviceNode_CCA::speed (*C++ member*), 1439
Digital (*class in brainstem.entity*), 600
digital_getConfiguration (*C++ function*), 1295
digital_getLinkChannel (*C++ function*), 1297
digital_getState (*C++ function*), 1295
digital_getStateAll (*C++ function*), 1296
digital_getValue (*C++ function*), 1297
digital_setConfiguration (*C++ function*), 1294
digital_setLinkChannel (*C++ function*), 1297
digital_setState (*C++ function*), 1295
digital_setStateAll (*C++ function*), 1296
digital_setValue (*C++ function*), 1296
digitalConfiguration (*C macro*), 899
digitalConfigurationHiZ (*C macro*), 899
digitalConfigurationInput (*C macro*), 899
digitalConfigurationInputNoPull (*C macro*), 899
digitalConfigurationInputPullDown (*C macro*), 899
digitalConfigurationInputPullUp (*C macro*), 899
digitalConfigurationLinkInput (*C macro*), 899
digitalConfigurationLinkOutput (*C macro*), 899
digitalConfigurationOutput (*C macro*), 899
digitalConfigurationRCServoInput (*C macro*), 899
digitalConfigurationRCServoOutput (*C macro*), 899
digitalConfigurationSignalCounterInput (*C macro*), 899
digitalConfigurationSignalInput (*C macro*), 899
digitalConfigurationSignalOutput (*C macro*), 899
digitalLinkChannel (*C macro*), 900
digitalNumberOfOptions (*C macro*), 900

digitalState (*C macro*), 899
 digitalStateAll (*C macro*), 899
 digitalTimeDelay (*C macro*), 900
 digitalValue (*C macro*), 900
 disconnect () (*brainstem.module.Module method*), 583
 discoverAndConnect () (*brainstem.module.Module method*), 583
 drain_UEI () (*brainstem.Entity_Entity.Entity method*), 576

E

enabled (*brainstem.link.aEtherConfig attribute*), 582
 Entity (*class in brainstem.Entity_Entity*), 576
 entityGroup () (*brainstem.module.Module method*), 584
 Equalizer (*class in brainstem.entity*), 603
 equalizer2p0 (*C macro*), 901
 equalizer3p0 (*C macro*), 901
 equalizer_getReceiverConfig (*C++ function*), 1298
 equalizer_getTransmitterConfig (*C++ function*), 1299
 equalizer_setReceiverConfig (*C++ function*), 1298
 equalizer_setTransmitterConfig (*C++ function*), 1298
 equalizerManualConfig (*C macro*), 900
 equalizerNumberOfOptions (*C macro*), 900
 equalizerReceiverConfig (*C macro*), 900
 equalizerTransmitterConfig (*C macro*), 900
 error (*brainstem.result.Result property*), 588
 Ethernet (*class in brainstem.entity*), 604
 ethernet_getEnabled (*C++ function*), 1299
 ethernet_getHostname (*C++ function*), 1305
 ethernet_getInterfacePort (*C++ function*), 1306
 ethernet_getIPv4Address (*C++ function*), 1302
 ethernet_getIPv4DNSAddress (*C++ function*), 1304
 ethernet_getIPv4Gateway (*C++ function*), 1303
 ethernet_getIPv4Netmask (*C++ function*), 1303
 ethernet_getMACAddress (*C++ function*), 1305
 ethernet_getNetworkConfiguration (*C++ function*), 1300
 ethernet_getStaticIPv4Address (*C++ function*), 1300
 ethernet_getStaticIPv4DNSAddress (*C++ function*), 1304
 ethernet_getStaticIPv4Gateway (*C++ function*), 1302
 ethernet_getStaticIPv4Netmask (*C++ function*), 1301
 ethernet_setEnabled (*C++ function*), 1299
 ethernet_setHostname (*C++ function*), 1304
 ethernet_setInterfacePort (*C++ function*), 1305
 ethernet_setNetworkConfiguration (*C++ function*), 1300
 ethernet_setStaticIPv4Address (*C++ function*), 1301
 ethernet_setStaticIPv4DNSAddress (*C++ function*), 1304
 ethernet_setStaticIPv4Gateway (*C++ function*), 1302
 ethernet_setStaticIPv4Netmask (*C++ function*), 1302
 ethernetConfigurationDhcp (*C macro*), 901
 ethernetConfigurationNone (*C macro*), 901
 ethernetConfigurationStatic (*C macro*), 901
 ethernetEnabled (*C macro*), 901
 ethernetHostname (*C macro*), 902
 ethernetInterfacePort (*C macro*), 902
 ethernetInterfacePort_BrainStem_DiscoveryReply (*C macro*), 902
 ethernetInterfacePort_BrainStem_DiscoveryRequest (*C macro*), 902
 ethernetInterfacePort_BrainStem_TCP (*C macro*), 902
 ethernetInterfacePort_Max (*C macro*), 902
 ethernetInterfacePort_RestServer_HTTP (*C macro*), 902
 ethernetInterfacePort_RestServer_HTTPS (*C macro*), 902
 ethernetIPv4Address (*C macro*), 902
 ethernetIPv4DNSAddress (*C macro*), 902
 ethernetIPv4Gateway (*C macro*), 902
 ethernetIPv4Netmask (*C macro*), 902
 ethernetMACAddress (*C macro*), 902
 ethernetNetworkConfiguration (*C macro*), 901
 ethernetNumberOfOptions (*C macro*), 902
 ethernetStaticIPv4Address (*C macro*), 901
 ethernetStaticIPv4DNSAddress (*C macro*), 902

ethernetStaticIPv4Gateway (C macro), 901
ethernetStaticIPv4Netmask (C macro), 901
EtherStem (class in *brainstem.stem*), 570
execute() (*brainstem.entity.App* method), 596
executeAndReturn() (*brainstem.entity.App* method), 597

F

fallback (*brainstem.link.aEtherConfig* attribute), 582
findAllModules() (in module *brainstem.discover*), 575
findFirstModule() (in module *brainstem.discover*), 575
findModule() (in module *brainstem.discover*), 575

G

get_UEI16() (*brainstem.Entity_Entity.Entity* method), 577
get_UEI16_with_subindex() (*brainstem.Entity_Entity.Entity* method), 577
get_UEI32() (*brainstem.Entity_Entity.Entity* method), 577
get_UEI32_with_subindex() (*brainstem.Entity_Entity.Entity* method), 577
get_UEI8() (*brainstem.Entity_Entity.Entity* method), 577
get_UEI8_with_subindex() (*brainstem.Entity_Entity.Entity* method), 578
get_UEIBytes() (*brainstem.Entity_Entity.Entity* method), 578
get_usbPortStateDaughterCard (C macro), 736
get_version_string() (in module *brainstem.version*), 589
getAllocatedPower() (*brainstem.entity.Port* method), 627
getAltModeConfig() (*brainstem.entity.USB* method), 697
getAttachTimeElapsed() (*brainstem.entity.PowerDelivery* method), 647
getAvailablePower() (*brainstem.entity.Port* method), 628
getAvailableProtocols() (*brainstem.entity.UART* method), 692
getBaudRate() (*brainstem.entity.UART* method), 692
getBootSlot() (*brainstem.entity.System* method), 677
getBuild() (*brainstem.entity.System* method), 678
getBuild() (*brainstem.module.Module* method), 584
getBulkCaptureNumberOfSamples() (*brainstem.entity.Analog* method), 592
getBulkCaptureSampleRate() (*brainstem.entity.Analog* method), 592
getBulkCaptureState() (*brainstem.entity.Analog* method), 592
getCableCurrentMax() (*brainstem.entity.PowerDelivery* method), 648
getCableFlip() (*brainstem.entity.USB* method), 699
getCableLength() (*brainstem.entity.HDBaseT* method), 610
getCableOrientation() (*brainstem.entity.PowerDelivery* method), 648
getCableSpeedMax() (*brainstem.entity.PowerDelivery* method), 648
getCableType() (*brainstem.entity.PowerDelivery* method), 649
getCableVoltageMax() (*brainstem.entity.PowerDelivery* method), 649
getCapableProtocols() (*brainstem.entity.UART* method), 693
getCC1AccumulatedPower() (*brainstem.entity.Port* method), 628
getCC1Current() (*brainstem.entity.Port* method), 628
getCC1Current() (*brainstem.entity.USB* method), 698
getCC1Enable() (*brainstem.entity.USB* method), 698
getCC1Enabled() (*brainstem.entity.Port* method), 628
getCC1State() (*brainstem.entity.Port* method), 629
getCC1Voltage() (*brainstem.entity.Port* method), 629
getCC1Voltage() (*brainstem.entity.USB* method), 698
getCC2AccumulatedPower() (*brainstem.entity.Port* method), 629
getCC2Current() (*brainstem.entity.Port* method), 630
getCC2Current() (*brainstem.entity.USB* method), 699
getCC2Enable() (*brainstem.entity.USB* method), 699
getCC2Enabled() (*brainstem.entity.Port* method), 630
getCC2State() (*brainstem.entity.Port* method), 630
getCC2Voltage() (*brainstem.entity.Port* method), 631
getCC2Voltage() (*brainstem.entity.USB* method), 699
getCCCurrentLimit() (*brainstem.entity.Port* method), 631
getCCEnabled() (*brainstem.entity.Port* method), 631
getChannel() (*brainstem.entity.Mux* method), 615
getChannelVoltage() (*brainstem.entity.Mux* method), 615
getChar() (*brainstem.entity.Pointer* method), 624
getConfig() (*brainstem.module.Module* method), 584
getConfig() (*brainstem.entity.Analog* method), 593
getConfig() (*brainstem.entity.Digital* method), 600
getConfig() (*brainstem.entity.Mux* method), 616

getConnectionState() (*brainstem.entity.PowerDelivery method*), 650
 getConnectMode() (*brainstem.entity.USB method*), 700
 getCurrent() (*brainstem.entity.Rail method*), 664
 getCurrentLimit() (*brainstem.entity.Port method*), 631
 getCurrentLimit() (*brainstem.entity.Rail method*), 664
 getCurrentLimitMode() (*brainstem.entity.Port method*), 632
 getCurrentSetpoint() (*brainstem.entity.Rail method*), 664
 getDataBits() (*brainstem.entity.UART method*), 693
 getDataEnabled() (*brainstem.entity.Port method*), 632
 getDataHS1Enabled() (*brainstem.entity.Port method*), 632
 getDataHS2Enabled() (*brainstem.entity.Port method*), 633
 getDataHSEnabled() (*brainstem.entity.Port method*), 633
 getDataHSMaxDataRate() (*brainstem.entity.USBSystem method*), 710
 getDataHSRoutingBehavior() (*brainstem.entity.Port method*), 633
 getDataRole() (*brainstem.entity.Port method*), 633
 getDataRoleBehavior() (*brainstem.entity.USBSystem method*), 711
 getDataRoleBehaviorConfig() (*brainstem.entity.USBSystem method*), 711
 getDataRoleCapabilities() (*brainstem.entity.PowerDelivery method*), 650
 getDataRoleList() (*brainstem.entity.USBSystem method*), 711
 getDataSpeed() (*brainstem.entity.Port method*), 635
 getDataSS1Enabled() (*brainstem.entity.Port method*), 634
 getDataSS2Enabled() (*brainstem.entity.Port method*), 634
 getDataSSEnabled() (*brainstem.entity.Port method*), 634
 getDataSSMaxDataRate() (*brainstem.entity.USBSystem method*), 711
 getDataSSRoutingBehavior() (*brainstem.entity.Port method*), 634
 getDay() (*brainstem.entity.Clock method*), 597
 getDownstreamBoostMode() (*brainstem.entity.USB method*), 700
 getDownstreamDataSpeed() (*brainstem.entity.USB method*), 700
 getDownstreamDevices(C++ function), 977
 getDownstreamDevices() (*in module brainstem.discover*), 576
 getEnable() (*brainstem.entity.Analog method*), 593
 getEnable() (*brainstem.entity.Mux method*), 616
 getEnable() (*brainstem.entity.Rail method*), 665
 getEnable() (*brainstem.entity.RCServo method*), 662
 getEnable() (*brainstem.entity.Relay method*), 671
 getEnable() (*brainstem.entity.Signal method*), 672
 getEnable() (*brainstem.entity.UART method*), 693
 getEnabled() (*brainstem.entity.Ethernet method*), 604
 getEnabled() (*brainstem.entity.Port method*), 635
 getEnabledList() (*brainstem.entity.USBSystem method*), 712
 getEncodingState() (*brainstem.entity.HDBaseT method*), 610
 getEnumerationDelay() (*brainstem.entity.USB method*), 701
 getEnumerationDelay() (*brainstem.entity.USBSystem method*), 712
 getErrorDescription() (*brainstem.result.Result static method*), 588
 getErrors() (*brainstem.entity.Port method*), 635
 getErrors() (*brainstem.entity.System method*), 678
 getErrorText() (*brainstem.result.Result static method*), 588
 getExpiration() (*brainstem.entity.Timer method*), 691
 getFastRoleSwapCurrent() (*brainstem.entity.PowerDelivery method*), 650
 getFirmwareVersion() (*brainstem.entity.HDBaseT method*), 610
 getFlagMode() (*brainstem.entity.PowerDelivery method*), 651
 getFlowControl() (*brainstem.entity.UART method*), 693
 getHardwareVersion() (*brainstem.entity.System method*), 678
 getHBInterval() (*brainstem.entity.System method*), 678
 getHostname() (*brainstem.entity.Ethernet method*), 604
 getHour() (*brainstem.entity.Clock method*), 597
 getHSBoost() (*brainstem.entity.Port method*), 635
 getHubMode() (*brainstem.entity.USB method*), 701
 getInputCurrent() (*brainstem.entity.System method*), 679
 getInputPowerBehavior() (*brainstem.entity.System method*), 679
 getInputPowerBehaviorConfig() (*brainstem.entity.System method*), 679
 getInputPowerSource() (*brainstem.entity.System method*), 679
 getInputVoltage() (*brainstem.entity.System method*), 680
 getInt() (*brainstem.entity.Pointer method*), 624
 getInterfacePort() (*brainstem.entity.Ethernet method*), 606
 getInvert() (*brainstem.entity.Signal method*), 673
 getIPv4Address() (*brainstem.entity.Ethernet method*), 604
 getIPv4DNSAddress() (*brainstem.entity.Ethernet method*), 605

getIPv4Gateway() (*brainstem.entity.Ethernet method*), 605
getIPv4Interfaces() (*in module brainstem.discover*), 576
getIPv4Netmask() (*brainstem.entity.Ethernet method*), 605
getKelvinSensingEnable() (*brainstem.entity.Rail method*), 665
getKelvinSensingState() (*brainstem.entity.Rail method*), 665
getLED() (*brainstem.entity.System method*), 680
getLEDMaxBrightness() (*brainstem.entity.System method*), 680
getLinkChannel() (*brainstem.entity.Digital method*), 600
getLinkChannel() (*brainstem.entity.UART method*), 694
getLinkInterface() (*brainstem.entity.System method*), 681
getLinkRole() (*brainstem.entity.HDBaseT method*), 610
getLinkState() (*brainstem.entity.PowerDelivery method*), 651
getLinkUtilization() (*brainstem.entity.HDBaseT method*), 611
getMACAddress() (*brainstem.entity.Ethernet method*), 606
getMajor() (*in module brainstem.version*), 589
getMaximumTemperature() (*brainstem.entity.System method*), 681
getMinimumTemperature() (*brainstem.entity.System method*), 681
getMinor() (*in module brainstem.version*), 589
getMinute() (*brainstem.entity.Clock method*), 598
getMode() (*brainstem.entity.Pointer method*), 624
getMode() (*brainstem.entity.Port method*), 636
getMode() (*brainstem.entity.Timer method*), 691
getModel() (*brainstem.entity.System method*), 681
getModelList() (*brainstem.entity.USBSystem method*), 712
getModule() (*brainstem.entity.System method*), 682
getModuleAddress() (*brainstem.module.Module method*), 584
getModuleBaseAddress() (*brainstem.entity.System method*), 682
getModuleHardwareOffset() (*brainstem.entity.System method*), 682
getModuleSoftwareOffset() (*brainstem.entity.System method*), 682
getMonth() (*brainstem.entity.Clock method*), 598
getMSEA() (*brainstem.entity.HDBaseT method*), 611
getMSEB() (*brainstem.entity.HDBaseT method*), 611
getName() (*brainstem.entity.Port method*), 636
getName() (*brainstem.entity.System method*), 683
getNetworkConfiguration() (*brainstem.entity.Ethernet method*), 606
getNumberOfPowerDataObjects() (*brainstem.entity.PowerDelivery method*), 651
getOffset() (*brainstem.entity.Pointer method*), 625
getOperationalMode() (*brainstem.entity.Rail method*), 665
getOperationalState() (*brainstem.entity.Rail method*), 666
getOverride() (*brainstem.entity.PowerDelivery method*), 652
getOverride() (*brainstem.entity.USBSystem method*), 712
getPacket() (*brainstem.pd_channel_logger.PDChannelLogger method*), 587
getPackets() (*brainstem.pd_channel_logger.PDChannelLogger method*), 587
getPairAccumulatedPower() (*brainstem.entity.PoE method*), 617
getPairCapacitance() (*brainstem.entity.PoE method*), 618
getPairCurrent() (*brainstem.entity.PoE method*), 618
getPairDetectionStatus() (*brainstem.entity.PoE method*), 618
getPairDiscoveredClass() (*brainstem.entity.PoE method*), 619
getPairEnabled() (*brainstem.entity.PoE method*), 619
getPairPower() (*brainstem.entity.PoE method*), 619
getPairRequestedClass() (*brainstem.entity.PoE method*), 620
getPairResistance() (*brainstem.entity.PoE method*), 620
getPairSourcingClass() (*brainstem.entity.PoE method*), 621
getPairVoltage() (*brainstem.entity.PoE method*), 621
getParity() (*brainstem.entity.UART method*), 694
getPatch() (*in module brainstem.version*), 589
getPeakCurrentConfiguration() (*brainstem.entity.PowerDelivery method*), 652
getPortCurrent() (*brainstem.entity.USB method*), 701
getPortCurrentLimit() (*brainstem.entity.USB method*), 702
getPortError() (*brainstem.entity.USB method*), 702
getPortMode() (*brainstem.entity.USB method*), 702
getPortState() (*brainstem.entity.USB method*), 703
getPortVoltage() (*brainstem.entity.USB method*), 703
getPosition() (*brainstem.entity.RCServo method*), 662
getPower() (*brainstem.entity.Rail method*), 666
getPowerBehavior() (*brainstem.entity.USBSystem method*), 713
getPowerBehaviorConfig() (*brainstem.entity.USBSystem method*), 713
getPowerDataObject() (*brainstem.entity.PowerDelivery method*), 652

getPowerDataObjectEnabled() (*brainstem.entity.PowerDelivery method*), 653
 getPowerDataObjectEnabledList() (*brainstem.entity.PowerDelivery method*), 654
 getPowerDataObjectList() (*brainstem.entity.PowerDelivery method*), 654
 getPowerEnabled() (*brainstem.entity.Port method*), 636
 getPowerLimit() (*brainstem.entity.Port method*), 637
 getPowerLimit() (*brainstem.entity.Rail method*), 666
 getPowerLimit() (*brainstem.entity.System method*), 683
 getPowerLimitMax() (*brainstem.entity.System method*), 683
 getPowerLimitMode() (*brainstem.entity.Port method*), 637
 getPowerLimitState() (*brainstem.entity.System method*), 684
 getPowerMode() (*brainstem.entity.PoE method*), 621
 getPowerMode() (*brainstem.entity.Port method*), 637
 getPowerRole() (*brainstem.entity.PowerDelivery method*), 654
 getPowerRoleCapabilities() (*brainstem.entity.PowerDelivery method*), 655
 getPowerRolePreferred() (*brainstem.entity.PowerDelivery method*), 655
 getPowerSetpoint() (*brainstem.entity.Rail method*), 667
 getPowerState() (*brainstem.entity.PoE method*), 622
 getProtocol() (*brainstem.entity.UART method*), 694
 getProtocolFeatures() (*brainstem.entity.System method*), 684
 getProtocolRevisionMax() (*brainstem.entity.PowerDelivery method*), 656
 getRange() (*brainstem.entity.Analog method*), 593
 getReceiverConfig() (*brainstem.entity.Equalizer method*), 603
 getRequestDataObject() (*brainstem.entity.PowerDelivery method*), 656
 getResistance() (*brainstem.entity.Rail method*), 667
 getResistanceSetpoint() (*brainstem.entity.Rail method*), 667
 getRetransmissionRate() (*brainstem.entity.HDBaseT method*), 611
 getReverse() (*brainstem.entity.RCServo method*), 663
 getRouter() (*brainstem.entity.System method*), 684
 getRouterAddressSetting() (*brainstem.entity.System method*), 684
 getSBU1Voltage() (*brainstem.entity.Port method*), 637
 getSBU1Voltage() (*brainstem.entity.USB method*), 703
 getSBU2Voltage() (*brainstem.entity.Port method*), 638
 getSBU2Voltage() (*brainstem.entity.USB method*), 703
 getSBUEnable() (*brainstem.entity.USB method*), 704
 getSecond() (*brainstem.entity.Clock method*), 598
 getSelectorMode() (*brainstem.entity.USBSystem method*), 713
 getSerialNumber() (*brainstem.entity.HDBaseT method*), 612
 getSerialNumber() (*brainstem.entity.System method*), 685
 getShort() (*brainstem.entity.Pointer method*), 625
 getSlotCapacity() (*brainstem.entity.Store method*), 675
 getSlotLocked() (*brainstem.entity.Store method*), 675
 getSlotSize() (*brainstem.entity.Store method*), 675
 getSlotState() (*brainstem.entity.Store method*), 676
 getSpeed() (*brainstem.entity.I2C method*), 613
 getSplitMode() (*brainstem.entity.Mux method*), 616
 getState() (*brainstem.entity.Digital method*), 600
 getState() (*brainstem.entity.HDBaseT method*), 612
 getState() (*brainstem.entity.Port method*), 638
 getStateAll() (*brainstem.entity.Digital method*), 601
 getStateList() (*brainstem.entity.USBSystem method*), 714
 getStaticIPv4Address() (*brainstem.entity.Ethernet method*), 606
 getStaticIPv4DNSAddress() (*brainstem.entity.Ethernet method*), 607
 getStaticIPv4Gateway() (*brainstem.entity.Ethernet method*), 607
 getStaticIPv4Netmask() (*brainstem.entity.Ethernet method*), 607
 getStatus() (*brainstem.module.Module method*), 584
 getStopBits() (*brainstem.entity.UART method*), 695
 getStreamStatus() (*brainstem.Entity_Entity.Entity method*), 577
 getT2Time() (*brainstem.entity.Signal method*), 673
 getT3Time() (*brainstem.entity.Signal method*), 673
 getTemperature() (*brainstem.entity.Rail method*), 668
 getTemperature() (*brainstem.entity.System method*), 685
 getTotalAccumulatedPower() (*brainstem.entity.PoE method*), 622
 getTotalPower() (*brainstem.entity.PoE method*), 622
 getTransferStore() (*brainstem.entity.Pointer method*), 625
 getTransmitterConfig() (*brainstem.entity.Equalizer method*), 603
 getUnregulatedCurrent() (*brainstem.entity.System method*), 685
 getUnregulatedVoltage() (*brainstem.entity.System method*), 685
 getUpstream() (*brainstem.entity.USBSystem method*), 714

getUpstreamBoostMode() (*brainstem.entity.USB method*), 704
getUpstreamHS() (*brainstem.entity.USBSystem method*), 714
getUpstreamMode() (*brainstem.entity.USB method*), 704
getUpstreamSS() (*brainstem.entity.USBSystem method*), 714
getUpstreamState() (*brainstem.entity.USB method*), 705
getUptime() (*brainstem.entity.System method*), 686
getUSB2DeviceTree() (*brainstem.entity.HDBaseT method*), 612
getUSB3DeviceTree() (*brainstem.entity.HDBaseT method*), 612
getValue() (*brainstem.entity.Analog method*), 594
getValue() (*brainstem.entity.Digital method*), 601
getValue() (*brainstem.entity.Temperature method*), 690
getValueMax() (*brainstem.entity.Temperature method*), 690
getValueMin() (*brainstem.entity.Temperature method*), 690
getVbusAccumulatedPower() (*brainstem.entity.Port method*), 638
getVbusCurrent() (*brainstem.entity.Port method*), 638
getVbusVoltage() (*brainstem.entity.Port method*), 639
getVconn1Enabled() (*brainstem.entity.Port method*), 639
getVconn2Enabled() (*brainstem.entity.Port method*), 639
getVconnAccumulatedPower() (*brainstem.entity.Port method*), 639
getVconnCurrent() (*brainstem.entity.Port method*), 640
getVconnEnabled() (*brainstem.entity.Port method*), 640
getVconnVoltage() (*brainstem.entity.Port method*), 640
getVersion() (*brainstem.entity.System method*), 686
getVoltage() (*brainstem.entity.Analog method*), 594
getVoltage() (*brainstem.entity.Rail method*), 668
getVoltage() (*brainstem.entity.Relay method*), 672
getVoltageMaxLimit() (*brainstem.entity.Rail method*), 668
getVoltageMinLimit() (*brainstem.entity.Rail method*), 668
getVoltageSetpoint() (*brainstem.entity.Port method*), 640
getVoltageSetpoint() (*brainstem.entity.Rail method*), 669
getYear() (*brainstem.entity.Clock method*), 598

H

hasUEI() (*brainstem.module.Module method*), 584
HDBaseT (*class in brainstem.entity*), 610
hdbaset_getCableLength (*C++ function*), 1307
hdbaset_getEncodingState (*C++ function*), 1309
hdbaset_getFirmwareVersion (*C++ function*), 1307
hdbaset_getLinkRole (*C++ function*), 1310
hdbaset_getLinkUtilization (*C++ function*), 1308
hdbaset_getMSEA (*C++ function*), 1307
hdbaset_getMSEB (*C++ function*), 1308
hdbaset_getRetransmissionRate (*C++ function*), 1308
hdbaset_getSerialNumber (*C++ function*), 1306
hdbaset_getState (*C++ function*), 1307
hdbaset_getUSB2DeviceTree (*C++ function*), 1309
hdbaset_getUSB3DeviceTree (*C++ function*), 1309
hdbaset_setLinkRole (*C++ function*), 1310
hdbasetCableLength (*C macro*), 904
hdbasetEncodingState (*C macro*), 904
hdbasetEncodingState_PAM16 (*C macro*), 904
hdbasetEncodingState_PAM4 (*C macro*), 905
hdbasetEncodingState_PAM8 (*C macro*), 904
hdbasetEncodingState_Unknown (*C macro*), 904
hdbasetFirmwareVersion (*C macro*), 903
hdbasetLinkRole (*C macro*), 905
hdbasetLinkRole_AutoDetect (*C macro*), 905
hdbasetLinkRole_Follower (*C macro*), 905
hdbasetLinkRole_Leader (*C macro*), 905
hdbasetLinkUtilization (*C macro*), 904
hdbasetMSEA (*C macro*), 904
hdbasetMSEB (*C macro*), 904
hdbasetNumberOfOptions (*C macro*), 905
hdbasetRetransmissionRate (*C macro*), 904
hdbasetSerialNumber (*C macro*), 903
hdbasetState (*C macro*), 903
hdbasetState_devicePresent_Bit (*C macro*), 903

hdbasetState_linkRole_Follower (*C macro*), 904
 hdbasetState_linkRole_Leader (*C macro*), 904
 hdbasetState_linkRole_Mask (*C macro*), 904
 hdbasetState_linkRole_Offset (*C macro*), 904
 hdbasetState_linkRole_Unknown (*C macro*), 904
 hdbasetState_linkUp_Bit (*C macro*), 904
 hdbasetUSB2DeviceTree (*C macro*), 905
 hdbasetUSB3DeviceTree (*C macro*), 905

I

i2c (*brainstem.stem.USBHub3c attribute*), 554
 I2C (*class in brainstem.entity*), 613
 i2c_getSpeed (*C++ function*), 1312
 i2c_read (*C++ function*), 1310
 i2c_setPullup (*C++ function*), 1311
 i2c_setSpeed (*C++ function*), 1311
 i2c_write (*C++ function*), 1311
 i2cDefaultSpeed (*C macro*), 906
 i2cSetPullup (*C macro*), 906
 i2cSpeed_1000Khz (*C macro*), 907
 i2cSpeed_100Khz (*C macro*), 906
 i2cSpeed_400Khz (*C macro*), 906
 id (*brainstem.module.Module property*), 585
 index (*brainstem.Entity_Entity.Entity property*), 578
 index (*brainstem.pd_channel_logger.PDChannelLogger property*), 587
 initiateBulkCapture () (*brainstem.entity.Analog method*), 594
 initiateTransferFromStore () (*brainstem.entity.Pointer method*), 625
 initiateTransferToStore () (*brainstem.entity.Pointer method*), 626
 INVALID (*brainstem.link.Spec attribute*), 581
 isAtLeast () (*in module brainstem.version*), 590
 isAtLeastCompare () (*in module brainstem.version*), 590
 isConnected () (*brainstem.module.Module method*), 585
 isLegacyFormat () (*in module brainstem.version*), 590
 items () (*brainstem.result.Result method*), 589

K

kelvinSensingOff_Value (*C macro*), 933
 kelvinSensingOn_Value (*C macro*), 934
 key (*brainstem.link.StreamStatusEntry property*), 581
 keys () (*brainstem.result.Result method*), 589

L

link (*brainstem.module.Module property*), 585
 link_enableStream (*C++ function*), 1435
 link_getLinkSpecifier (*C++ function*), 1435
 link_getStreamKeyElement (*C++ function*), 1436
 link_getStreamStatus (*C++ function*), 1436
 link_registerStreamCallback (*C++ function*), 1435
 linkSpec (*C++ struct*), 876
 linkSpec::model (*C++ member*), 876
 linkSpec::module (*C++ member*), 876
 linkSpec::router (*C++ member*), 876
 linkSpec::router_serial_num (*C++ member*), 876
 linkSpec::serial_num (*C++ member*), 876
 linkSpec::t (*C++ member*), 877
 linkSpec::type (*C++ member*), 876
 linkSpec_CCA (*C++ struct*), 1434
 linkSpec_CCA::baudrate (*C++ member*), 1434
 linkSpec_CCA::ip_address (*C++ member*), 1434
 linkSpec_CCA::ip_port (*C++ member*), 1434
 linkSpec_CCA::model (*C++ member*), 1434
 linkSpec_CCA::module (*C++ member*), 1434
 linkSpec_CCA::port (*C++ member*), 1434
 linkSpec_CCA::router (*C++ member*), 1434
 linkSpec_CCA::router_serial_num (*C++ member*), 1434
 linkSpec_CCA::serial_num (*C++ member*), 1434

linkSpec_CCA::type (C++ member), 1434
linkSpec_CCA::usb_id (C++ member), 1434
linkStatus (C++ enum), 885
linkStatus::INITIALIZING (C++ enumerator), 885
linkStatus::INVALID_LINK_STREAM (C++ enumerator), 885
linkStatus::IO_ERROR (C++ enumerator), 885
linkStatus::RESETTING (C++ enumerator), 885
linkStatus::RUNNING (C++ enumerator), 885
linkStatus::STOPPED (C++ enumerator), 885
linkStatus::STOPPING (C++ enumerator), 885
linkStatus::SYNCING (C++ enumerator), 885
linkStatus::UNKNOWN_ERROR (C++ enumerator), 885
linkType (C++ enum), 875
linkType::AETHER (C++ enumerator), 875
linkType::AETHER_SERIAL (C++ enumerator), 876
linkType::AETHER_TCPIP (C++ enumerator), 875
linkType::AETHER_USB (C++ enumerator), 875
linkType::INVALID (C++ enumerator), 875
linkType::SERIAL (C++ enumerator), 875
linkType::TCPIP (C++ enumerator), 875
linkType::USB (C++ enumerator), 875
loadEntityFromSavedValues () (brainstem.Entity.Entity method), 578
loadSlot () (brainstem.entity.Store method), 676
localOnly (brainstem.link.aEtherConfig attribute), 582
logEvents () (brainstem.entity.System method), 686

M

model (brainstem.module.Module property), 585
model_info () (in module brainstem.defs), 574
model_name () (in module brainstem.defs), 574
module
 brainstem.defs, 574
 brainstem.discover, 574
 brainstem.link, 580
 brainstem.module, 582
 brainstem.pd_channel_logger, 586
 brainstem.result, 588
 brainstem.version, 589
module (brainstem.Entity.Entity.Entity property), 578
module (brainstem.pd_channel_logger.PDChannelLogger property), 588
Module (class in brainstem.module), 582
module_clearAllStems (C++ function), 1433
module_connectThroughLinkModule (C++ function), 1432
module_createStem (C++ function), 1431
module_disconnect (C++ function), 1432
module_disconnectAndDestroyStem (C++ function), 1431
module_discoverAndConnect (C++ function), 1432
module_getModuleAddress (C++ function), 1433
module_isConnected (C++ function), 1433
module_reconnect (C++ function), 1432
module_sDiscover (C++ function), 1432
module_setModuleAddress (C++ function), 1433
module_setNetworkingMode (C++ function), 1433
MTMDAQ1 (class in brainstem.stem), 568
MTMDAQ2 (class in brainstem.stem), 560
MTMEtherStem (class in brainstem.stem), 562
MTMIOSerial (class in brainstem.stem), 563
MTMIOSerial.Hub (class in brainstem.stem), 564
MTMLOAD1 (class in brainstem.stem), 564
MTMPM1 (class in brainstem.stem), 565
MTMRelay (class in brainstem.stem), 566
MTMUSBStem (class in brainstem.stem), 567
Mux (class in brainstem.entity), 615
mux_getChannel (C++ function), 1313
mux_getChannelVoltage (C++ function), 1313
mux_getConfiguration (C++ function), 1314
mux_getEnable (C++ function), 1312

mux_getSplitMode (C++ function), 1314
 mux_setChannel (C++ function), 1313
 mux_setConfiguration (C++ function), 1314
 mux_setEnable (C++ function), 1312
 mux_setSplitMode (C++ function), 1314
 muxChannel (C macro), 907
 muxConfig (C macro), 907
 muxConfig_channelpriority (C macro), 907
 muxConfig_default (C macro), 907
 muxConfig_splitMode (C macro), 907
 muxEnable (C macro), 907
 muxNumberOfOptions (C macro), 908
 muxSplit (C macro), 908
 muxVoltage (C macro), 907

N

networkInterface (*brainstem.link.aEtherConfig* attribute), 582

O

OS_NEW_LN (C macro), 874

P

pack () (in module *brainstem.version*), 591
 packDataObjectAttributes () (*brainstem.entity.PowerDelivery* static method), 656
 parseMajor () (in module *brainstem.version*), 591
 parseMinor () (in module *brainstem.version*), 591
 parsePatch () (in module *brainstem.version*), 591
 pdCableCurrent_3Amps (C macro), 926
 pdCableCurrent_5Amps (C macro), 926
 pdCableCurrent_Invalid (C macro), 926
 pdCableOrientation_CC1 (C macro), 927
 pdCableOrientation_CC2 (C macro), 927
 pdCableOrientation_Invalid (C macro), 927
 pdCableSpeed_Invalid (C macro), 927
 pdCableSpeed_USB2p0 (C macro), 927
 pdCableSpeed_USB3p2_Gen1 (C macro), 927
 pdCableSpeed_USB3p2_USB4p0_Gen2 (C macro), 927
 pdCableSpeed_USB4p0_Gen3 (C macro), 927
 pdCableSpeed_USB4p0_Gen4 (C macro), 927
 pdCableType_Active (C macro), 927
 pdCableType_Invalid (C macro), 927
 pdCableType_Passive (C macro), 927
 pdCableVoltage_20VDC (C macro), 926
 pdCableVoltage_30VDC (C macro), 926
 pdCableVoltage_40VDC (C macro), 926
 pdCableVoltage_50VDC (C macro), 926
 pdCableVoltage_Invalid (C macro), 926
 PDChannelLogger (class in *brainstem.pd_channel_logger*), 587
 PDChannelLogger_create (C++ function), 1437
 PDChannelLogger_destroy (C++ function), 1437
 PDChannelLogger_freePayloadBuffer (C++ function), 1439
 PDChannelLogger_getPacket (C++ function), 1438
 PDChannelLogger_getPackets (C++ function), 1438
 PDChannelLogger_setEnabled (C++ function), 1438
 pdConnectionState_AudioAccessory (C macro), 924
 pdConnectionState_DebugAccessory (C macro), 924
 pdConnectionState_None (C macro), 924
 pdConnectionState_PoweredCable (C macro), 924
 pdConnectionState_PoweredCableWithSink (C macro), 924
 pdConnectionState_Sink (C macro), 924
 pdConnectionState_Source (C macro), 924
 pdDataRoleCapabilities_DFP (C macro), 928
 pdDataRoleCapabilities_DualRole (C macro), 928
 pdDataRoleCapabilities_None (C macro), 928
 pdDataRoleCapabilities_UFP (C macro), 928
 pdEventBistEnter (C macro), 932

pdEventBistExit (*C macro*), 932
pdEventBistSharedCapacityEnter (*C macro*), 932
pdEventBistSharedCapacityExit (*C macro*), 932
pdEventCableResetReceived (*C macro*), 931
pdEventCableResetSent (*C macro*), 931
pdEventConnect (*C macro*), 931
pdEventDisconnect (*C macro*), 931
pdEventHardResetReceived (*C macro*), 931
pdEventHardResetSent (*C macro*), 931
pdEventInvalidPacket (*C macro*), 932
pdEventLast (*C macro*), 932
pdEventMessageTransmitDiscarded (*C macro*), 931
pdEventMessageTransmitFailed (*C macro*), 931
pdEventNone (*C macro*), 931
pdEventPacket (*C macro*), 931
pdEventPDFunctionDisabled (*C macro*), 931
pdEventRp1A5 (*C macro*), 931
pdEventRp3A0 (*C macro*), 932
pdEventVBUSDisabled (*C macro*), 931
pdEventVBUSEnabled (*C macro*), 931
pdEventVCONNDisabled (*C macro*), 931
pdEventVCONNEnabled (*C macro*), 931
pdFlagCapabilityMismatch (*C macro*), 930
pdFlagDualRoleData (*C macro*), 930
pdFlagDualRolePower (*C macro*), 930
pdFlagGivebackFlag (*C macro*), 931
pdFlagHigherCapability (*C macro*), 930
pdFlagLast (*C macro*), 931
pdFlagSuspendPossible (*C macro*), 930
pdFlagUnchunkedMessageSupport (*C macro*), 930
pdFlagUnconstrainedPower (*C macro*), 930
pdFlagUSBComPossible (*C macro*), 930
pdLinkState_connectionState_AudioAccessory (*C macro*), 925
pdLinkState_connectionState_DebugAccessory (*C macro*), 925
pdLinkState_connectionState_Mask (*C macro*), 925
pdLinkState_connectionState_None (*C macro*), 925
pdLinkState_connectionState_Offset (*C macro*), 925
pdLinkState_connectionState_PoweredCable (*C macro*), 925
pdLinkState_connectionState_PoweredCableWithSink (*C macro*), 925
pdLinkState_connectionState_Sink (*C macro*), 925
pdLinkState_connectionState_Source (*C macro*), 925
pdLinkState_dataRole_DFP (*C macro*), 925
pdLinkState_dataRole_Mask (*C macro*), 925
pdLinkState_dataRole_None (*C macro*), 925
pdLinkState_dataRole_Offset (*C macro*), 925
pdLinkState_dataRole_UFP (*C macro*), 925
pdLinkState_linkType_EPR (*C macro*), 924
pdLinkState_linkType_Legacy (*C macro*), 924
pdLinkState_linkType_Mask (*C macro*), 924
pdLinkState_linkType_None (*C macro*), 924
pdLinkState_linkType_Offset (*C macro*), 924
pdLinkState_linkType_SPR (*C macro*), 924
pdLinkState_powerRole_Mask (*C macro*), 925
pdLinkState_powerRole_None (*C macro*), 925
pdLinkState_powerRole_Offset (*C macro*), 925
pdLinkState_powerRole_Sink (*C macro*), 925
pdLinkState_powerRole_Source (*C macro*), 925
pdLinkState_specRevision_1 (*C macro*), 926
pdLinkState_specRevision_2 (*C macro*), 926
pdLinkState_specRevision_3 (*C macro*), 926
pdLinkState_specRevision_Mask (*C macro*), 926
pdLinkState_specRevision_Offset (*C macro*), 926
pdLinkState_specRevision_Unknown (*C macro*), 926
pdLinkState_vconnState_Mask (*C macro*), 925
pdLinkState_vconnState_NotSource (*C macro*), 926
pdLinkState_vconnState_Off (*C macro*), 926
pdLinkState_vconnState_Offset (*C macro*), 925
pdLinkState_vconnState_Source (*C macro*), 926

pdOverrideAutoDiscovery (*C macro*), 929
 pdOverrideCableCurrent (*C macro*), 928
 pdOverrideDataPathDisable (*C macro*), 929
 pdOverridePortPower (*C macro*), 928
 pdPowerRole_None (*C macro*), 928
 pdPowerRole_Sink (*C macro*), 928
 pdPowerRole_Source (*C macro*), 928
 pdPowerRole_SourceSink (*C macro*), 928
 pdPowerRoleCapabilities_DualRole (*C macro*), 927
 pdPowerRoleCapabilities_None (*C macro*), 927
 pdPowerRoleCapabilities_Sink (*C macro*), 927
 pdPowerRoleCapabilities_Source (*C macro*), 927
 pdPowerRolePreferred_Auto (*C macro*), 928
 pdPowerRolePreferred_FollowData (*C macro*), 928
 pdPowerRolePreferred_None (*C macro*), 928
 pdPowerRolePreferred_Sink (*C macro*), 928
 pdPowerRolePreferred_Source (*C macro*), 928
 pdProtocolRevision2 (*C macro*), 932
 pdProtocolRevision3 (*C macro*), 932
 pdRequestBatteryCapabilities (*C macro*), 929
 pdRequestBatteryStatus (*C macro*), 929
 pdRequestCountryCode (*C macro*), 930
 pdRequestCountryInfo (*C macro*), 930
 pdRequestDataReset (*C macro*), 929
 pdRequestDataRoleSwap (*C macro*), 929
 pdRequestDiscoverIdentitySop (*C macro*), 930
 pdRequestDiscoverIdentitySopp (*C macro*), 930
 pdRequestDiscoverIdentitySoppo (*C macro*), 930
 pdRequestHardReset (*C macro*), 929
 pdRequestManufacturerInfoSop (*C macro*), 929
 pdRequestManufacturerInfoSopp (*C macro*), 930
 pdRequestManufacturerInfoSoppo (*C macro*), 930
 pdRequestPowerFastRoleSwap (*C macro*), 929
 pdRequestPowerRoleSwap (*C macro*), 929
 pdRequestPPSStatus (*C macro*), 929
 pdRequestRemoteSinkEPRCapabilities (*C macro*), 930
 pdRequestRemoteSinkExtendedCapabilities (*C macro*), 929
 pdRequestRemoteSinkPowerDataObjects (*C macro*), 929
 pdRequestRemoteSourceEPRCapabilities (*C macro*), 930
 pdRequestRemoteSourceExtendedCapabilities (*C macro*), 929
 pdRequestRemoteSourcePowerDataObjects (*C macro*), 929
 pdRequestRevision (*C macro*), 930
 pdRequestSinkGoToMinimum (*C macro*), 929
 pdRequestSoftReset (*C macro*), 929
 pdRequestSourceInfo (*C macro*), 930
 pdRequestStatus (*C macro*), 929
 pdRequestVconnSwap (*C macro*), 929
 PoE (*class in brainstem.entity*), 617
 poe_getPairAccumulatedPower (*C++ function*), 1320
 poe_getPairCapacitance (*C++ function*), 1319
 poe_getPairCurrent (*C++ function*), 1319
 poe_getPairDetectionStatus (*C++ function*), 1318
 poe_getPairDiscoveredClass (*C++ function*), 1317
 poe_getPairEnabled (*C++ function*), 1315
 poe_getPairPower (*C++ function*), 1320
 poe_getPairRequestedClass (*C++ function*), 1317
 poe_getPairResistance (*C++ function*), 1319
 poe_getPairSourcingClass (*C++ function*), 1316
 poe_getPairVoltage (*C++ function*), 1318
 poe_getPowerMode (*C++ function*), 1316
 poe_getPowerState (*C++ function*), 1316
 poe_getTotalAccumulatedPower (*C++ function*), 1321
 poe_getTotalPower (*C++ function*), 1320
 poe_setPairAccumulatedPower (*C++ function*), 1321
 poe_setPairEnabled (*C++ function*), 1315
 poe_setPairSourcingClass (*C++ function*), 1317
 poe_setPowerMode (*C++ function*), 1316
 poe_setTotalAccumulatedPower (*C++ function*), 1321

poeNumberOfOptions (*C macro*), 912
poePairAccumulatedPower (*C macro*), 911
poePairCapacitance (*C macro*), 912
poePairCurrent (*C macro*), 911
poePairDetectionStatus (*C macro*), 911
poePairDetectionStatus_High_Resistance (*C macro*), 911
poePairDetectionStatus_Low_Resistance (*C macro*), 911
poePairDetectionStatus_Open_Circuit (*C macro*), 911
poePairDetectionStatus_Short_Circuit (*C macro*), 911
poePairDetectionStatus_Switch_Failure (*C macro*), 911
poePairDetectionStatus_Unknown (*C macro*), 911
poePairDetectionStatus_Valid (*C macro*), 911
poePairDiscoveredClass (*C macro*), 910
poePairDiscoveredClass_0 (*C macro*), 910
poePairDiscoveredClass_1 (*C macro*), 910
poePairDiscoveredClass_2 (*C macro*), 910
poePairDiscoveredClass_3 (*C macro*), 910
poePairDiscoveredClass_4 (*C macro*), 910
poePairDiscoveredClass_4_PlusType1Limited (*C macro*), 910
poePairDiscoveredClass_5_4PairDualSignature (*C macro*), 910
poePairDiscoveredClass_5_4PairSingleSignature (*C macro*), 910
poePairDiscoveredClass_6_4PairSingleSignature (*C macro*), 911
poePairDiscoveredClass_7_4PairSingleSignature (*C macro*), 911
poePairDiscoveredClass_8_4PairSingleSignature (*C macro*), 911
poePairDiscoveredClass_Mismatch (*C macro*), 911
poePairDiscoveredClass_OverCurrent (*C macro*), 911
poePairDiscoveredClass_Unknown (*C macro*), 910
poePairEnabled (*C macro*), 909
poePairPower (*C macro*), 911
poePairRequestedClass (*C macro*), 909
poePairRequestedClass_0 (*C macro*), 909
poePairRequestedClass_1 (*C macro*), 909
poePairRequestedClass_2 (*C macro*), 910
poePairRequestedClass_3 (*C macro*), 910
poePairRequestedClass_4 (*C macro*), 910
poePairRequestedClass_4_PlusType1Limited (*C macro*), 910
poePairRequestedClass_5_4PairDualSignature (*C macro*), 910
poePairRequestedClass_5_4PairSingleSignature (*C macro*), 910
poePairRequestedClass_6_4PairSingleSignature (*C macro*), 910
poePairRequestedClass_7_4PairSingleSignature (*C macro*), 910
poePairRequestedClass_8_4PairSingleSignature (*C macro*), 910
poePairRequestedClass_Mismatch (*C macro*), 910
poePairRequestedClass_OverCurrent (*C macro*), 910
poePairRequestedClass_Unknown (*C macro*), 909
poePairResistance (*C macro*), 912
poePairSourcingClass (*C macro*), 909
poePairSourcingClass_0 (*C macro*), 909
poePairSourcingClass_1 (*C macro*), 909
poePairSourcingClass_2 (*C macro*), 909
poePairSourcingClass_3 (*C macro*), 909
poePairSourcingClass_4 (*C macro*), 909
poePairSourcingClass_4_PlusType1Limited (*C macro*), 909
poePairSourcingClass_5_4PairDualSignature (*C macro*), 909
poePairSourcingClass_5_4PairSingleSignature (*C macro*), 909
poePairSourcingClass_6_4PairSingleSignature (*C macro*), 909
poePairSourcingClass_7_4PairSingleSignature (*C macro*), 909
poePairSourcingClass_8_4PairSingleSignature (*C macro*), 909
poePairSourcingClass_Mismatch (*C macro*), 909
poePairSourcingClass_OverCurrent (*C macro*), 909
poePairSourcingClass_Unknown (*C macro*), 909
poePairVoltage (*C macro*), 911
poePowerMode (*C macro*), 908
poePowerMode_Auto (*C macro*), 908
poePowerMode_Off (*C macro*), 908
poePowerMode_PD (*C macro*), 908
poePowerMode_PSE (*C macro*), 908
poePowerState (*C macro*), 908
poePowerState_Off (*C macro*), 908

poePowerState_PD (*C macro*), 908
 poePowerState_PSE (*C macro*), 908
 poeTotalAccumulatedPower (*C macro*), 911
 poeTotalPower (*C macro*), 911
 Pointer (*class in brainstem.entity*), 624
 pointer_getChar (*C++ function*), 1324
 pointer_getInt (*C++ function*), 1325
 pointer_getMode (*C++ function*), 1323
 pointer_getOffset (*C++ function*), 1322
 pointer_getShort (*C++ function*), 1325
 pointer_getTransferStore (*C++ function*), 1323
 pointer_initiateTransferFromStore (*C++ function*), 1324
 pointer_initiateTransferToStore (*C++ function*), 1324
 pointer_setChar (*C++ function*), 1324
 pointer_setInt (*C++ function*), 1326
 pointer_setMode (*C++ function*), 1323
 pointer_setOffset (*C++ function*), 1322
 pointer_setShort (*C++ function*), 1325
 pointer_setTransferStore (*C++ function*), 1323
 pointerChar (*C macro*), 912
 pointerInt (*C macro*), 913
 pointerMode (*C macro*), 912
 pointerModeDefault (*C macro*), 912
 pointerModeIncrement (*C macro*), 912
 pointerModeStatic (*C macro*), 912
 pointerOffset (*C macro*), 912
 pointerShort (*C macro*), 913
 pointerTransferFromStore (*C macro*), 913
 pointerTransferStore (*C macro*), 912
 pointerTransferToStore (*C macro*), 913
 Port (*class in brainstem.entity*), 627
 port_getAllocatedPower (*C++ function*), 1340
 port_getAvailablePower (*C++ function*), 1339
 port_getCC1AccumulatedPower (*C++ function*), 1349
 port_getCC1Current (*C++ function*), 1348
 port_getCC1Enabled (*C++ function*), 1335
 port_getCC1State (*C++ function*), 1346
 port_getCC1Voltage (*C++ function*), 1348
 port_getCC2AccumulatedPower (*C++ function*), 1349
 port_getCC2Current (*C++ function*), 1348
 port_getCC2Enabled (*C++ function*), 1336
 port_getCC2State (*C++ function*), 1346
 port_getCC2Voltage (*C++ function*), 1348
 port_getCCCurentLimit (*C++ function*), 1342
 port_getCCEnabled (*C++ function*), 1335
 port_getCurrentLimit (*C++ function*), 1338
 port_getCurrentLimitMode (*C++ function*), 1339
 port_getDataEnabled (*C++ function*), 1328
 port_getDataHS1Enabled (*C++ function*), 1329
 port_getDataHS2Enabled (*C++ function*), 1330
 port_getDataHSEnabled (*C++ function*), 1329
 port_getDataHSRoutingBehavior (*C++ function*), 1342
 port_getDataRole (*C++ function*), 1333
 port_getDataSpeed (*C++ function*), 1337
 port_getDataSS1Enabled (*C++ function*), 1331
 port_getDataSS2Enabled (*C++ function*), 1331
 port_getDataSSEnabled (*C++ function*), 1330
 port_getDataSSRoutingBehavior (*C++ function*), 1343
 port_getEnabled (*C++ function*), 1328
 port_getErrors (*C++ function*), 1338
 port_getHSBoost (*C++ function*), 1346
 port_getMode (*C++ function*), 1337
 port_getName (*C++ function*), 1341
 port_getPowerEnabled (*C++ function*), 1332
 port_getPowerLimit (*C++ function*), 1340
 port_getPowerLimitMode (*C++ function*), 1340
 port_getPowerMode (*C++ function*), 1327
 port_getSBU1Voltage (*C++ function*), 1347

port_getSBU2Voltage (*C++ function*), 1347
port_getState (*C++ function*), 1337
port_getVbusAccumulatedPower (*C++ function*), 1344
port_getVbusCurrent (*C++ function*), 1326
port_getVbusVoltage (*C++ function*), 1326
port_getVconn1Enabled (*C++ function*), 1333
port_getVconn2Enabled (*C++ function*), 1334
port_getVconnAccumulatedPower (*C++ function*), 1344
port_getVconnCurrent (*C++ function*), 1327
port_getVconnEnabled (*C++ function*), 1333
port_getVconnVoltage (*C++ function*), 1327
port_getVoltageSetpoint (*C++ function*), 1336
port_resetVbusAccumulatedPower (*C++ function*), 1344
port_resetVconnAccumulatedPower (*C++ function*), 1345
port_setCC1AccumulatedPower (*C++ function*), 1349
port_setCC1Enabled (*C++ function*), 1335
port_setCC2AccumulatedPower (*C++ function*), 1349
port_setCC2Enabled (*C++ function*), 1336
port_setCCCurrentLimit (*C++ function*), 1342
port_setCCEnabled (*C++ function*), 1335
port_setCurrentLimit (*C++ function*), 1338
port_setCurrentLimitMode (*C++ function*), 1339
port_setDataEnabled (*C++ function*), 1328
port_setDataHS1Enabled (*C++ function*), 1330
port_setDataHS2Enabled (*C++ function*), 1330
port_setDataHSEnabled (*C++ function*), 1329
port_setDataHSRoutingBehavior (*C++ function*), 1343
port_setDataSS1Enabled (*C++ function*), 1331
port_setDataSS2Enabled (*C++ function*), 1332
port_setDataSSEnabled (*C++ function*), 1331
port_setDataSSRoutingBehavior (*C++ function*), 1343
port_setEnabled (*C++ function*), 1328
port_setHSBoost (*C++ function*), 1345
port_setMode (*C++ function*), 1338
port_setName (*C++ function*), 1341
port_setPowerEnabled (*C++ function*), 1332
port_setPowerLimit (*C++ function*), 1340
port_setPowerLimitMode (*C++ function*), 1341
port_setPowerMode (*C++ function*), 1327
port_setVbusAccumulatedPower (*C++ function*), 1344
port_setVconn1Enabled (*C++ function*), 1334
port_setVconn2Enabled (*C++ function*), 1334
port_setVconnAccumulatedPower (*C++ function*), 1345
port_setVconnEnabled (*C++ function*), 1333
port_setVoltageSetpoint (*C++ function*), 1336
PORT_SPEED (*C++ enum*), 976
PORT_SPEED::kPORT_SPEED_FULL (*C++ enumerator*), 976
PORT_SPEED::kPORT_SPEED_HIGH (*C++ enumerator*), 976
PORT_SPEED::kPORT_SPEED_LOW (*C++ enumerator*), 976
PORT_SPEED::kPORT_SPEED_SUPER (*C++ enumerator*), 976
PORT_SPEED::kPORT_SPEED_SUPER_PLUS (*C++ enumerator*), 976
PORT_SPEED::kPORT_SPEED_UNKNOWN (*C++ enumerator*), 976
portAllocatedPower (*C macro*), 920
portAvailablePower (*C macro*), 916
portCC1AccumulatedPower (*C macro*), 923
portCC1Current (*C macro*), 923
portCC1Enabled (*C macro*), 914
portCC1State (*C macro*), 921
portCC1State_Invalid (*C macro*), 921
portCC1State_Managed (*C macro*), 922
portCC1State_None (*C macro*), 921
portCC1State_Ra (*C macro*), 922
portCC1State_Rd (*C macro*), 922
portCC1State_Rp1p5 (*C macro*), 921
portCC1State_Rp3p0 (*C macro*), 921
portCC1State_RpDefault (*C macro*), 921
portCC1State_Unknown (*C macro*), 922
portCC1Voltage (*C macro*), 922

portCC2AccumulatedPower (*C macro*), 923
 portCC2Current (*C macro*), 923
 portCC2Enabled (*C macro*), 914
 portCC2State (*C macro*), 922
 portCC2State_Invalid (*C macro*), 922
 portCC2State_Managed (*C macro*), 922
 portCC2State_None (*C macro*), 922
 portCC2State_Ra (*C macro*), 922
 portCC2State_Rd (*C macro*), 922
 portCC2State_Rp1p5 (*C macro*), 922
 portCC2State_Rp3p0 (*C macro*), 922
 portCC2State_RpDefault (*C macro*), 922
 portCC2State_Unknown (*C macro*), 922
 portCC2Voltage (*C macro*), 923
 portCCCurrentLimit (*C macro*), 916
 portCCCurrentLimit_1p5 (*C macro*), 916
 portCCCurrentLimit_3p0 (*C macro*), 916
 portCCCurrentLimit_Default (*C macro*), 916
 portCCCurrentLimit_None (*C macro*), 916
 portCCEnabled (*C macro*), 914
 portCurrentLimit (*C macro*), 916
 portCurrentLimitMode (*C macro*), 916
 portDataEnabled (*C macro*), 914
 portDataHS1Enabled (*C macro*), 914
 portDataHS2Enabled (*C macro*), 914
 portDataHSEnabled (*C macro*), 914
 portDataHSRoutingBehavior (*C macro*), 920
 portDataHSRoutingBehavior_FollowCC (*C macro*), 920
 portDataHSRoutingBehavior_Shorted (*C macro*), 920
 portDataHSRoutingBehavior_Side1 (*C macro*), 920
 portDataHSRoutingBehavior_Side2 (*C macro*), 920
 portDataRole (*C macro*), 917
 portDataRole_Control_Value (*C macro*), 917
 portDataRole_Disabled_Value (*C macro*), 917
 portDataRole_Downstream_Value (*C macro*), 917
 portDataRole_Upstream_Value (*C macro*), 917
 portDataSpeed (*C macro*), 917
 portDataSpeed_Connected_2p0_Bit (*C macro*), 918
 portDataSpeed_Connected_3p0_Bit (*C macro*), 918
 portDataSpeed_fs_12M_Bit (*C macro*), 917
 portDataSpeed_hs_480M_Bit (*C macro*), 917
 portDataSpeed_ls_1p5M_Bit (*C macro*), 917
 portDataSpeed_ss_10G_Bit (*C macro*), 918
 portDataSpeed_ss_5G_Bit (*C macro*), 917
 portDataSS1Enabled (*C macro*), 914
 portDataSS2Enabled (*C macro*), 914
 portDataSSEnabled (*C macro*), 914
 portDataSSRoutingBehavior (*C macro*), 920
 portDataSSRoutingBehavior_FollowCC (*C macro*), 920
 portDataSSRoutingBehavior_Side1 (*C macro*), 920
 portDataSSRoutingBehavior_Side2 (*C macro*), 920
 portErrors (*C macro*), 916
 portHSBoost (*C macro*), 920
 portHSBoost_m5Percent (*C macro*), 921
 portHSBoost_Nominal (*C macro*), 921
 portHSBoost_p10Percent (*C macro*), 921
 portHSBoost_p15Percent (*C macro*), 921
 portHSBoost_p20Percent (*C macro*), 921
 portHSBoost_p25Percent (*C macro*), 921
 portHSBoost_p30Percent (*C macro*), 921
 portHSBoost_p5Percent (*C macro*), 921
 portMapping_getDownstreamDevices (*C++ function*), 1440
 portName (*C macro*), 916
 portNumberOfOptions (*C macro*), 923
 portPortEnabled (*C macro*), 913
 portPortMode (*C macro*), 918
 portPortMode_CC1Enabled_Bit (*C macro*), 918
 portPortMode_CC2Enabled_Bit (*C macro*), 918

portPortMode_CCFlipEnabled_Bit (C macro), 919
portPortMode_cdp_dcp_Value (C macro), 919
portPortMode_HS1Enabled_Bit (C macro), 918
portPortMode_HS2Enabled_Bit (C macro), 918
portPortMode_HSFlipEnabled_Bit (C macro), 919
portPortMode_KACEEnabled_Bit (C macro), 919
portPortMode_portPowerMode_Mask (C macro), 919
portPortMode_portPowerMode_none_Value (C macro), 919
portPortMode_portPowerMode_Offset (C macro), 919
portPortMode_portPowerMode_pd_Value (C macro), 919
portPortMode_portPowerMode_ps_Value (C macro), 919
portPortMode_portPowerMode_qc_Value (C macro), 919
portPortMode_portPowerMode_sdp_Value (C macro), 919
portPortMode_portPowerMode_usbc_Value (C macro), 919
portPortMode_powerEnabled_Bit (C macro), 918
portPortMode_SBU1Enabled_Bit (C macro), 918
portPortMode_SBU2Enabled_Bit (C macro), 918
portPortMode_SBUFlipEnabled_Bit (C macro), 919
portPortMode_SS1Enabled_Bit (C macro), 918
portPortMode_SS2Enabled_Bit (C macro), 918
portPortMode_SSFlipEnabled_Bit (C macro), 919
portPortMode_Vconn1Enabled_Bit (C macro), 918
portPortMode_Vconn2Enabled_Bit (C macro), 918
portPortState (C macro), 914
portPortState_CC1Enabled_Bit (C macro), 915
portPortState_CC2Enabled_Bit (C macro), 915
portPortState_CC_Flipped_Bit (C macro), 915
portPortState_HS1Enabled_Bit (C macro), 915
portPortState_HS2Enabled_Bit (C macro), 915
portPortState_HS_Flipped_Bit (C macro), 915
portPortState_KACEEnabled_Bit (C macro), 915
portPortState_powerEnabled_Bit (C macro), 914
portPortState_SBU1Enabled_Bit (C macro), 915
portPortState_SBU2Enabled_Bit (C macro), 915
portPortState_SBU_Flipped_Bit (C macro), 915
portPortState_SS1Enabled_Bit (C macro), 915
portPortState_SS2Enabled_Bit (C macro), 915
portPortState_SS_Flipped_Bit (C macro), 915
portPortState_Vconn1Enabled_Bit (C macro), 915
portPortState_Vconn2Enabled_Bit (C macro), 915
portPowerEnabled (C macro), 913
portPowerLimit (C macro), 916
portPowerLimitMode (C macro), 916
portPowerMode (C macro), 916
portPowerMode_cdp_dcp_Value (C macro), 917
portPowerMode_none_Value (C macro), 916
portPowerMode_pd_Value (C macro), 917
portPowerMode_ps_Value (C macro), 917
portPowerMode_qc_Value (C macro), 917
portPowerMode_sdp_Value (C macro), 916
portPowerMode_usbc_Value (C macro), 917
portResetEntityToFactoryDefaults (C macro), 921
portResetVbusAccumulatedPower (C macro), 920
portResetVconnAccumulatedPower (C macro), 920
portSBU1Voltage (C macro), 923
portSBU2Voltage (C macro), 923
portVbusAccumulatedPower (C macro), 920
portVbusCurrent (C macro), 913
portVbusVoltage (C macro), 913
portVconn1Enabled (C macro), 914
portVconn2Enabled (C macro), 914
portVconnAccumulatedPower (C macro), 920
portVconnCurrent (C macro), 913
portVconnEnabled (C macro), 914
portVconnVoltage (C macro), 913
portVoltageSetpoint (C macro), 919
PowerDelivery (class in *brainstem.entity*), 647
powerdelivery_getAttachTimeElapsed (C++ function), 1355

powerdelivery_getCableCurrentMax (*C++ function*), 1358
 powerdelivery_getCableOrientation (*C++ function*), 1359
 powerdelivery_getCableSpeedMax (*C++ function*), 1358
 powerdelivery_getCableType (*C++ function*), 1359
 powerdelivery_getCableVoltageMax (*C++ function*), 1357
 powerdelivery_getConnectionState (*C++ function*), 1350
 powerdelivery_getDataRoleCapabilities (*C++ function*), 1357
 powerdelivery_getFastRoleSwapCurrent (*C++ function*), 1363
 powerdelivery_getFlagMode (*C++ function*), 1362
 powerdelivery_getLinkState (*C++ function*), 1355
 powerdelivery_getNumberOfPowerDataObjects (*C++ function*), 1350
 powerdelivery_getOverride (*C++ function*), 1361
 powerdelivery_getPeakCurrentConfiguration (*C++ function*), 1363
 powerdelivery_getPowerDataObject (*C++ function*), 1351
 powerdelivery_getPowerDataObjectEnabled (*C++ function*), 1352
 powerdelivery_getPowerDataObjectEnabledList (*C++ function*), 1353
 powerdelivery_getPowerDataObjectList (*C++ function*), 1352
 powerdelivery_getPowerRole (*C++ function*), 1356
 powerdelivery_getPowerRoleCapabilities (*C++ function*), 1355
 powerdelivery_getPowerRolePreferred (*C++ function*), 1356
 powerdelivery_getProtocolRevisionMax (*C++ function*), 1361
 powerdelivery_getRequestDataObject (*C++ function*), 1354
 powerdelivery_packDataObjectAttributes (*C++ function*), 1364
 powerdelivery_request (*C++ function*), 1359
 powerdelivery_requestStatus (*C++ function*), 1360
 powerdelivery_resetPowerDataObjectToDefault (*C++ function*), 1352
 powerdelivery_setFastRoleSwapCurrent (*C++ function*), 1364
 powerdelivery_setFlagMode (*C++ function*), 1362
 powerdelivery_setOverride (*C++ function*), 1361
 powerdelivery_setPeakCurrentConfiguration (*C++ function*), 1363
 powerdelivery_setPowerDataObject (*C++ function*), 1351
 powerdelivery_setPowerDataObjectEnabled (*C++ function*), 1353
 powerdelivery_setPowerRole (*C++ function*), 1356
 powerdelivery_setPowerRolePreferred (*C++ function*), 1357
 powerdelivery_setProtocolRevisionMax (*C++ function*), 1361
 powerdelivery_setRequestDataObject (*C++ function*), 1354
 powerdelivery_unpackDataObjectAttributes (*C++ function*), 1365
 powerdeliveryAttachTimeElapsed (*C macro*), 926
 powerdeliveryCableCurrentMax (*C macro*), 926
 powerdeliveryCableOrientation (*C macro*), 927
 powerdeliveryCableSpeedMax (*C macro*), 927
 powerdeliveryCableType (*C macro*), 927
 powerdeliveryCableVoltageMax (*C macro*), 926
 powerdeliveryConnectionState (*C macro*), 924
 powerdeliveryDataRoleCapabilities (*C macro*), 928
 powerdeliveryFastRoleSwapCurrent (*C macro*), 928
 powerdeliveryFlagMode (*C macro*), 930
 powerdeliveryLinkState (*C macro*), 924
 powerdeliveryLogEnable (*C macro*), 931
 powerdeliveryLogEvent (*C macro*), 931
 powerdeliveryLogPacket (*C macro*), 931
 powerdeliveryNumberOfOptions (*C macro*), 932
 powerdeliveryNumberOfPowerDataObjects (*C macro*), 924
 powerdeliveryOverride (*C macro*), 928
 powerdeliveryPartnerLocal (*C macro*), 932
 powerdeliveryPartnerRemote (*C macro*), 932
 powerdeliveryPeakCurrentConfiguration (*C macro*), 928
 powerdeliveryPowerDataObject (*C macro*), 924
 powerdeliveryPowerDataObjectEnabled (*C macro*), 924
 powerdeliveryPowerDataObjectEnabledList (*C macro*), 924
 powerdeliveryPowerDataObjectList (*C macro*), 924
 powerdeliveryPowerRole (*C macro*), 927
 powerdeliveryPowerRoleCapabilities (*C macro*), 927
 powerdeliveryPowerRoleDisabled (*C macro*), 932
 powerdeliveryPowerRolePreferred (*C macro*), 928
 powerdeliveryPowerRoleSink (*C macro*), 932
 powerdeliveryPowerRoleSource (*C macro*), 932
 powerdeliveryPowerRoleSourceSink (*C macro*), 933

powerDeliveryProtocolRevisionMax (*C macro*), 932
powerDeliveryRequestCommand (*C macro*), 929
powerDeliveryRequestDataObject (*C macro*), 924
powerDeliveryRequestStatus (*C macro*), 933
powerDeliveryRequestStatus (*C macro*), 930
powerDeliveryResetPowerDataObjectToDefault (*C macro*), 926
powerDeliveryVDM (*C macro*), 932

R

Rail (*class in brainstem.entity*), 664
rail_clearFaults (*C++ function*), 1376
rail_getCurrent (*C++ function*), 1367
rail_getCurrentLimit (*C++ function*), 1368
rail_getCurrentSetpoint (*C++ function*), 1368
rail_getEnable (*C++ function*), 1369
rail_getKelvinSensingEnable (*C++ function*), 1374
rail_getKelvinSensingState (*C++ function*), 1375
rail_getOperationalMode (*C++ function*), 1375
rail_getOperationalState (*C++ function*), 1376
rail_getPower (*C++ function*), 1372
rail_getPowerLimit (*C++ function*), 1373
rail_getPowerSetpoint (*C++ function*), 1372
rail_getResistance (*C++ function*), 1373
rail_getResistanceSetpoint (*C++ function*), 1374
rail_getTemperature (*C++ function*), 1369
rail_getVoltage (*C++ function*), 1370
rail_getVoltageMaxLimit (*C++ function*), 1371
rail_getVoltageMinLimit (*C++ function*), 1371
rail_getVoltageSetpoint (*C++ function*), 1370
rail_setCurrentLimit (*C++ function*), 1368
rail_setCurrentSetpoint (*C++ function*), 1368
rail_setEnable (*C++ function*), 1369
rail_setKelvinSensingEnable (*C++ function*), 1374
rail_setOperationalMode (*C++ function*), 1375
rail_setPowerLimit (*C++ function*), 1373
rail_setPowerSetpoint (*C++ function*), 1372
rail_setResistanceSetpoint (*C++ function*), 1374
rail_setVoltageMaxLimit (*C++ function*), 1371
rail_setVoltageMinLimit (*C++ function*), 1371
rail_setVoltageSetpoint (*C++ function*), 1370
railClearFaults (*C macro*), 936
railCurrent (*C macro*), 933
railCurrentLimit (*C macro*), 933
railCurrentSetpoint (*C macro*), 936
railEnable (*C macro*), 933
railFactoryReserved (*C macro*), 937
railFactoryReserved2 (*C macro*), 937
railKelvinSensingEnable (*C macro*), 933
railKelvinSensingState (*C macro*), 934
railNumberOfOptions (*C macro*), 937
railOperationalMode (*C macro*), 934
railOperationalMode_HardwareConfiguration_Offset (*C macro*), 934
railOperationalMode_Mode_Offset (*C macro*), 934
railOperationalModeAuto_Value (*C macro*), 934
railOperationalModeConstantCurrent_Value (*C macro*), 934
railOperationalModeConstantPower_Value (*C macro*), 934
railOperationalModeConstantResistance_Value (*C macro*), 934
railOperationalModeConstantVoltage_Value (*C macro*), 934
railOperationalModeFactoryReserved_Value (*C macro*), 934
railOperationalModeLinear_Value (*C macro*), 934
railOperationalModeSwitcher_Value (*C macro*), 934
railOperationalModeSwitcherLinear_Value (*C macro*), 934
railOperationalState (*C macro*), 935
railOperationalState_Enabled_Bit (*C macro*), 935
railOperationalState_Fault_Bit (*C macro*), 935
railOperationalState_HardwareConfiguration_Offset (*C macro*), 935
railOperationalState_Initializing_Bit (*C macro*), 935

railOperationalStateConstantCurrent_Value (*C macro*), 936
 railOperationalStateConstantPower_Value (*C macro*), 936
 railOperationalStateConstantResistance_Value (*C macro*), 936
 railOperationalStateConstantVoltage_Value (*C macro*), 936
 railOperationalStateLinear_Value (*C macro*), 935
 railOperationalStateOperatingMode_Offset (*C macro*), 936
 railOperationalStateOverCurrentFault_Bit (*C macro*), 935
 railOperationalStateOverPowerFault_Bit (*C macro*), 935
 railOperationalStateOverTemperatureFault_Bit (*C macro*), 935
 railOperationalStateOverVoltageFault_Bit (*C macro*), 935
 railOperationalStateReverseCurrentFault_Bit (*C macro*), 935
 railOperationalStateReversePolarityFault_Bit (*C macro*), 935
 railOperationalStateSwitcher_Value (*C macro*), 935
 railOperationalStateSwitcherLinear_Value (*C macro*), 935
 railOperationalStateUnderVoltageFault_Bit (*C macro*), 935
 railPower (*C macro*), 936
 railPowerLimit (*C macro*), 936
 railPowerSetpoint (*C macro*), 936
 railResistance (*C macro*), 936
 railResistanceSetpoint (*C macro*), 936
 railTemperature (*C macro*), 933
 railValue (*C macro*), 933
 railVoltage (*C macro*), 933
 railVoltageMaxLimit (*C macro*), 936
 railVoltageMinLimit (*C macro*), 936
 railVoltageSetpoint (*C macro*), 936
 RCServo (*class in brainstem.entity*), 662
 rcservo_getEnable (*C++ function*), 1366
 rcservo_getPosition (*C++ function*), 1366
 rcservo_getReverse (*C++ function*), 1367
 rcservo_setEnable (*C++ function*), 1365
 rcservo_setPosition (*C++ function*), 1366
 rcservo_setReverse (*C++ function*), 1366
 read () (*brainstem.entity.I2C method*), 613
 reconnect () (*brainstem.module.Module method*), 585
 registerOptionCallback () (*brainstem.Entity_Entity.Entity method*), 578
 Relay (*class in brainstem.entity*), 671
 relay_getEnable (*C++ function*), 1377
 relay_getVoltage (*C++ function*), 1377
 relay_setEnable (*C++ function*), 1376
 relayEnable (*C macro*), 937
 relayNumberOfOptions (*C macro*), 937
 relayVoltage (*C macro*), 937
 request () (*brainstem.entity.PowerDelivery method*), 657
 requestStatus () (*brainstem.entity.PowerDelivery method*), 658
 reset () (*brainstem.entity.System method*), 686
 resetDeviceToFactoryDefaults () (*brainstem.entity.System method*), 687
 resetEntityToFactoryDefaults () (*brainstem.Entity_Entity.Entity method*), 579
 resetPowerDataObjectToDefault () (*brainstem.entity.PowerDelivery method*), 658
 resetVbusAccumulatedPower () (*brainstem.entity.Port method*), 641
 resetVconnAccumulatedPower () (*brainstem.entity.Port method*), 641
 Result (*class in brainstem.result*), 588
 routeToMe () (*brainstem.entity.System method*), 687

S

save () (*brainstem.entity.System method*), 687
 saveEntity () (*brainstem.Entity_Entity.Entity method*), 579
 SERIAL (*brainstem.link.Spec attribute*), 581
 servoEnable (*C macro*), 938
 servoNumberOfOptions (*C macro*), 938
 servoPosition (*C macro*), 938
 servoReverse (*C macro*), 938
 set_UEI16 () (*brainstem.Entity_Entity.Entity method*), 579
 set_UEI16_with_subindex () (*brainstem.Entity_Entity.Entity method*), 579
 set_UEI32 () (*brainstem.Entity_Entity.Entity method*), 579
 set_UEI32_with_subindex () (*brainstem.Entity_Entity.Entity method*), 579
 set_UEI8 () (*brainstem.Entity_Entity.Entity method*), 580

set_UEI8_with_subindex() (*brainstem.Entity_Entity.Entity method*), 580
set_UEIBytes() (*brainstem.Entity_Entity.Entity method*), 580
setAltModeConfig() (*brainstem.entity.USB method*), 705
setBaudRate() (*brainstem.entity.UART method*), 695
setBootSlot() (*brainstem.entity.System method*), 687
setBulkCaptureNumberOfSamples() (*brainstem.entity.Analog method*), 595
setBulkCaptureSampleRate() (*brainstem.entity.Analog method*), 595
setCableFlip() (*brainstem.entity.USB method*), 706
setCC1AccumulatedPower() (*brainstem.entity.Port method*), 641
setCC1Enable() (*brainstem.entity.USB method*), 705
setCC1Enabled() (*brainstem.entity.Port method*), 641
setCC2AccumulatedPower() (*brainstem.entity.Port method*), 641
setCC2Enable() (*brainstem.entity.USB method*), 705
setCC2Enabled() (*brainstem.entity.Port method*), 641
setCCCurrentLimit() (*brainstem.entity.Port method*), 642
setCCEnabled() (*brainstem.entity.Port method*), 642
setChannel() (*brainstem.entity.Mux method*), 616
setChar() (*brainstem.entity.Pointer method*), 626
setConfig() (*brainstem.module.Module method*), 585
setConfiguration() (*brainstem.entity.Analog method*), 595
setConfiguration() (*brainstem.entity.Digital method*), 601
setConfiguration() (*brainstem.entity.Mux method*), 616
setConnectMode() (*brainstem.entity.USB method*), 706
setCurrentLimit() (*brainstem.entity.Port method*), 642
setCurrentLimit() (*brainstem.entity.Rail method*), 669
setCurrentLimitMode() (*brainstem.entity.Port method*), 642
setCurrentSetpoint() (*brainstem.entity.Rail method*), 669
setDataBits() (*brainstem.entity.UART method*), 695
setDataDisable() (*brainstem.entity.USB method*), 706
setDataEnable() (*brainstem.entity.USB method*), 707
setDataEnabled() (*brainstem.entity.Port method*), 643
setDataHS1Enabled() (*brainstem.entity.Port method*), 643
setDataHS2Enabled() (*brainstem.entity.Port method*), 643
setDataHSEnabled() (*brainstem.entity.Port method*), 643
setDataHSMaxDataRate() (*brainstem.entity.USBSystem method*), 715
setDataHSRoutingBehavior() (*brainstem.entity.Port method*), 643
setDataRoleBehavior() (*brainstem.entity.USBSystem method*), 715
setDataRoleBehaviorConfig() (*brainstem.entity.USBSystem method*), 715
setDataSS1Enabled() (*brainstem.entity.Port method*), 644
setDataSS2Enabled() (*brainstem.entity.Port method*), 644
setDataSSEnabled() (*brainstem.entity.Port method*), 644
setDataSSMaxDataRate() (*brainstem.entity.USBSystem method*), 715
setDataSSRoutingBehavior() (*brainstem.entity.Port method*), 644
setDay() (*brainstem.entity.Clock method*), 599
setDownstreamBoostMode() (*brainstem.entity.USB method*), 707
setEnabled() (*brainstem.entity.Analog method*), 595
setEnabled() (*brainstem.entity.Mux method*), 617
setEnabled() (*brainstem.entity.Rail method*), 669
setEnabled() (*brainstem.entity.RCServo method*), 663
setEnabled() (*brainstem.entity.Relay method*), 672
setEnabled() (*brainstem.entity.Signal method*), 674
setEnabled() (*brainstem.entity.UART method*), 695
setEnabled() (*brainstem.entity.Ethernet method*), 608
setEnabled() (*brainstem.entity.Port method*), 644
setEnabled() (*brainstem.pd_channel_logger.PDChannelLogger method*), 588
setEnabledList() (*brainstem.entity.USBSystem method*), 715
setEnumerationDelay() (*brainstem.entity.USB method*), 707
setEnumerationDelay() (*brainstem.entity.USBSystem method*), 716
setExpiration() (*brainstem.entity.Timer method*), 691
setFastRoleSwapCurrent() (*brainstem.entity.PowerDelivery method*), 658
setFlagMode() (*brainstem.entity.PowerDelivery method*), 659
setFlowControl() (*brainstem.entity.UART method*), 696
setHBInterval() (*brainstem.entity.System method*), 687
setHiSpeedDataDisable() (*brainstem.entity.USB method*), 707
setHiSpeedDataEnable() (*brainstem.entity.USB method*), 707
setHostname() (*brainstem.entity.Ethernet method*), 608
setHour() (*brainstem.entity.Clock method*), 599
setHSBoost() (*brainstem.entity.Port method*), 645

setHubMode () (*brainstem.entity.USB method*), 708
 setInputPowerBehavior () (*brainstem.entity.System method*), 688
 setInputPowerBehaviorConfig () (*brainstem.entity.System method*), 688
 setInt () (*brainstem.entity.Pointer method*), 626
 setInterfacePort () (*brainstem.entity.Ethernet method*), 608
 setInvert () (*brainstem.entity.Signal method*), 674
 setKelvinSensingEnable () (*brainstem.entity.Rail method*), 670
 setLED () (*brainstem.entity.System method*), 688
 setLEDMaxBrightness () (*brainstem.entity.System method*), 688
 setLinkChannel () (*brainstem.entity.Digital method*), 602
 setLinkChannel () (*brainstem.entity.UART method*), 696
 setLinkInterface () (*brainstem.entity.System method*), 689
 setLinkRole () (*brainstem.entity.HDBaseT method*), 613
 setMinute () (*brainstem.entity.Clock method*), 599
 setMode () (*brainstem.entity.Pointer method*), 626
 setMode () (*brainstem.entity.Port method*), 645
 setMode () (*brainstem.entity.Timer method*), 692
 setModeList () (*brainstem.entity.USBSystem method*), 716
 setModuleAddress () (*brainstem.module.Module method*), 585
 setModuleSoftwareOffset () (*brainstem.entity.System method*), 689
 setMonth () (*brainstem.entity.Clock method*), 599
 setName () (*brainstem.entity.Port method*), 645
 setName () (*brainstem.entity.System method*), 689
 setNetworkConfiguration () (*brainstem.entity.Ethernet method*), 608
 setNetworkingMode () (*brainstem.module.Module method*), 585
 setOffset () (*brainstem.entity.Pointer method*), 626
 setOperationalMode () (*brainstem.entity.Rail method*), 670
 setOverride () (*brainstem.entity.PowerDelivery method*), 659
 setOverride () (*brainstem.entity.USBSystem method*), 716
 setPairAccumulatedPower () (*brainstem.entity.PoE method*), 622
 setPairEnabled () (*brainstem.entity.PoE method*), 623
 setPairSourcingClass () (*brainstem.entity.PoE method*), 623
 setParity () (*brainstem.entity.UART method*), 696
 setPeakCurrentConfiguration () (*brainstem.entity.PowerDelivery method*), 659
 setPortCurrentLimit () (*brainstem.entity.USB method*), 708
 setPortDisable () (*brainstem.entity.USB method*), 708
 setPortEnable () (*brainstem.entity.USB method*), 708
 setPortMode () (*brainstem.entity.USB method*), 708
 setPosition () (*brainstem.entity.RCServo method*), 663
 setPowerBehavior () (*brainstem.entity.USBSystem method*), 716
 setPowerBehaviorConfig () (*brainstem.entity.USBSystem method*), 717
 setPowerDataObject () (*brainstem.entity.PowerDelivery method*), 660
 setPowerDataObjectEnabled () (*brainstem.entity.PowerDelivery method*), 660
 setPowerDisable () (*brainstem.entity.USB method*), 709
 setPowerEnable () (*brainstem.entity.USB method*), 709
 setPowerEnabled () (*brainstem.entity.Port method*), 645
 setPowerLimit () (*brainstem.entity.Port method*), 645
 setPowerLimit () (*brainstem.entity.Rail method*), 670
 setPowerLimitMax () (*brainstem.entity.System method*), 689
 setPowerLimitMode () (*brainstem.entity.Port method*), 646
 setPowerMode () (*brainstem.entity.PoE method*), 623
 setPowerMode () (*brainstem.entity.Port method*), 646
 setPowerRole () (*brainstem.entity.PowerDelivery method*), 660
 setPowerRolePreferred () (*brainstem.entity.PowerDelivery method*), 661
 setPowerSetpoint () (*brainstem.entity.Rail method*), 670
 setProtocol () (*brainstem.entity.UART method*), 696
 setProtocolRevisionMax () (*brainstem.entity.PowerDelivery method*), 661
 setPullup () (*brainstem.entity.I2C method*), 614
 setRange () (*brainstem.entity.Analog method*), 595
 setReceiverConfig () (*brainstem.entity.Equalizer method*), 603
 setRequestDataObject () (*brainstem.entity.PowerDelivery method*), 661
 setResistanceSetpoint () (*brainstem.entity.Rail method*), 670
 setReverse () (*brainstem.entity.RCServo method*), 663
 setRouter () (*brainstem.entity.System method*), 690
 setSBUEnable () (*brainstem.entity.USB method*), 709
 setSecond () (*brainstem.entity.Clock method*), 599
 setSelectorMode () (*brainstem.entity.USBSystem method*), 717
 setShort () (*brainstem.entity.Pointer method*), 627

setSlotLocked() (*brainstem.entity.Store method*), 676
setSpeed() (*brainstem.entity.I2C method*), 614
setSplitMode() (*brainstem.entity.Mux method*), 617
setState() (*brainstem.entity.Digital method*), 602
setStateAll() (*brainstem.entity.Digital method*), 602
setStaticIPv4Address() (*brainstem.entity.Ethernet method*), 609
setStaticIPv4DNSAddress() (*brainstem.entity.Ethernet method*), 609
setStaticIPv4Gateway() (*brainstem.entity.Ethernet method*), 609
setStaticIPv4Netmask() (*brainstem.entity.Ethernet method*), 609
setStopBits() (*brainstem.entity.UART method*), 697
setStreamEnabled() (*brainstem.Entity_Entity.Entity method*), 579
setSuperSpeedDataDisable() (*brainstem.entity.USB method*), 709
setSuperSpeedDataEnable() (*brainstem.entity.USB method*), 709
setT2Time() (*brainstem.entity.Signal method*), 674
setT3Time() (*brainstem.entity.Signal method*), 674
setTotalAccumulatedPower() (*brainstem.entity.PoE method*), 624
setTransferStore() (*brainstem.entity.Pointer method*), 627
setTransmitterConfig() (*brainstem.entity.Equalizer method*), 604
setUpstream() (*brainstem.entity.USBSystem method*), 717
setUpstreamBoostMode() (*brainstem.entity.USB method*), 710
setUpstreamHS() (*brainstem.entity.USBSystem method*), 717
setUpstreamMode() (*brainstem.entity.USB method*), 710
setUpstreamSS() (*brainstem.entity.USBSystem method*), 717
setValue() (*brainstem.entity.Analog method*), 596
setValue() (*brainstem.entity.Digital method*), 602
setVbusAccumulatedPower() (*brainstem.entity.Port method*), 646
setVconn1Enabled() (*brainstem.entity.Port method*), 646
setVconn2Enabled() (*brainstem.entity.Port method*), 646
setVconnAccumulatedPower() (*brainstem.entity.Port method*), 647
setVconnEnabled() (*brainstem.entity.Port method*), 647
setVoltage() (*brainstem.entity.Analog method*), 596
setVoltageMaxLimit() (*brainstem.entity.Rail method*), 671
setVoltageMinLimit() (*brainstem.entity.Rail method*), 671
setVoltageSetpoint() (*brainstem.entity.Port method*), 647
setVoltageSetpoint() (*brainstem.entity.Rail method*), 671
setYear() (*brainstem.entity.Clock method*), 600
Signal (*class in brainstem.entity*), 672
signal_getEnable (*C++ function*), 1378
signal_getInvert (*C++ function*), 1378
signal_getT2Time (*C++ function*), 1379
signal_getT3Time (*C++ function*), 1379
signal_setEnable (*C++ function*), 1377
signal_setInvert (*C++ function*), 1378
signal_setT2Time (*C++ function*), 1379
signal_setT3Time (*C++ function*), 1379
signalEnable (*C macro*), 939
signalInvert (*C macro*), 939
signalNumberOfOptions (*C macro*), 939
signalT2Time (*C macro*), 939
signalT3Time (*C macro*), 939
slotCapacity (*C macro*), 939
slotClose (*C macro*), 940
slotDisable() (*brainstem.entity.Store method*), 676
slotEnable() (*brainstem.entity.Store method*), 677
slotNumberOfOptions (*C macro*), 940
slotOpenRead (*C macro*), 939
slotOpenWrite (*C macro*), 939
slotRead (*C macro*), 940
slotSeek (*C macro*), 939
slotSize (*C macro*), 939
slotWrite (*C macro*), 940
Spec (*class in brainstem.link*), 580
Status (*class in brainstem.link*), 581
Store (*class in brainstem.entity*), 675
store_getSlotCapacity (*C++ function*), 1382
store_getSlotLocked (*C++ function*), 1382
store_getSlotSize (*C++ function*), 1382
store_getSlotState (*C++ function*), 1380

store_loadSlot (*C++ function*), 1380
 store_setSlotLocked (*C++ function*), 1383
 store_slotDisable (*C++ function*), 1381
 store_slotEnable (*C++ function*), 1381
 store_unloadSlot (*C++ function*), 1381
 storeCloseSlot (*C macro*), 941
 storeEEPROMStore (*C macro*), 941
 storeInternalStore (*C macro*), 941
 storeLock (*C macro*), 941
 storeMaxStoreIndex (*C macro*), 941
 storeNumberOfOptions (*C macro*), 941
 storeRAMStore (*C macro*), 941
 storeReadSlot (*C macro*), 941
 storeSDStore (*C macro*), 941
 storeSlotDisable (*C macro*), 940
 storeSlotEnable (*C macro*), 940
 storeSlotState (*C macro*), 940
 storeWriteSlot (*C macro*), 941
 streamCapacity (*C macro*), 942
 streamDisable (*C macro*), 942
 streamEnable (*C macro*), 942
 streamNumberOfOptions (*C macro*), 942
 StreamStatusEntry (*class in brainstem.link*), 581
 StreamStatusEntry_CCA (*C++ struct*), 1434
 StreamStatusEntry_CCA::key (*C++ member*), 1435
 StreamStatusEntry_CCA::value (*C++ member*), 1435
 subClassQuantity () (*brainstem.module.Module method*), 585
 System (*class in brainstem.entity*), 677
 system_getBootSlot (*C++ function*), 1386
 system_getBuild (*C++ function*), 1387
 system_getErrors (*C++ function*), 1397
 system_getHardwareVersion (*C++ function*), 1388
 system_getHBInterval (*C++ function*), 1385
 system_getInputCurrent (*C++ function*), 1390
 system_getInputPowerBehavior (*C++ function*), 1394
 system_getInputPowerBehaviorConfig (*C++ function*), 1395
 system_getInputPowerSource (*C++ function*), 1394
 system_getInputVoltage (*C++ function*), 1390
 system_getLED (*C++ function*), 1385
 system_getLEDMaxBrightness (*C++ function*), 1386
 system_getLinkInterface (*C++ function*), 1396
 system_getMaximumTemperature (*C++ function*), 1390
 system_getMinimumTemperature (*C++ function*), 1390
 system_getModel (*C++ function*), 1387
 system_getModule (*C++ function*), 1383
 system_getModuleBaseAddress (*C++ function*), 1383
 system_getModuleHardwareOffset (*C++ function*), 1391
 system_getModuleSoftwareOffset (*C++ function*), 1391
 system_getName (*C++ function*), 1395
 system_getPowerLimit (*C++ function*), 1392
 system_getPowerLimitMax (*C++ function*), 1392
 system_getPowerLimitState (*C++ function*), 1393
 system_getProtocolFeatures (*C++ function*), 1397
 system_getRouter (*C++ function*), 1384
 system_getRouterAddressSetting (*C++ function*), 1392
 system_getSerialNumber (*C++ function*), 1388
 system_getTemperature (*C++ function*), 1389
 system_getUnregulatedCurrent (*C++ function*), 1394
 system_getUnregulatedVoltage (*C++ function*), 1393
 system_getUptime (*C++ function*), 1389
 system_getVersion (*C++ function*), 1387
 system_logEvents (*C++ function*), 1389
 system_reset (*C++ function*), 1389
 system_resetDeviceToFactoryDefaults (*C++ function*), 1396
 system_routeToMe (*C++ function*), 1392
 system_save (*C++ function*), 1388
 system_setBootSlot (*C++ function*), 1386
 system_setHBInterval (*C++ function*), 1384

system_setInputPowerBehavior (C++ function), 1394
system_setInputPowerBehaviorConfig (C++ function), 1395
system_setLED (C++ function), 1385
system_setLEDMaxBrightness (C++ function), 1385
system_setLinkInterface (C++ function), 1397
system_setModuleSoftwareOffset (C++ function), 1391
system_setName (C++ function), 1396
system_setPowerLimitMax (C++ function), 1393
system_setRouter (C++ function), 1384
systemBootSlot (C macro), 943
systemBuild (C macro), 946
systemErrors (C macro), 946
systemErrors_OutputPowerProtection_Bit (C macro), 946
systemErrors_ThermalProtection_Bit (C macro), 946
systemHardwareVersion (C macro), 946
systemHBInterval (C macro), 943
systemInputCurrent (C macro), 944
systemInputPowerBehavior (C macro), 945
systemInputPowerBehaviorConfig (C macro), 945
systemInputPowerSource (C macro), 945
systemInputVoltage (C macro), 943
systemIPAddress (C macro), 944
systemIPConfiguration (C macro), 944
systemIPModeDefault (C macro), 944
systemIPModeDHCP (C macro), 944
systemIPModeStatic (C macro), 944
systemIPStaticAddressSetting (C macro), 944
systemLED (C macro), 943
systemLEDMaxBrightness (C macro), 946
systemLinkAuto (C macro), 945
systemLinkInterface (C macro), 945
systemLinkUSBAll (C macro), 945
systemLinkUSBControl (C macro), 945
systemLinkUSBHub (C macro), 945
systemLogEvents (C macro), 944
systemMaxTemperature (C macro), 944
systemMinTemperature (C macro), 945
systemModel (C macro), 943
systemModule (C macro), 943
systemModuleBaseAddress (C macro), 943
systemModuleHardwareOffset (C macro), 943
systemModuleSoftwareOffset (C macro), 943
systemName (C macro), 945
systemNumberOfOptions (C macro), 946
systemPowerLimit (C macro), 945
systemPowerLimitMax (C macro), 945
systemPowerLimitState (C macro), 945
systemProtocolFeatures (C macro), 946
systemProtocolFeatures_pooledPackets_Bit (C macro), 946
systemReserved (C macro), 945
systemReset (C macro), 943
systemResetDeviceToFactoryDefaults (C macro), 945
systemResetEntityToFactoryDefaults (C macro), 945
systemRouter (C macro), 943
systemRouterAddressSetting (C macro), 944
systemRouteToMe (C macro), 944
systemSave (C macro), 943
systemSerialNumber (C macro), 943
systemSleep (C macro), 943
systemTemperature (C macro), 944
systemUnregulatedCurrent (C macro), 944
systemUnregulatedVoltage (C macro), 944
systemUptime (C macro), 944
systemVersion (C macro), 943

T

TCP/IP (*brainstem.link.Spec attribute*), 581

Temperature (*class in brainstem.entity*), 690
 temperature_getValue (*C++ function*), 1398
 temperature_getValueMax (*C++ function*), 1398
 temperature_getValueMin (*C++ function*), 1398
 temperatureMaximumMicroCelsius (*C macro*), 947
 temperatureMicroCelsius (*C macro*), 947
 temperatureMinimumMicroCelsius (*C macro*), 947
 temperatureNumberOfOptions (*C macro*), 947
 temperatureResetLoggedValues (*C macro*), 947
 Timer (*class in brainstem.entity*), 691
 timer_getExpiration (*C++ function*), 1399
 timer_getMode (*C++ function*), 1399
 timer_setExpiration (*C++ function*), 1399
 timer_setMode (*C++ function*), 1400
 timerExpiration (*C macro*), 947
 timerMode (*C macro*), 947
 timerModeRepeat (*C macro*), 947
 timerModeSingle (*C macro*), 947
 timerNumberOfOptions (*C macro*), 948

U

UART (*class in brainstem.entity*), 692
 uart_getAvailableProtocols (*C++ function*), 1406
 uart_getBaudRate (*C++ function*), 1401
 uart_getCapableProtocols (*C++ function*), 1405
 uart_getDataBits (*C++ function*), 1404
 uart_getEnable (*C++ function*), 1400
 uart_getFlowControl (*C++ function*), 1405
 uart_getLinkChannel (*C++ function*), 1402
 uart_getParity (*C++ function*), 1403
 uart_getProtocol (*C++ function*), 1401
 uart_getStopBits (*C++ function*), 1403
 uart_setBaudRate (*C++ function*), 1401
 uart_setDataBits (*C++ function*), 1404
 uart_setEnable (*C++ function*), 1400
 uart_setFlowControl (*C++ function*), 1404
 uart_setLinkChannel (*C++ function*), 1402
 uart_setParity (*C++ function*), 1403
 uart_setProtocol (*C++ function*), 1401
 uart_setStopBits (*C++ function*), 1402
 uartAvailableProtocols (*C macro*), 950
 uartBaudRate (*C macro*), 948
 uartBaudRate_Auto_Value (*C macro*), 948
 uartCapableProtocols (*C macro*), 950
 uartDataBits (*C macro*), 950
 uartEnable (*C macro*), 948
 uartFlowControl (*C macro*), 950
 uartFlowControl_DSR_DTR_Bit (*C macro*), 950
 uartFlowControl_RTS_CTS_Bit (*C macro*), 950
 uartFlowControl_XON_XOFF_Bit (*C macro*), 950
 uartLinkChannel (*C macro*), 949
 uartNumberOfOptions (*C macro*), 950
 uartParity (*C macro*), 949
 uartParity_Even_Value (*C macro*), 949
 uartParity_Mark_Value (*C macro*), 949
 uartParity_None_Value (*C macro*), 949
 uartParity_Odd_Value (*C macro*), 949
 uartParity_Space_Value (*C macro*), 949
 uartProtocol (*C macro*), 948
 uartProtocol_Brainstem_Value (*C macro*), 948
 uartProtocol_Extron_Value (*C macro*), 948
 uartProtocol_ExtronInitiator_Value (*C macro*), 948
 uartProtocol_ExtronResponder_Value (*C macro*), 948
 uartProtocol_Loopback_Value (*C macro*), 949
 uartProtocol_Reserved4_Value (*C macro*), 949
 uartProtocol_Reserved5_Value (*C macro*), 949
 uartProtocol_RestSerial_Value (*C macro*), 949

uartProtocol_Undefined (*C macro*), 948
uartStopBits (*C macro*), 949
uartStopBits_1_Value (*C macro*), 949
uartStopBits_1p5_Value (*C macro*), 949
uartStopBits_2_Value (*C macro*), 949
uei (*C++ struct*), 972
uei::byteVal (*C++ member*), 973
uei::command (*C++ member*), 973
uei::intVal (*C++ member*), 973
uei::module (*C++ member*), 973
uei::option (*C++ member*), 973
uei::shortVal (*C++ member*), 973
uei::specifier (*C++ member*), 973
uei::type (*C++ member*), 973
ueiBYTES_CONTINUE (*C macro*), 893
ueiBYTES_CONTINUE_MASK (*C macro*), 893
ueiOPTION_ACK (*C macro*), 893
ueiOPTION_GET (*C macro*), 893
ueiOPTION_MASK (*C macro*), 893
ueiOPTION_OP_MASK (*C macro*), 893
ueiOPTION_SET (*C macro*), 893
ueiOPTION_VAL (*C macro*), 893
ueiREPLY_ERROR (*C macro*), 892
ueiREPLY_STREAM (*C macro*), 893
ueiSPECIFIER_INDEX_MASK (*C macro*), 892
ueiSPECIFIER_RETURN_HOST (*C macro*), 892
ueiSPECIFIER_RETURN_I2C (*C macro*), 892
ueiSPECIFIER_RETURN_MASK (*C macro*), 892
ueiSPECIFIER_RETURN_VM (*C macro*), 892
unloadSlot () (*brainstem.entity.Store method*), 677
unpack_version () (*in module brainstem.version*), 592
unpackDataObjectAttributes () (*brainstem.entity.PowerDelivery static method*), 661
USB (*brainstem.link.Spec attribute*), 581
USB (*class in brainstem.entity*), 697
usb_clearPortErrorStatus (*C++ function*), 1410
usb_getAltModeConfig (*C++ function*), 1421
usb_getCableFlip (*C++ function*), 1420
usb_getCC1Current (*C++ function*), 1418
usb_getCC1Enable (*C++ function*), 1417
usb_getCC1Voltage (*C++ function*), 1418
usb_getCC2Current (*C++ function*), 1418
usb_getCC2Enable (*C++ function*), 1417
usb_getCC2Voltage (*C++ function*), 1419
usb_getConnectMode (*C++ function*), 1416
usb_getDownstreamBoostMode (*C++ function*), 1415
usb_getDownstreamDataSpeed (*C++ function*), 1415
usb_getEnumerationDelay (*C++ function*), 1412
usb_getHubMode (*C++ function*), 1410
usb_getPortCurrent (*C++ function*), 1409
usb_getPortCurrentLimit (*C++ function*), 1412
usb_getPortError (*C++ function*), 1414
usb_getPortMode (*C++ function*), 1413
usb_getPortState (*C++ function*), 1413
usb_getPortVoltage (*C++ function*), 1409
usb_getSBU1Voltage (*C++ function*), 1421
usb_getSBU2Voltage (*C++ function*), 1421
usb_getSBUEnable (*C++ function*), 1419
usb_getUpstreamBoostMode (*C++ function*), 1415
usb_getUpstreamMode (*C++ function*), 1410
usb_getUpstreamState (*C++ function*), 1411
usb_setAltModeConfig (*C++ function*), 1420
usb_setCableFlip (*C++ function*), 1420
usb_setCC1Enable (*C++ function*), 1416
usb_setCC2Enable (*C++ function*), 1417
usb_setConnectMode (*C++ function*), 1416
usb_setDataDisable (*C++ function*), 1407
usb_setDataEnable (*C++ function*), 1407
usb_setDownstreamBoostMode (*C++ function*), 1414

usb_setEnumerationDelay (C++ function), 1411
 usb_setHiSpeedDataDisable (C++ function), 1407
 usb_setHiSpeedDataEnable (C++ function), 1407
 usb_setHubMode (C++ function), 1410
 usb_setPortCurrentLimit (C++ function), 1412
 usb_setPortDisable (C++ function), 1406
 usb_setPortEnable (C++ function), 1406
 usb_setPortMode (C++ function), 1412
 usb_setPowerDisable (C++ function), 1409
 usb_setPowerEnable (C++ function), 1408
 usb_setSBUEnable (C++ function), 1419
 usb_setSuperSpeedDataDisable (C++ function), 1408
 usb_setSuperSpeedDataEnable (C++ function), 1408
 usb_setUpstreamBoostMode (C++ function), 1414
 usb_setUpstreamMode (C++ function), 1411
 usbAltMode (C macro), 955
 usbAltMode_2LaneDP_ComToHost_wUSB3 (C macro), 956
 usbAltMode_2LaneDP_ComToHost_wUSB3_Inverted (C macro), 956
 usbAltMode_2LaneDP_MuxToHost_wUSB3 (C macro), 956
 usbAltMode_2LaneDP_MuxToHost_wUSB3_Inverted (C macro), 956
 usbAltMode_4LaneDP_ComToHost (C macro), 955
 usbAltMode_4LaneDP_MuxToHost (C macro), 956
 usbAltMode_disabled (C macro), 955
 usbAltMode_normal (C macro), 955
 usbAltMode_USB4_ComToHost (C macro), 956
 usbAltMode_USB4_ComToHost_Inverted (C macro), 956
 usbAltMode_USB4_MuxToHost (C macro), 956
 usbAltMode_USB4_MuxToHost_Inverted (C macro), 956
 usbAutoConnect (C macro), 955
 usbBoostMode_0 (C macro), 952
 usbBoostMode_12 (C macro), 952
 usbBoostMode_4 (C macro), 952
 usbBoostMode_8 (C macro), 952
 usbCableFlip (C macro), 955
 usbCC1Current (C macro), 955
 usbCC1Enable (C macro), 955
 usbCC1Voltage (C macro), 955
 usbCC2Current (C macro), 955
 usbCC2Enable (C macro), 955
 usbCC2Voltage (C macro), 955
 usbConnectMode (C macro), 954
 USBSwitch (class in *brainstem.stem*), 558
 USBSwitchPro (class in *brainstem.stem*), 557
 usbDataDisable (C macro), 951
 usbDataEnable (C macro), 951
 usbDownstreamBoostMode (C macro), 952
 usbDownstreamDataSpeed (C macro), 954
 usbDownstreamDataSpeed_hs (C macro), 954
 usbDownstreamDataSpeed_ls (C macro), 954
 usbDownstreamDataSpeed_na (C macro), 954
 usbDownstreamDataSpeed_ss (C macro), 954
 usbHiSpeedDataDisable (C macro), 954
 usbHiSpeedDataEnable (C macro), 954
 USBHub2x4 (class in *brainstem.stem*), 555
 USBHub2x4.Hub (class in *brainstem.stem*), 556
 USBHub3c (class in *brainstem.stem*), 553
 USBHub3c.Hub (class in *brainstem.stem*), 553
 USBHub3p (class in *brainstem.stem*), 554
 USBHub3p.Hub (class in *brainstem.stem*), 555
 usbHubEnumerationDelay (C macro), 952
 usbHubMode (C macro), 951
 usbManualConnect (C macro), 954
 usbNumberOfOptions (C macro), 956
 usbPortClearErrorStatus (C macro), 951
 usbPortCurrent (C macro), 951
 usbPortCurrentLimit (C macro), 952
 usbPortDisable (C macro), 951
 usbPortEnable (C macro), 951

usbPortError (*C macro*), 955
usbPortMode (*C macro*), 952
usbPortMode_AutoConnectEnable (*C macro*), 953
usbPortMode_CC1Enable (*C macro*), 953
usbPortMode_CC1InjectEnable (*C macro*), 954
usbPortMode_CC2Enable (*C macro*), 953
usbPortMode_CC2InjectEnable (*C macro*), 954
usbPortMode_CCFFlipEnable (*C macro*), 953
usbPortMode_cdp (*C macro*), 952
usbPortMode_charging (*C macro*), 953
usbPortMode_passive (*C macro*), 953
usbPortMode_SBUEnable (*C macro*), 953
usbPortMode_SBUFlipEnable (*C macro*), 954
usbPortMode_sdp (*C macro*), 952
usbPortMode_SSFlipEnable (*C macro*), 953
usbPortMode_SuperSpeed1Enable (*C macro*), 953
usbPortMode_SuperSpeed2Enable (*C macro*), 953
usbPortMode_USB2AEnable (*C macro*), 953
usbPortMode_USB2BEnable (*C macro*), 953
usbPortMode_USB2BoostEnable (*C macro*), 953
usbPortMode_USB2FlipEnable (*C macro*), 954
usbPortMode_USB3BoostEnable (*C macro*), 953
usbPortMode_VBusEnable (*C macro*), 953
usbPortState (*C macro*), 955
usbPortStateCC1 (*C macro*), 735
usbPortStateCC1Detect (*C macro*), 736
usbPortStateCC1Inject (*C macro*), 735
usbPortStateCC1LogicState (*C macro*), 736
usbPortStateCC2 (*C macro*), 735
usbPortStateCC2Detect (*C macro*), 736
usbPortStateCC2Inject (*C macro*), 735
usbPortStateCC2LogicState (*C macro*), 736
usbPortStateCCFlip (*C macro*), 735
usbPortStateConnectionEstablished (*C macro*), 735
usbPortStateSBU (*C macro*), 735
usbPortStateSBUFlip (*C macro*), 735
usbPortStateSS1 (*C macro*), 735
usbPortStateSS2 (*C macro*), 735
usbPortStateSSFlip (*C macro*), 735
usbPortStateUSB2A (*C macro*), 735
usbPortStateUSB2B (*C macro*), 735
usbPortStateUSB2Flip (*C macro*), 735
usbPortStateVBUS (*C macro*), 735
usbPortVoltage (*C macro*), 951
usbPowerDisable (*C macro*), 951
usbPowerEnable (*C macro*), 951
usbSBU1Voltage (*C macro*), 956
usbSBU2Voltage (*C macro*), 956
usbSBUEnable (*C macro*), 955
USBStem (*class in brainstem.stem*), 571
usbSuperSpeedDataDisable (*C macro*), 954
usbSuperSpeedDataEnable (*C macro*), 954
USBSysstem (*class in brainstem.entity*), 710
usbsystem_getDataHSMaxDataRate (*C++ function*), 1430
usbsystem_getDataRoleBehavior (*C++ function*), 1426
usbsystem_getDataRoleBehaviorConfig (*C++ function*), 1427
usbsystem_getDataRoleList (*C++ function*), 1423
usbsystem_getDataSSMaxDataRate (*C++ function*), 1431
usbsystem_getEnabledList (*C++ function*), 1423
usbsystem_getEnumerationDelay (*C++ function*), 1423
usbsystem_getModeList (*C++ function*), 1424
usbsystem_getOverride (*C++ function*), 1429
usbsystem_getPowerBehavior (*C++ function*), 1425
usbsystem_getPowerBehaviorConfig (*C++ function*), 1426
usbsystem_getSelectorMode (*C++ function*), 1428
usbsystem_getStateList (*C++ function*), 1425
usbsystem_getUpstream (*C++ function*), 1422
usbsystem_getUpstreamHS (*C++ function*), 1428

usbsystem_getUpstreamSS (*C++ function*), 1429
 usbsystem_setDataHSMMaxDataRate (*C++ function*), 1430
 usbsystem_setDataRoleBehavior (*C++ function*), 1427
 usbsystem_setDataRoleBehaviorConfig (*C++ function*), 1427
 usbsystem_setDataSSMaxDataRate (*C++ function*), 1430
 usbsystem_setEnabledList (*C++ function*), 1424
 usbsystem_setEnumerationDelay (*C++ function*), 1423
 usbsystem_setModeList (*C++ function*), 1424
 usbsystem_setOverride (*C++ function*), 1430
 usbsystem_setPowerBehavior (*C++ function*), 1425
 usbsystem_setPowerBehaviorConfig (*C++ function*), 1426
 usbsystem_setSelectorMode (*C++ function*), 1428
 usbsystem_setUpstream (*C++ function*), 1422
 usbsystem_setUpstreamHS (*C++ function*), 1428
 usbsystem_setUpstreamSS (*C++ function*), 1429
 usbsystemDataBehavior (*C macro*), 958
 usbsystemDataBehavior_FollowPD (*C macro*), 958
 usbsystemDataBehavior_HardCoded (*C macro*), 958
 usbsystemDataBehavior_LastConnected (*C macro*), 958
 usbsystemDataBehavior_PortPriority (*C macro*), 958
 usbsystemDataBehaviorConfig (*C macro*), 958
 usbsystemDataHSMMaxDataRate (*C macro*), 959
 usbsystemDataHSMMaxDataRate_FullSpeed (*C macro*), 959
 usbsystemDataHSMMaxDataRate_HighSpeed (*C macro*), 959
 usbsystemDataHSMMaxDataRate_LowSpeed (*C macro*), 959
 usbsystemDataHSMMaxDataRate_None (*C macro*), 959
 usbsystemDataRoleList (*C macro*), 957
 usbsystemDataSSMaxDataRate (*C macro*), 959
 usbsystemDataSSMaxDataRate_None (*C macro*), 959
 usbsystemDataSSMaxDataRate_SuperSpeed (*C macro*), 959
 usbsystemDataSSMaxDataRate_SuperSpeedPlus (*C macro*), 959
 usbsystemEnabledList (*C macro*), 957
 usbsystemEnumerationDelay (*C macro*), 957
 usbsystemModeList (*C macro*), 957
 usbsystemNumberOfOptions (*C macro*), 959
 usbsystemOverride (*C macro*), 958
 usbsystemOverride_AutoVbusToggle_Bit (*C macro*), 958
 usbsystemOverride_VbusDetect_Bit (*C macro*), 958
 usbsystemPowerBehavior (*C macro*), 957
 usbsystemPowerBehavior_Balancing (*C macro*), 957
 usbsystemPowerBehavior_Even (*C macro*), 957
 usbsystemPowerBehaviorConfig (*C macro*), 958
 usbsystemPowerDataMode (*C macro*), 957
 usbsystemResetEntityToFactoryDefaults (*C macro*), 958
 usbsystemSelectorMode (*C macro*), 958
 usbsystemSelectorMode_Disabled (*C macro*), 958
 usbsystemSelectorMode_Mux (*C macro*), 958
 usbsystemSelectorMode_Upstream (*C macro*), 958
 usbsystemStateList (*C macro*), 957
 usbsystemUpstreamHSPort (*C macro*), 959
 usbsystemUpstreamPort (*C macro*), 957
 usbsystemUpstreamPortNone (*C macro*), 957
 usbsystemUpstreamSSPort (*C macro*), 959
 usbUpstreamBoostMode (*C macro*), 952
 usbUpstreamMode (*C macro*), 951
 usbUpstreamModeAuto (*C macro*), 951
 usbUpstreamModeDefault (*C macro*), 951
 usbUpstreamModeNone (*C macro*), 951
 usbUpstreamModePort0 (*C macro*), 951
 usbUpstreamModePort1 (*C macro*), 951
 usbUpstreamState (*C macro*), 952
 usbUpstreamStateNone (*C macro*), 952
 usbUpstreamStatePort0 (*C macro*), 952
 usbUpstreamStatePort1 (*C macro*), 952
 userconfigLoadEntityFromStore (*C macro*), 960
 userconfigNumberOfOptions (*C macro*), 960
 userconfigResetEntityToFactoryDefaults (*C macro*), 960
 userconfigSaveEntityToStore (*C macro*), 960

V

`val_HB_H2S_DOWN` (*C macro*), 906
`val_HB_H2S_UP` (*C macro*), 906
`val_HB_M2R_DOWN` (*C macro*), 906
`val_HB_M2R_UP` (*C macro*), 906
`val_HB_S2H_DOWN` (*C macro*), 906
`val_HB_S2H_UP` (*C macro*), 906
`VALIDPACKET` (*C++ member*), 890
`value` (*brainstem.link.StreamStatusEntry property*), 582
`value` (*brainstem.result.Result property*), 589
`values()` (*brainstem.result.Result method*), 589
`version_GetMajor` (*C++ function*), 1441
`version_GetMinor` (*C++ function*), 1441
`version_GetPatch` (*C++ function*), 1441
`version_IsAtLeast` (*C++ function*), 1441
`version_IsAtLeastCompare` (*C++ function*), 1441
`version_IsLegacyFormat` (*C++ function*), 1441
`version_Pack` (*C++ function*), 1441
`version_ParseMajor` (*C++ function*), 1440
`version_ParseMinor` (*C++ function*), 1440
`version_ParsePatch` (*C++ function*), 1440

W

`write()` (*brainstem.entity.I2C method*), 614