



BrainStem Reference Manual

Release 2.9.29

Acroname, Inc.

Apr 24, 2023

1	Overview	1
1.1	Organization	1
1.1.1	Products	1
1.1.2	API Reference	1
1.1.3	BrainStem	1
1.2	Support	1
2	Products	3
2.1	USBHub3p	4
2.1.1	Quick Start Guide	4
2.1.2	Basic Example	5
2.1.3	Indicators and Connections	7
2.1.4	Programming Interface	8
2.1.5	USBHub3+ Module Entities	9
2.2	USBHub3c	19
2.2.1	Quick Start Guide	19
2.2.2	Basic Example	21
2.2.3	Indicators and Connections	22
2.2.4	Programming Interface	25
2.2.5	USBHub3c Module Entities	30
2.2.6	USBHub3c Software Features	50
2.3	USBHub2x4	66
2.3.1	Quick Start Guide	66
2.3.2	Basic Example	67
2.3.3	Indicators and Connections	70
2.3.4	Programming Interface	71
2.3.5	USBHub2x4 Module Entities	72
2.4	USB-C-Switch	81
2.4.1	Passive and Redriver Models	81
2.4.2	Overview	82
2.4.3	Quick Start Guide	82
2.4.4	Basic Example	83
2.4.5	Indicators and Connections	85
2.4.6	Programming Interface	86
2.4.7	USB-C-Switch Module Entities	90
2.5	MTM Products	101
2.5.1	MTM-Relay	102

2.5.2	MTM-DAQ-2	114
2.5.3	MTM-PM-1	128
2.5.4	MTM-Load-1	143
2.5.5	MTM-IO-Serial	158
2.5.6	MTM-EtherStem	180
2.5.7	MTM-USBStem	197
3	API Reference	213
3.1	BrainStem Entities	214
3.1.1	Entities	214
3.1.2	Analog Entity	214
3.1.3	App Entity	216
3.1.4	Clock Entity	217
3.1.5	Digital Entity	219
3.1.6	Equalizer Entity	221
3.1.7	I2C Entity	223
3.1.8	Mux Entity	224
3.1.9	Pointer Entity	225
3.1.10	Port Entity	228
3.1.11	Power Delivery Entity	231
3.1.12	Rail Entity	237
3.1.13	RCServo Entity	241
3.1.14	Relay Entity	243
3.1.15	Signal Entity	244
3.1.16	Store Entity	246
3.1.17	System Entity	249
3.1.18	Temperature Entity	253
3.1.19	Timer Entity	254
3.1.20	UART Entity	255
3.1.21	USB Entity	257
3.1.22	USB System Entity	263
3.2	Python API Reference	266
3.2.1	Getting (Quickly) Started	266
3.2.2	Acroname Modules	269
3.2.3	Package Structure	287
3.2.4	Analog	288
3.2.5	App	292
3.2.6	Clock	293
3.2.7	Definitions	295
3.2.8	Digital	296
3.2.9	Discovery	298
3.2.10	Entities	300
3.2.11	Equalizer	307
3.2.12	I2C	308
3.2.13	Link	309
3.2.14	Module	311
3.2.15	Mux	317
3.2.16	Pointer	319
3.2.17	Port	322
3.2.18	Power Delivery	333
3.2.19	Rail	340
3.2.20	RCServo	347
3.2.21	Relay	348
3.2.22	Results	349

3.2.23	Signal	349
3.2.24	System	351
3.2.25	Store	359
3.2.26	Temperature	361
3.2.27	Timer	362
3.2.28	UART	363
3.2.29	USB	364
3.2.30	USB System	372
3.2.31	Version	375
3.3	C++ API Reference	376
3.3.1	Errors	376
3.3.2	Classes	376
3.3.3	Acroname Modules	379
3.3.4	Analog Class	411
3.3.5	App Class	415
3.3.6	Clock Class	416
3.3.7	Digital Class	418
3.3.8	Entity Class	419
3.3.9	Equalizer Class	424
3.3.10	I2C Class	425
3.3.11	Link Class	426
3.3.12	Module Class	438
3.3.13	Mux Class	442
3.3.14	Pointer Class	444
3.3.15	Port Class	447
3.3.16	Power Delivery Class	456
3.3.17	Rail Class	464
3.3.18	RCServo Class	469
3.3.19	Relay Class	471
3.3.20	Signal Class	472
3.3.21	Store Class	473
3.3.22	System Class	475
3.3.23	Temperature Class	483
3.3.24	Timer Class	484
3.3.25	UART Class	485
3.3.26	USB Class	486
3.3.27	USBSysntem Class	494
3.4	C API Reference	498
3.4.1	Modules	498
3.4.2	aDefs.h	500
3.4.3	aDiscovery.h	501
3.4.4	Error Codes	504
3.4.5	aFile.h	507
3.4.6	aLink.h	510
3.4.7	aMutex.h	514
3.4.8	aPacket.h	515
3.4.9	aProtocoldefs.h	517
3.4.10	aStream.h	544
3.4.11	aTime.h	555
3.4.12	aUEI.h	556
3.4.13	aVersion.h	558
3.4.14	PortMapping.h	560
3.5	.NET API Reference	562
3.5.1	Classes	562

3.5.2	Errors	564
3.5.3	BrainStem2 CLI Types	566
3.5.4	Analog Class	568
3.5.5	App Class	572
3.5.6	Clock Class	573
3.5.7	Digital Class	575
3.5.8	Equalizer Class	576
3.5.9	I2C Class	577
3.5.10	Module Class	579
3.5.11	Mux Class	582
3.5.12	Pointer Class	584
3.5.13	Port Class	587
3.5.14	Port Mapping	596
3.5.15	Power Delivery Class	599
3.5.16	Rail Class	606
3.5.17	RCServo Class	611
3.5.18	Relay Class	612
3.5.19	Signal Class	613
3.5.20	Store Class	615
3.5.21	System Class	616
3.5.22	Temperature Class	621
3.5.23	Timer Class	622
3.5.24	UART Class	623
3.5.25	USB Class	624
3.5.26	USBSystem Class	632
3.6	LabVIEW API Reference	637
3.6.1	Entities	637
3.6.2	Analog Entity	639
3.6.3	App Entity	644
3.6.4	Clock Entity	645
3.6.5	Digital Entity	648
3.6.6	Equalizer Entity	650
3.6.7	I2C Entity	651
3.6.8	Module Entity	653
3.6.9	Mux Entity	655
3.6.10	Pointer Entity	658
3.6.11	Port Entity	661
3.6.12	PowerDelivery Entity	678
3.6.13	Rail Entity	689
3.6.14	RCServo Entity	697
3.6.15	Relay Entity	698
3.6.16	Signal Entity	699
3.6.17	Store Entity	701
3.6.18	System Entity	704
3.6.19	Temperature Entity	715
3.6.20	Timer Entity	716
3.6.21	UART Entity	717
3.6.22	USB Entity	718
3.6.23	USBSystem Entity	730
3.7	Reflex Language Reference	736
3.7.1	Introduction	736
3.7.2	Working with Reflex files	736
3.7.3	A Basic “Hello World” Example.	738
3.7.4	Blink My LED Example	740

3.7.5	Built in reflex origins	743
3.7.6	Keywords in the Reflex Language	745
3.7.7	Operators and Precedence	745
3.7.8	Types, Identifiers and Numbers	747
3.7.9	The Reflex Preprocessor	749
3.7.10	Variable Declaration	749
3.7.11	Statements	751
3.7.12	Reflex and Routine Definition	753
3.7.13	Appendix	754
4	BrainStem	763
4.1	BrainStem Platform	763
4.1.1	What is BrainStem?	763
4.1.2	Explore your module's capabilities With HubTool.	763
4.1.3	Are you up-to-date?	763
4.1.4	Hello World Reflex	763
4.1.5	Hello World C++	764
4.1.6	Next Steps	764
4.2	What is BrainStem	764
4.2.1	Embedded With Reflex	764
4.2.2	Scalable	765
4.2.3	Usable	766
4.2.4	Next Steps	766
4.3	Getting Started	766
4.3.1	Do I need Drivers?	766
4.3.2	Connecting to a BrainStem Device	766
4.3.3	Launch HubTool	767
4.3.4	Toggling the LED	768
4.4	Firmware Management	768
4.4.1	Firmware Upgrade Precautions	768
4.4.2	Brainstem Firmware	768
4.4.3	Firmware Update Tools	769
4.4.4	Using Updater via CLI	769
4.4.5	Example: Updating to the Latest Firmware	772
4.4.6	Example: Reverting to a Previous Version	773
4.4.7	Example: Recovering a BrainStem Module	776
4.4.8	Example: Updating a Brainstem Module via the Brainstem Network.	780
4.5	Terminology	784
4.5.1	BrainStem® Network	784
4.5.2	BrainStem® Bus	784
4.5.3	Routing	785
4.5.4	Module	785
4.5.5	Host	785
4.5.6	Reflex	785
4.5.7	Entity	785
4.5.8	Discovery	786
4.6	USB Drivers	786
4.6.1	Mac OS X	786
4.6.2	Linux Ubuntu	786
4.6.3	Windows 7 USB Driverless Installation	786
4.7	Appendix	787
4.7.1	Appendix I: BrainStem Universal Entity Interface (UEI)	788
4.7.2	Appendix II: BrainStem Communication Protocol	796
4.7.3	Appendix III: BrainStem Networking	798

4.7.4	Appendix IV: Updater File Structure	805
Index		809

You've found the Acroname Software Reference. This reference covers the various APIs that make up the Acroname software Ecosystem.

1.1 Organization

1.1.1 Products

If you recently purchased one of our products and are looking to get started quickly, the [Products](#) section will introduce you to your acroname device and get you up and running quickly.

1.1.2 API Reference

The [API Reference](#) is organized by language and provides full API documentation when working with the BrainStem APIs and trying to answer specific questions.

1.1.3 BrainStem

Acroname products are built around [BrainStem](#) technology. Brainstem is a protocol, organizing principle, and operating system underlying all of our devices. If you want to learn more, check out this section.

1.2 Support

We love feedback and questions! It helps us become better and results in better products and documentation or you. If you have a question or concern or would like to leave us a love note, we'd love to hear from you.

- **Contact Page:** <https://acroname.com/contact-us>

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

2.1 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.

2.1.1 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 you particular operating system and architecture.

Play

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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](#)

2.1.2 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/software/brainstem-development-kit>

(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)

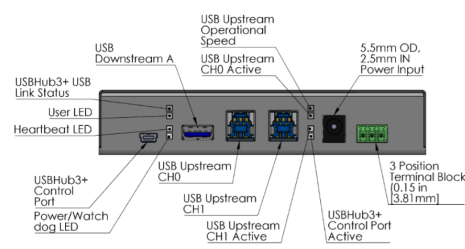
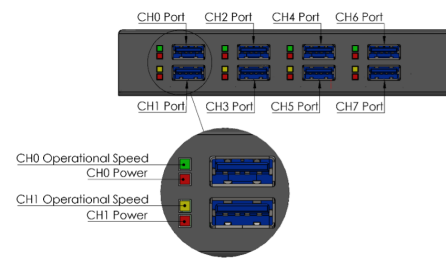
#####
#Do Stuff
#####

hub.usb.setPortDisable(PORT)

#Close the connection
hub.disconnect()
```

2.1.3 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

2.1.4 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.

2.1.5 USBHub3+ Module Entities

Temperature

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

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].getTemperature(μC) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

where temperature is in micro-degrees Celcius.

System

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

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.usb.setSuperSpeedDataDisable(channel) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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,  $\mu$ V) [cpp] [python] [NET] [LabVIEW]
```

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

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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] [LabVIEW]
```



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

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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

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

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

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] [LabVIEW]
```

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

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] [LabVIEW]
```

This command returns a 32-bit value which indicates:

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

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] [LabVIEW]
```

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

Bit	Hub Operational Mode Word Definition
0	USB Ch 0 USB Hi-Speed Data Enabled
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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
system[0]	slotSize	
	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getInputCurrent	
	getVersion	
	setHBInterval	
	getHBInterval	
	getModule	
	getSerialNumber	
	getModel	
temperature[0]	getTemperature	
timer[0-8]	getExpiration	
	setExpiration	
	getMode	
usb[0]	setMode	
	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
	setPortMode	Channels 0-7
	getPortMode	Channels 0-7
	getDownstreamDataSpeed	Channels 0-7
	getHubMode	
	setHubMode	
	getPortState	Channels 0-7
	getPortError	

continues on next page

Table 2 – continued from previous page

Entity Class	Entity Option	Variable(s)	Notes
	getEnumerationDelay		
	setEnumerationDelay		
	clearPortErrorStatus		
	getUpstreamMode		
	setUpstreamMode		
	getUpstreamState		
	getUpstreamBoostMode		
	setUpstreamBoostMode		
	getDownstreamBoostMode		
	setDownstreamBoostMode		
	Pointer[0-3]		
	getOffset		
	setOffset		
	getMode		
	setMode		
	getTransferStore		
	setTransferStore		
	initiateTransferToStore		
	initiateTransferFromStore		
	getChar		
	setChar		
	getShort		
	setShort		
	getInt		
	setInt		
App[0-3]	execute		

2.2 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.

2.2.1 Quick Start Guide

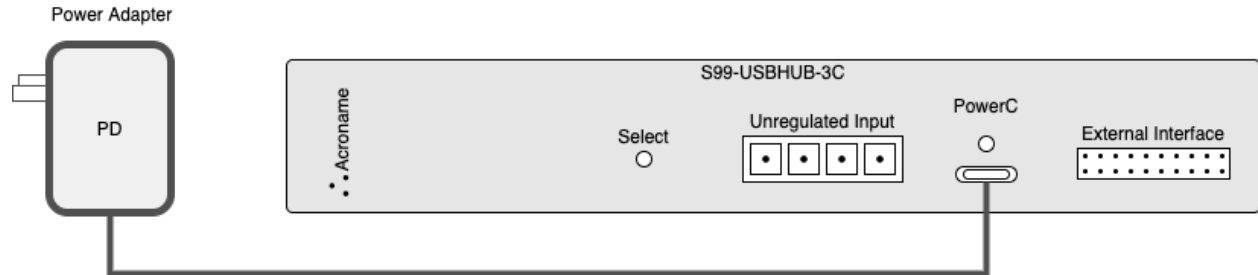
1. Download The Development Kit

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

² <https://acroname.com/software/brainstem-development-kit>

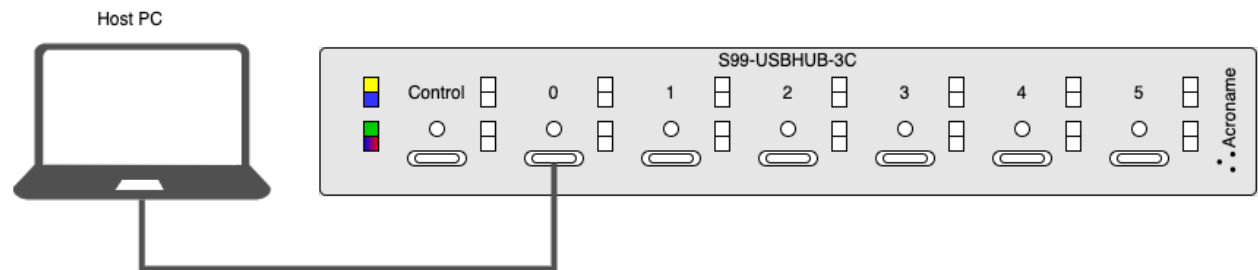
2. Connect Power

- 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

- Unzip the downloaded package
- Navigate to the “/bin” folder
- 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](#)

2.2.2 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 \(BDK\)](#)³ 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 USBHub3p
device = brainstem.stem.USBHub3c()

#Connect to USBHub3p
result = device.discoverAndConnect(brainstem.link.Spec.USB)
```

(continues on next page)

³ <https://acroname.com/software/brainstem-development-kit>

(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
#####

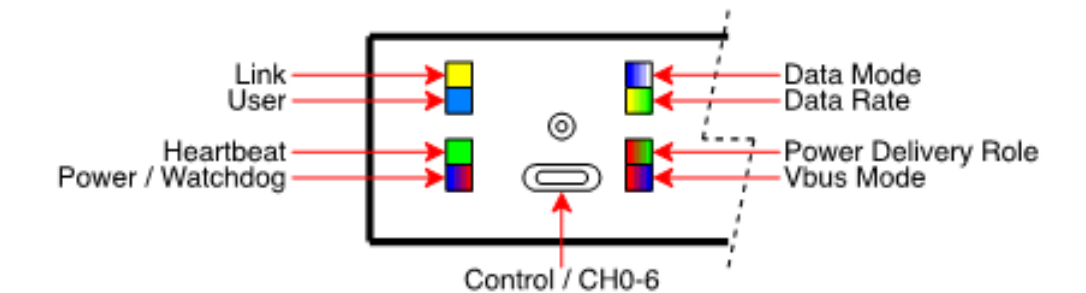
# Enable Port
device.hub.port[PORT].setEnabled(1)

#Close the connection
device.disconnect()

```

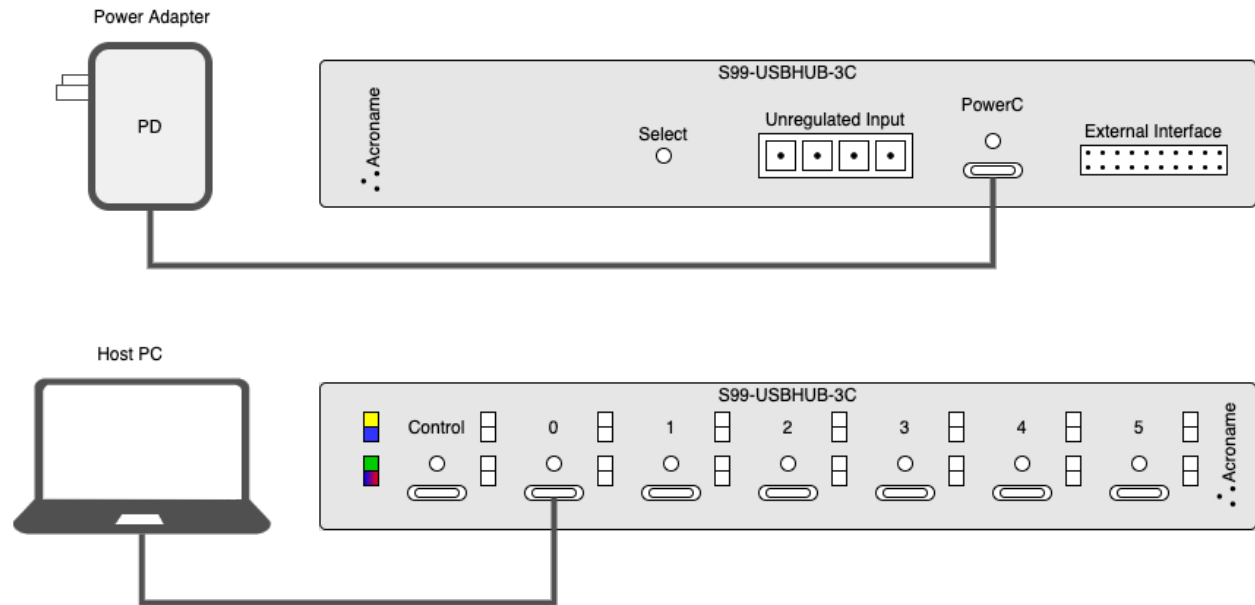
2.2.3 Indicators and Connections

LEDs

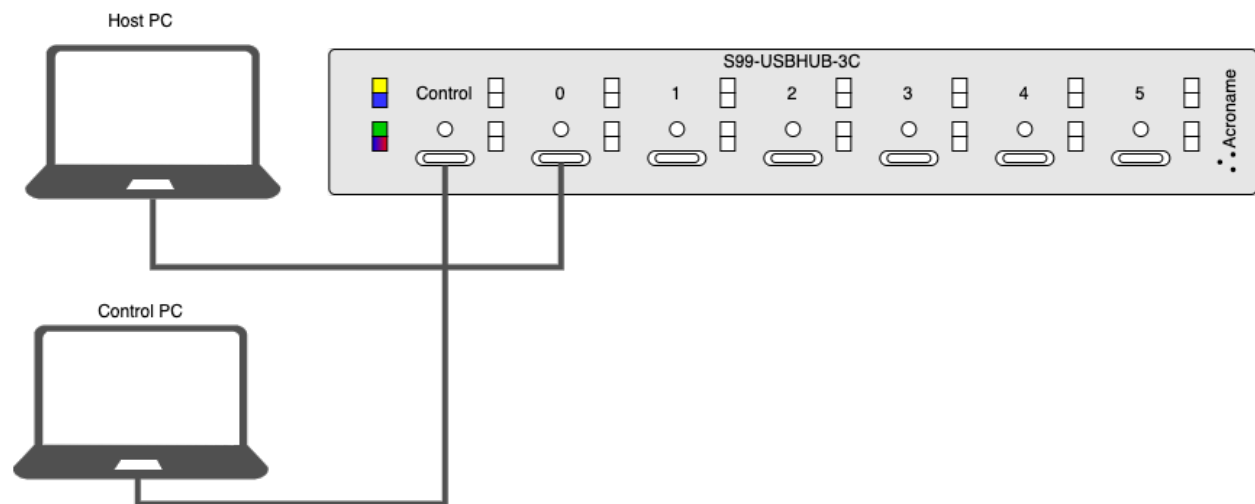


LED Name	Color	Description
Link Status LED	Yellow	On once a host device has enumerated the BrainStem controller
User LED	Blue	Can be manipulated through any of the available APIs
Heartbeat LED	Green	Indicates active BrainStem connection; pulses at a rate determined by the system heartbeat rate
Power/Watchdog 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

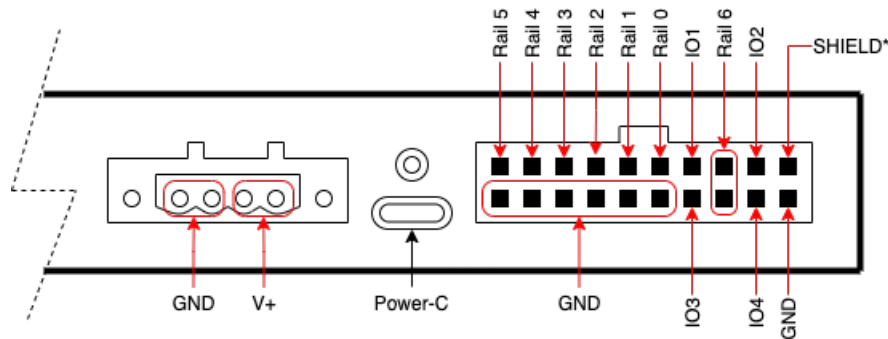
Connections



Separate Control Computer



External Connector



2.2.4 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:

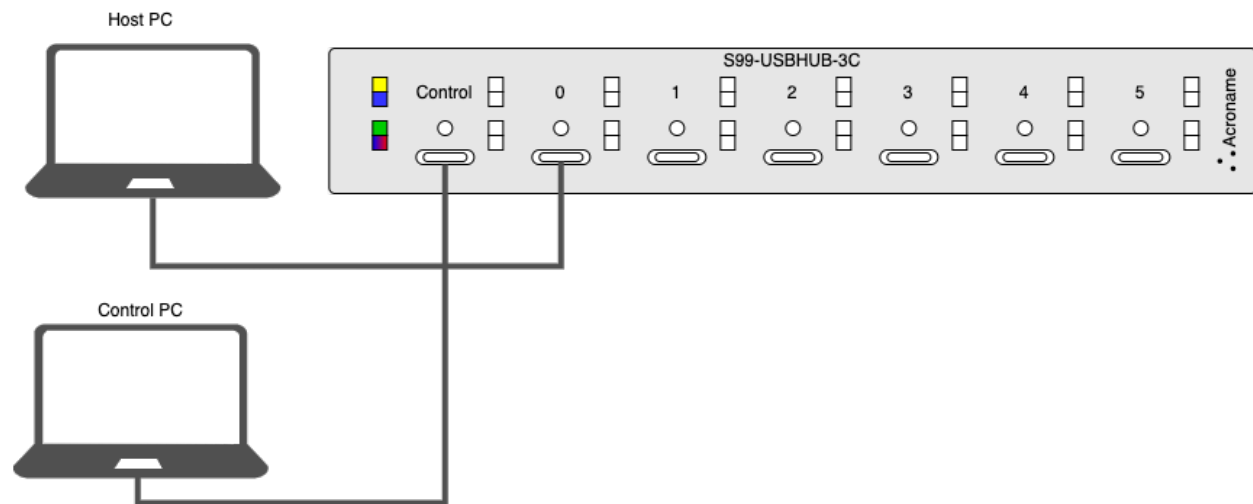
```
device.link.discoverAndConnect(USB)
```

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.

2.2.5 USBHub3c Module Entities

I2C

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

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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

Port

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

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] [LabVIEW]
stem.hub.port[x].getEnabled() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.hub.port[x].getPowerEnabled() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.hub.port[x].getDataEnabled() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.hub.port[x].getDataHSEnabled() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.hub.port[x].getDataHS1Enabled() [cpp] [python] [NET] [LabVIEW]
```

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

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] [LabVIEW]
stem.hub.port[x].getDataSSEnabled() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.hub.port[x].getDataSS1Enabled() [cpp] [python] [NET] [LabVIEW]
```

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

The CC lines can also be individually controlled.

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

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

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

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

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] [LabVIEW]
stem.hub.port[x].getVconnEnabled() [cpp] [python] [NET] [LabVIEW]
```

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] :[LabVIEW]
stem.hub.port[x].getVconn1Enabled() [cpp] [python] [NET] [LabVIEW]
```

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

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] [LabVIEW]
stem.hub.port[x].getVbusCurrent() [cpp] [python] [NET] [LabVIEW]
```

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

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] [LabVIEW]
stem.hub.port[x].getPowerMode() [cpp] [python] [NET] [LabVIEW]
```

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

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] [LabVIEW]
stem.hub.port[x].getMode() [cpp] [python] [NET] [LabVIEW]
```

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	4	portPortMode_portPowerMode_ps_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] [LabVIEW]
```

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

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

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

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

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

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] [LabVIEW]
```

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] [LabVIEW]  
stem.hub.port[x].getVconnAccumulatedPower() [cpp] [python] [NET] [LabVIEW]
```

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]  
stem.hub.port[x].resetVconnAccumulatedPower() [cpp] [python] [NET] [LabVIEW]
```

Port Errors

Power Delivery

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

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.

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

```
stem.hub.pd[x].setRequestDataObject() [cpp] [python] [NET] [LabVIEW]
stem.hub.pd[x].getRequestDataObject() [cpp] [python] [NET] [LabVIEW]
```

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.hub.pd[x].request() [cpp] [python] [NET] [LabVIEW]
```

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

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.hub.pd[x].requestStatus() [cpp] [python] [NET] [LabVIEW]
```

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.hub.pd[x].getCableOrientation() [cpp] [python] [NET] [LabVIEW]
```

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.hub.pd[x].getOverride() [cpp] [python] [NET] [LabVIEW]
stem.hub.pd[x].setOverride() [cpp] [python] [NET] [LabVIEW]
```

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

Rail

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

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] [LabVIEW]  
stem.rail[x].getEnable() [cpp] [python] [NET] [LabVIEW]
```

System

API Documentation: [cpp] [python] [NET] [LabVIEW]

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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]
```

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

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] [LabVIEW]
```

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] [LabVIEW]  
stem.system.getInputPowerBehavior() [cpp] [python] [NET] [LabVIEW]
```

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]  
stem.system.getInputPowerBehaviorConfig() [cpp] [python] [NET] [LabVIEW]
```

Voltage and Current Monitoring

Temperature

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

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].getTemperature(μC) [cpp] [python] [NET] [LabVIEW]
```

where temperature is in micro-degrees Celcius.

Uart

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

The UART entity is a class which allows the configuration of a specified uart port.

Uart Control

The USBHub3c has the ability to be controlled through an RS232 interface on the external expansion connector of the USBHub3c. For additional information about the serial protocol, reference *USBHub3c Serial Communication Feature*

Uart Protocols

The USBHub3c has two protocol values enumerated.

Value	Description
0	Disabled/Undefined
1	Extron Compatible Protocol

Uart APIs

Uarts are controlled through the following APIs:

```
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]
```

USB System

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

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

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

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() \[cpp\] \[python\] \[NET\] \[LabVIEW\]  
stem.hub.getPowerBehavior() \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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() \[cpp\] \[python\] \[NET\] \[LabVIEW\]  
stem.hub.getPowerBehaviorConfig() \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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	usbssystemDataBehavior_HardCoded
Reserved	1	usbssystemDataBehavior_Reserved
Port Priority	2	usbssystemDataBehavior_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] [LabVIEW]
stem.hub.getDataRoleBehavior() [cpp] [python] [NET] [LabVIEW]
```

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]
stem.hub.getDataRoleBehaviorConfig() [cpp] [python] [NET] [LabVIEW]
```


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

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.setModelList() [cpp] [python] [NET] [LabVIEW]
stem.hub.getModelList() [cpp] [python] [NET] [LabVIEW]
```

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

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

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

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

Power

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

Voltage

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

Current

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

Data

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

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	
	getModuleBaseAddress	
	setRouter	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	getVersion	
	getModel	
	getHardwareVersion	
	getSerialNumber	
	save	
	reset	
	logEvents	
	getUptime	
	getTemperature	
	getMinimumTemperature	
	getMaximumTemperature	
	getInputPowerSource	
	getInputVoltage	
	getInputCurrent	
	getModuleHardwareOffset	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	routeToMe	
	getUnregulatedVoltage	
	getUnregulatedCurrent	
	getPowerLimit	
	getPowerLimitMax	
	setPowerLimitMax	
	resetDeviceToFactoryDefaults	
temperature[0-2]	getValue	
	getValueMax	
	getValueMin	
port[0-7]	getVbusVoltage	
	getVbusCurrent	
	getVconnVoltage	
	getVconnCurrent	

continues on next page

Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getPowerEnabled	
	setPowerEnabled	
	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	
	getCurrentLimit	
	setCurrentLimit	
	getAllocatedPower	
	getAvailablePower	
	getPowerLimit	
	setPowerLimit	
	getVbusAccumulatedPower	
	resetVbusAccumulatedPower	
	getVconnAccumulatedPower	
	resetVconnAccumulatedPower	
	getHSBoost	

continues on next page

Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
USBSystem [0]	setHSBoost	
	getDataHSRoutingBehavior	
	setDataHSRoutingBehavior	
	getDataSSRoutingBehavior	
	setDataSSRoutingBehavior	
	getUpstream	
PowerDelivery [0-8]	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
	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	
	resetEntityToFactoryDefaults	
UART[0]	setEnabled	
	getEnable	
	setBaudRate	
	getBaudRate	
	setProtocol	
rail[0-6]	getProtocol	
	setEnabled	
i2c[0]	getEnable	
	read	
	write	
	setPullup	

continues on next page

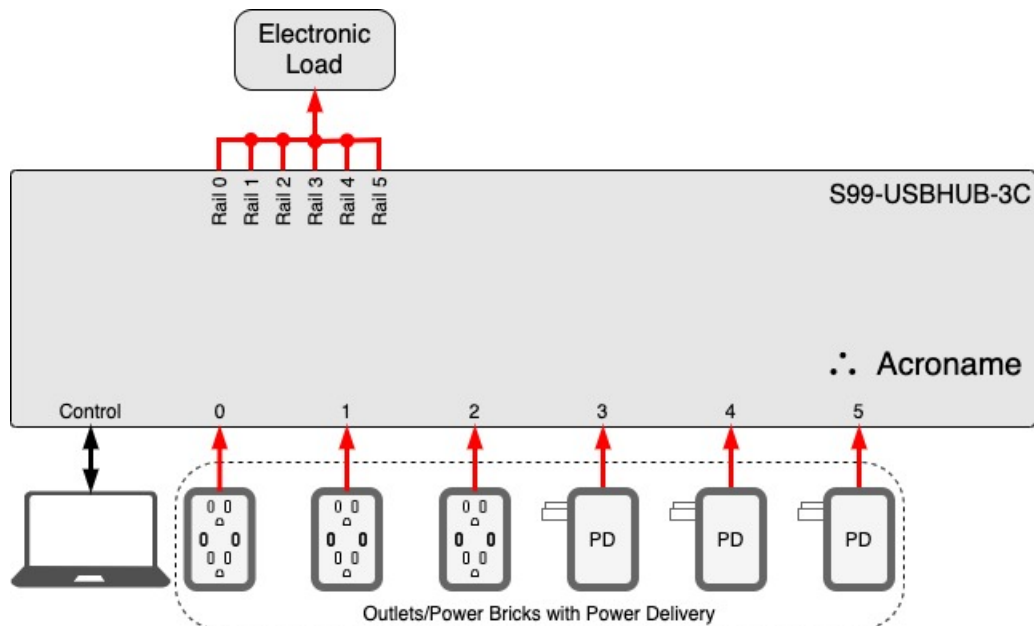
Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
usb[0]	setSpeed	
	getSpeed	
	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
	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

2.2.6 USBHub3c Software Features

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] [LabVIEW]
```

```
stem.rail[x].getEnabled() [cpp] [python] [NET] [LabVIEW]
```

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
<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	<div>3400</div>	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;

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] [LabVIEW]
stem.hub.pd[x].getPowerDataObject() [cpp] [python] [NET] [LabVIEW]
```

PD Logging

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

Use Cases

- Validating Power Delivery behavior
- Debugging PD communication.
- Decoding Vendor Defined Messages

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 a device with SN: 0x4F7A0C15, Model: 24, Firmware: 2.9.6, and Module: 6. It also displays Voltage: 20.0 VDC, Current: 0.5 A, and Temperature: 30 C. Hardware and SW offsets are both 0. Below this is a row of tabs: 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, showing a table of log entries. The table has columns for Timestamp (SuS), Port, Direction, Spec, SOP, Power Role, Data Role, ID, Packet Type, Msg Type, and Raw. The log contains 25 entries, showing various PD messages like Source Capabilities, Request, Accept, PS Ready, Vendor Defined, and Control messages. At the bottom, there are sections for 'Software Feature' (listing features like Rail Enable, QC Mode, PDO Editing, VBUS Validation, and PD Logging as enabled) and 'Device Type' (listing USBHub3c and USBHub3p).

	Timestamp (SuS)	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

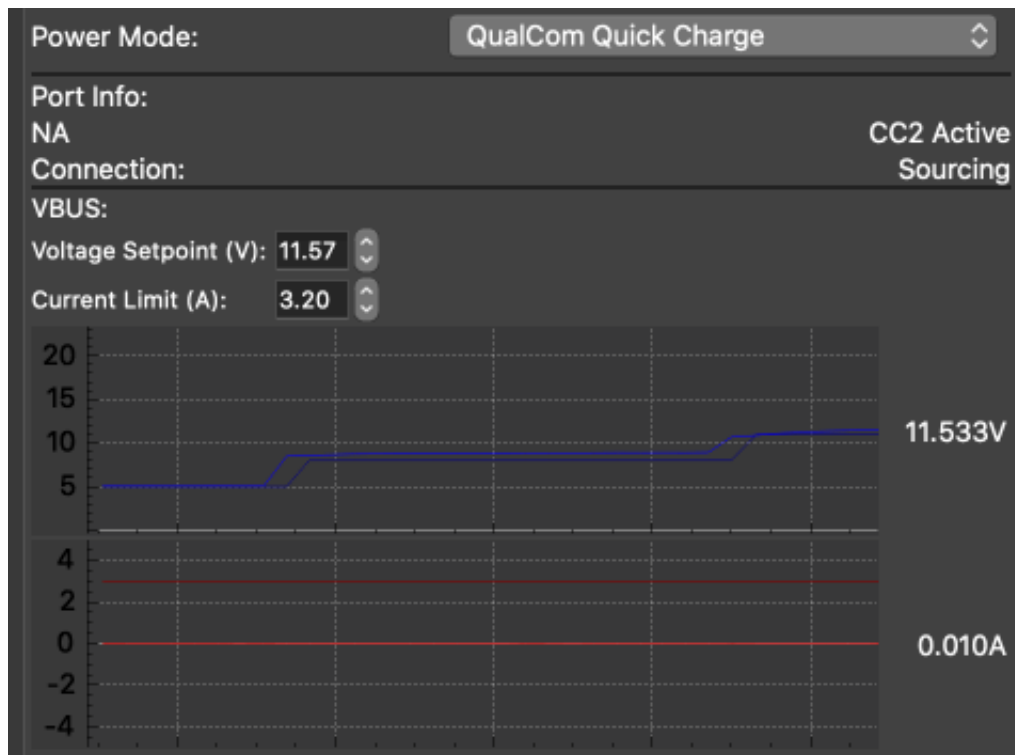
Device Type: | Serial Number:
USBHub3c 4F7A0C15
USBHub3p AF62130D

Quick Charge™

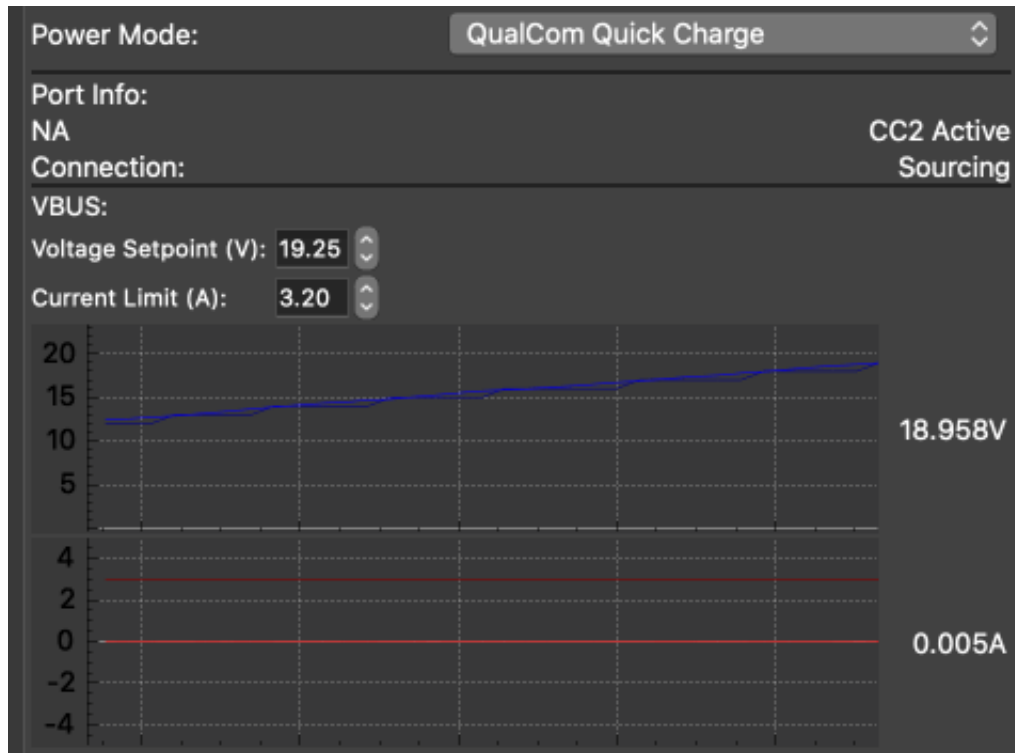
The USBHub3c Quick Charge™ feature adds the ability to enable the Quick Charge™ port power mode on one or more ports. This enables fast charging on devices that support Quick Charge™.

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] [LabVIEW] stem.hub.port[x].  
getPowerMode() [cpp] [python] [NET] [LabVIEW]
```

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.

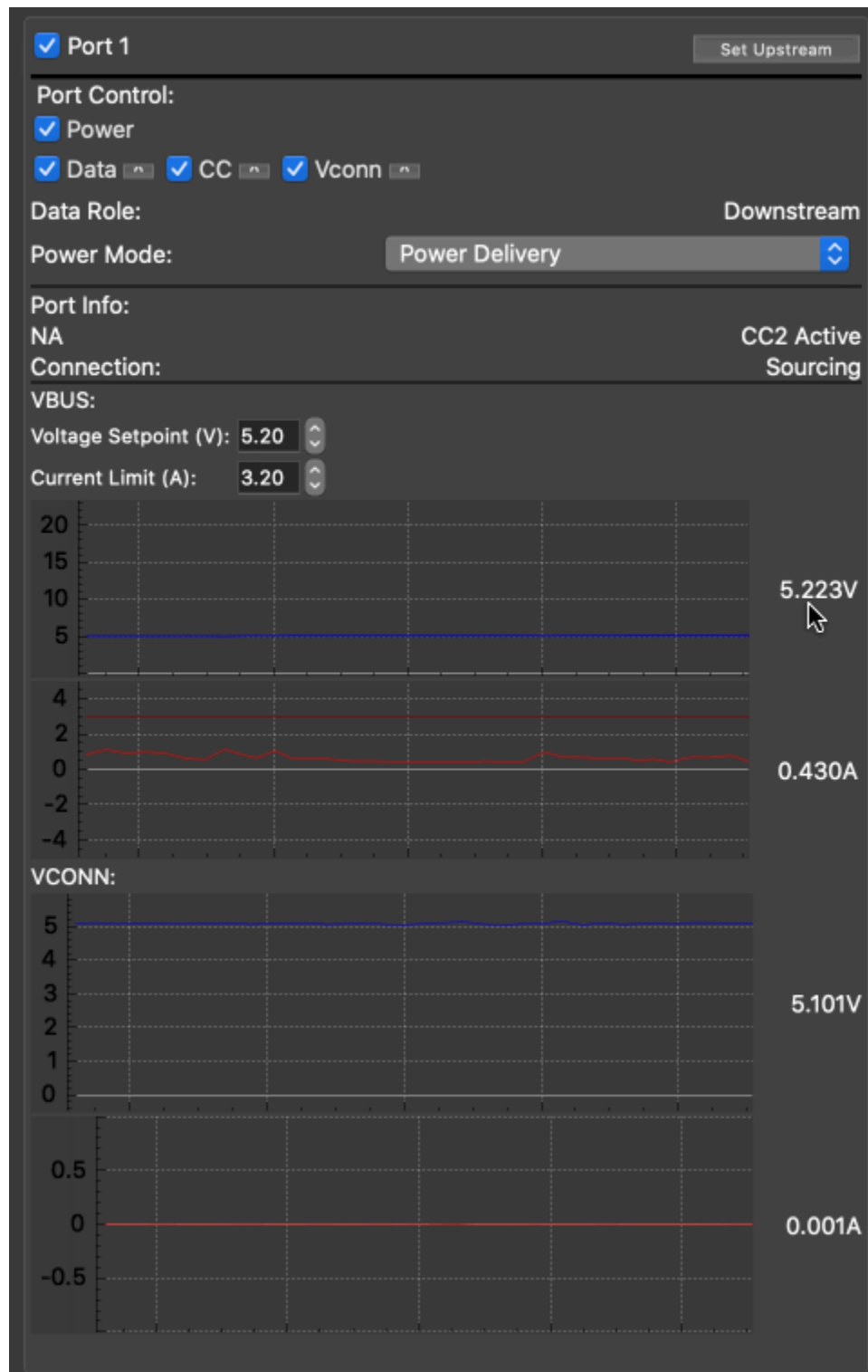
This software feature can be exploited through the *USBHub3c Port Entity*

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)):

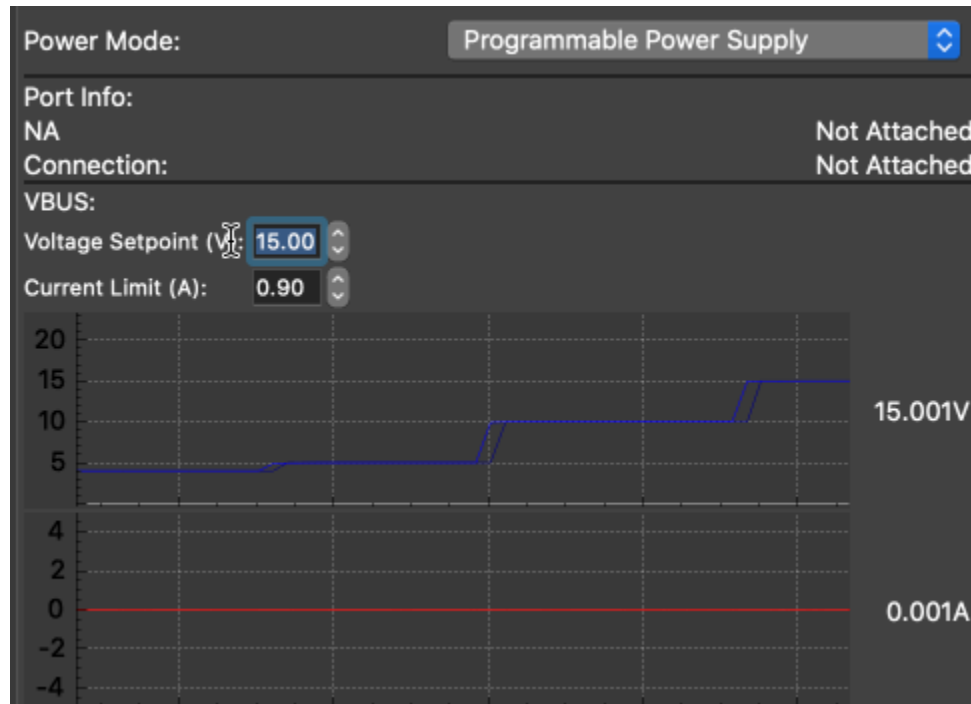
    #Do Stuff
    #Set desired port voltage and limit.
    stem.hub.port[TEST_PORT].setVoltageSetpoint(5500000); #5.5VDC
    stem.hub.port[TEST_PORT].setCurrentLimit(500000); #500mA
    #Do Stuff

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] [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 affect control and functionality of the USBHub3c.

Use Cases

- Affecting USBHub3c 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 *USBHub3c UART Entity*

Extron Compatible Serial Commands

The RS232 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
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
stem.uart[TEST_SERIAL].setProtocol(1)

# Perform a system save so the changes persist
# through power cycles
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]
```

2.3 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.

2.3.1 Quick Start Guide

1. Download The Development Kit

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

⁴ <https://acroname.com/software/brainstem-development-kit>

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

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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](#)

2.3.2 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

```

(continues on next page)

(continued from previous page)

```

import time
import sys

# 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)

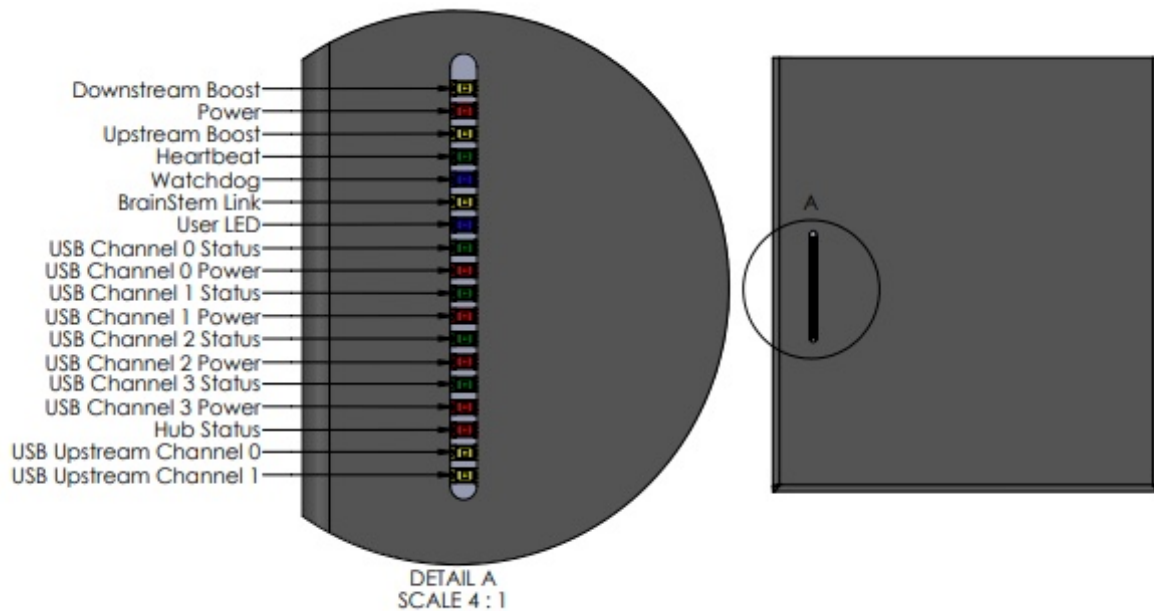
```
#####
##Do Stuff on Port 3
#####

##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")
```

2.3.3 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

2.3.4 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.

2.3.5 USBHub2x4 Module Entities

System

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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].getTemperature(μC) [cpp] [python] [NET] [LabVIEW]
```

where temperature is in micro-degrees Celcius.

USB

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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

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] [LabVIEW]
```

Value	Hub Upstream State Descriptions
0	Upstream port 0 is actively selected
1	Upstream port 1 is actively selected
2	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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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

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

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

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

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Non Standard Value
system[0]	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getInputCurrent	
	getVersion	
	setHBInterval	
	getHBInterval	
	getModule	
	getSerialNumber	
	getModel	
temperature[0]	getTemperature	
usb[0]	setPortEnable	Channels 0-3
	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	

continues on next page

Table 4 – continued from previous page

Entity Class	Entity Option	Non Standard Value
	setEnumerationDelay	
	clearPortErrorStatus	
	getUpstreamMode	
	setUpstreamMode	
	getUpstreamState	
	getUpstreamBoostMode	
	setUpstreamBoostMode	
	getDownstreamBoostMode	
	setDownstreamBoostMode	

2.4 USB-C-Switch



2.4.1 Passive and Redriver Models

There are two available models of the USB-C-Switch: Passive and Redriver. The two models have different hardware installed by Acroname during manufacturing; the model cannot be changed after delivery. Acroname places a label on the side of each USB-C-Switch indicating both the model and hardware revision. High-level summary and intended applications of the two models are below with detailed differences in the specification tables.

To get up to speed with the USB-C-Switch 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 USB-C-Switch for a more in depth view into the capabilities of each model.

2.4.2 Overview

The USB-C-Switch is a platform to simplify switching of multiple USB Type-C ports. The switch is a bidirectional four-to-one or one-to-four multiplexer (mux) which can create a dedicated connection between a device on the common port and a device on one of the available mux channels. The USB-C-Switch gives engineers advanced control of USB connections in testing and development applications. The USB-C-Switch consists of several layers of internal switches to achieve the 4:1 selector and USB line control functionality.

Without any hub or other directional intermediary devices, the USB-C-Switch can behave “like a cable” to connected devices. USB2, USB3, power, CC, Vconn, and SBU, are passed through the USB-C-Switch between the common-port and the selected mux port. Data link, power negotiations and power between USB devices are provided by the attached devices themselves, allowing the USB-C-Switch to be used bidirectionally in either a 1:4 or 4:1 configuration.

2.4.3 Quick Start Guide

1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)⁵ for you particular operating system and architecture.

2. Connect Device(s)

- Using the Acroname Universal Orientation Cable (UOC), or any standard USB-C cables, connect the Control and Common Ports of the USB-C-Switch to a device with access to the host device.
- Connect any other devices to any of Ports 0-3.

3. Run System

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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](#)

⁵ <https://acroname.com/software/brainstem-development-kit>

2.4.4 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);
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
        return 1;

    } else { printf("Connected to BrainStem module.\n"); }

    //Prep USBCSwitch 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
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 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)
```

(continues on next page)

(continued from previous page)

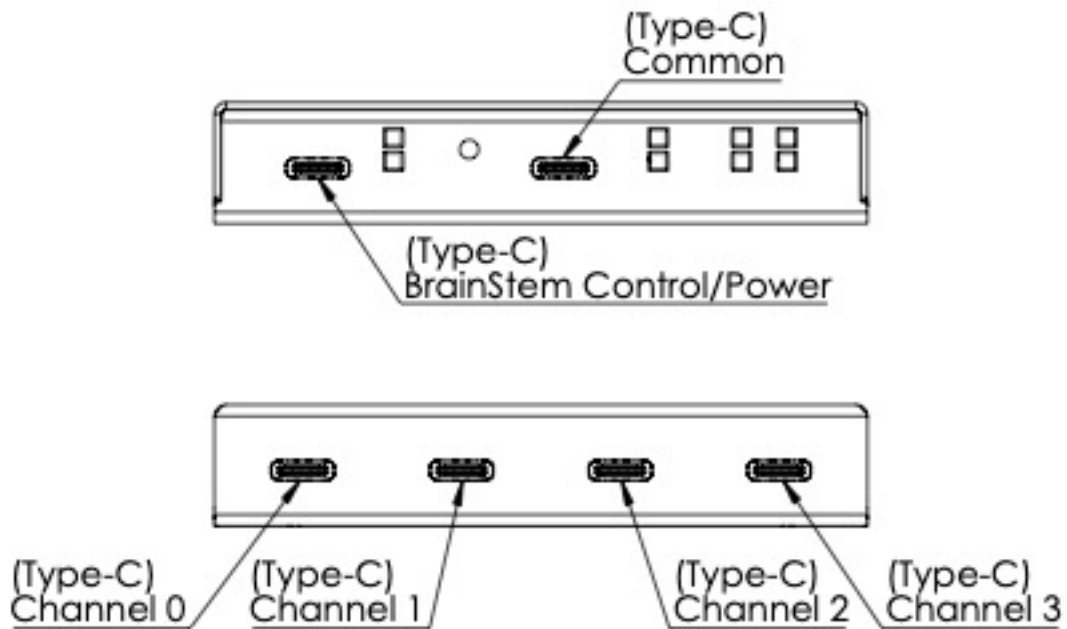
```
#####
##Do Stuff on Mux Channel 3
#####

#Finished with testing.
#De-initialize.
cswitch.usb.setPortDisable(0)
cswitch.mux.setEnable(False)

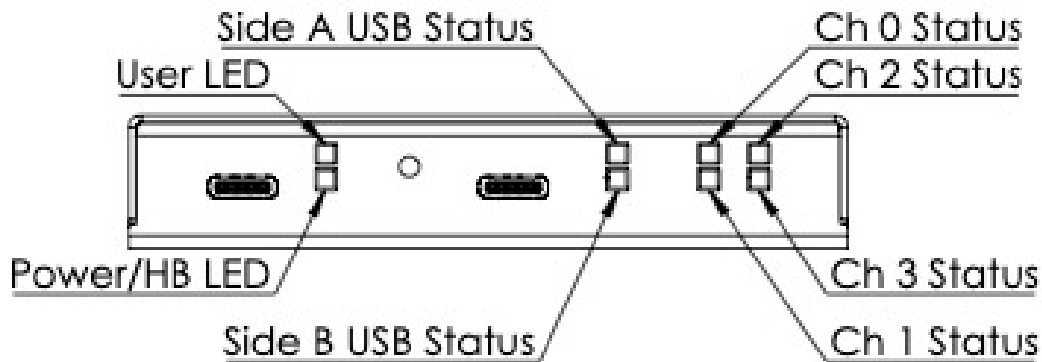
# Disconnect from device.
cswitch.disconnect()
```

2.4.5 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 heart-beat 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.

2.4.6 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

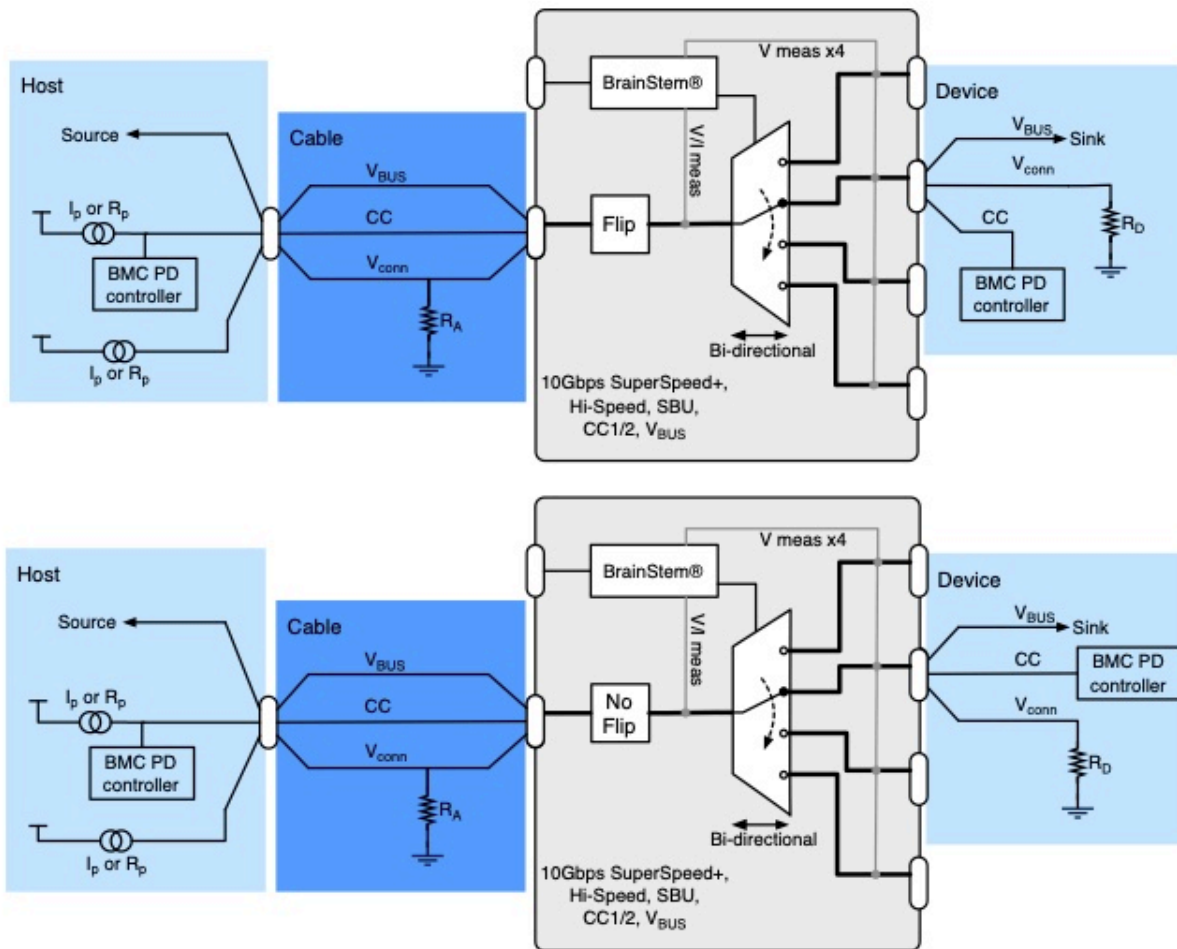
```
brainstem.discover.findAllModules(USB)  (Python)  
Acroname::BrainStem::Link::sDiscover()  (C++)
```

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



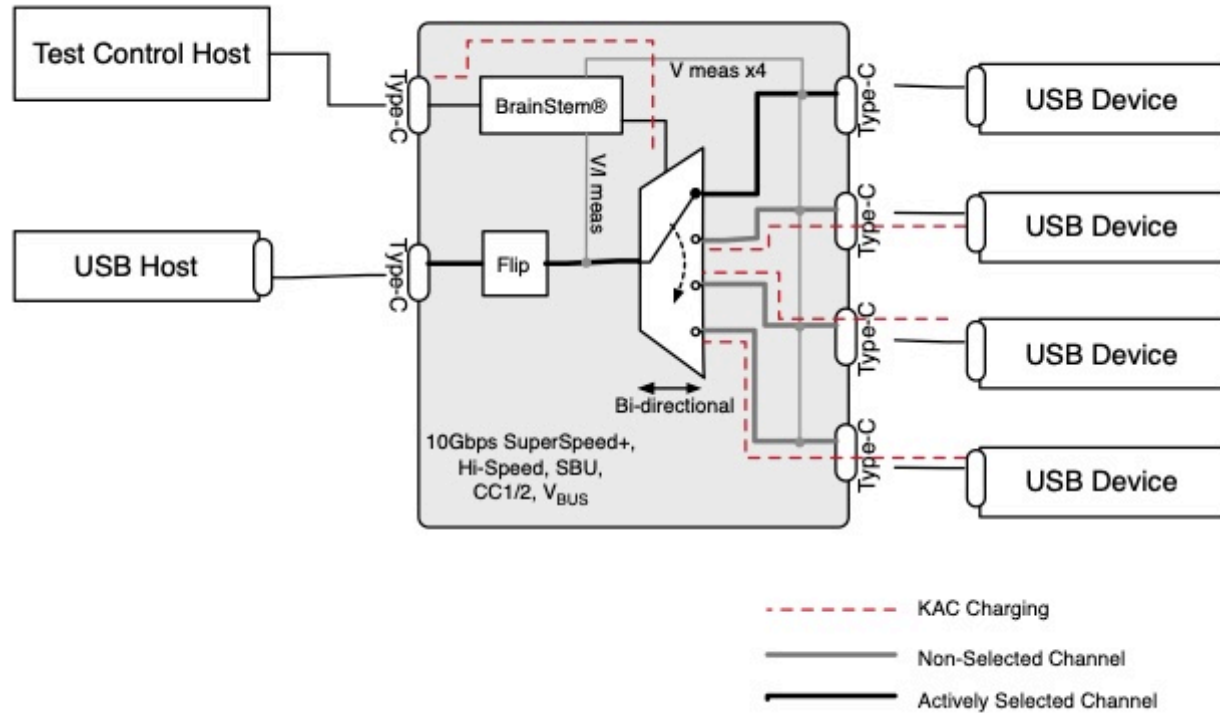
A UOC should be used on either the common port or mux port to enable automated cable flips. The UOC should be connected to the device under test.

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:



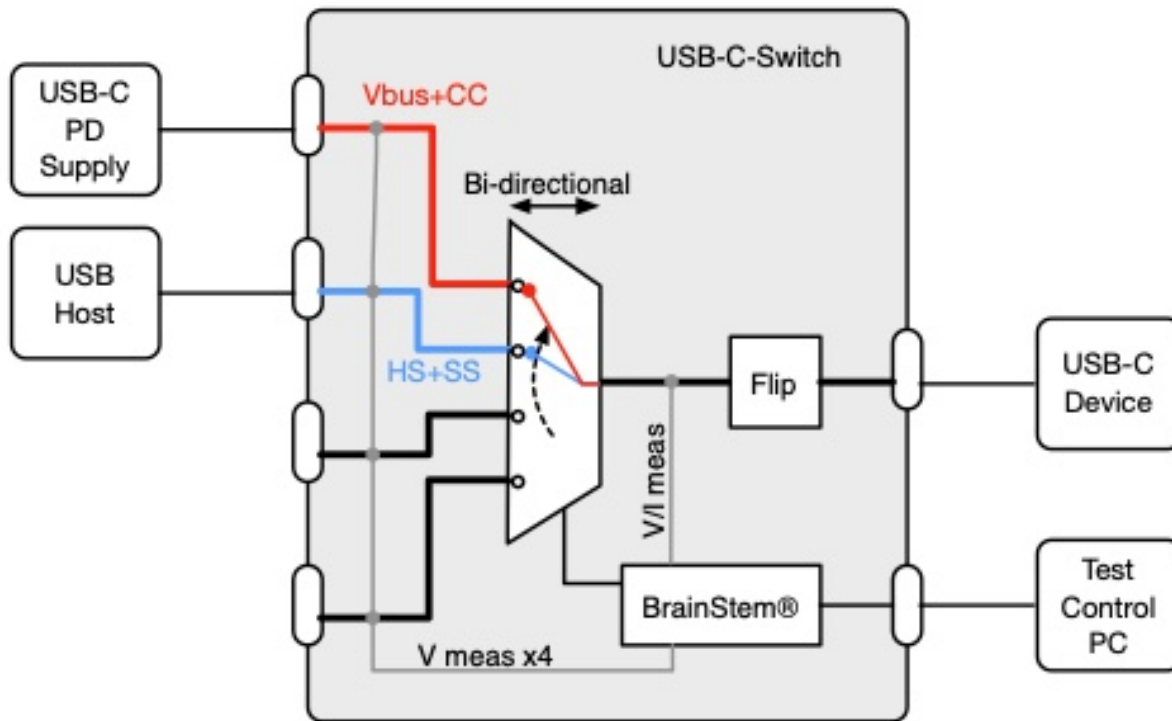
When KAC is enabled, the KAC circuit connects power from the control port VBUS to all non-selected mux channel VBUS lines. KAC power is applied only to inactive mux channels and is not applied to the actively selected mux channel since the actively selected channel has a power path to the common port. KAC is automatically disabled when mux split mode is enabled.

Mux Split 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. 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.

2.4.7 USB-C-Switch Module Entities

Equalizer

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

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 on 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] [LabVIEW]
```

```
stem.equalizer[x].getTransmitterConfig(config) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.equalizer[x].getReceiverConfig(chan, config) [cpp] [python] [NET] [LabVIEW]
```

where (chan) parameters 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] [LabVIEW]
```

```
stem.equalizer[0].getTransmitterConfig(config) [cpp] [python] [NET] [LabVIEW]
```

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 equalizer reciever 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 reciever 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].setRecieverConfig(0,config) [cpp] [python] [NET] [LabVIEW]
```

```
stem.equalizer[0].getRecieverConfig(0,config) [cpp] [python] [NET] [LabVIEW]
```

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 ful-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.

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 reciever option controls the reciever gain. The actual reciever gain is dependent on the alt-mode configuration and the port data direction (mux to common vs common to mux). There are independent reciever gain settings for the common and mux ports of the switch. Gains across settings, direction, and frequency is shown here:

Value	SS Recieve Gain Lever
0-15	Increasing levels of gain

Mux

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

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 (multiplexor) 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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.mux.getChannel(channel) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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". Default or split modes can be enabled with:

```
stem.mux.getConfiguration(config) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.mux.setConfiguration(config) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

where (config) is 0 for default and 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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.mux.setSplitMode(splitMode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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 uses 4-bits to define which mux ports are connected to the common port Vbus lines.

Bit	Mux Split Mode Bit Map
0:1	SBU1
2:3	SBU2
4:5	CC1
6:7	Reserved
8:9	CC2
10:11	Reserved
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

System

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

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) \[cpp\] \[python\] \[.NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

Saved Configurations	
USB Mode (usb)	Equalizer Configuration (equalizer)
Mux Split Mode (mux)	Mux Enable (mux)
Mux Configuration (mux)	Mux Port (mux)

USB

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

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] [LabVIEW]
```

```
stem.usb.setAltModeConfig(0, configuration) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.usb.setCableFlip(setting) [cpp] [python] [NET] [LabVIEW]
```

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 functionalite is interfaced with:

```
stem.usb.setConnectMode(0, mode) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.usb.setCC2Enable(0, enabled) [cpp] [python] [NET] [LabVIEW]
stem.usb.getCC1Enable(0, enabled) [cpp] [python] [NET] [LabVIEW]
stem.usb.getCC2Enable(0, enable) [cpp] [python] [NET] [LabVIEW]
```

where (enable) is a Boolean vale of 0 or 1.

CC line current and voltage can be measured with:

```
stem.usb.getCC1Voltage(0,  $\mu$ V) [cpp] [python] [NET] [LabVIEW]
stem.usb.setCC2Voltage(0,  $\mu$ V) [cpp] [python] [NET] [LabVIEW]
stem.usb.getCC1Current(0,  $\mu$ A) [cpp] [python] [NET] [LabVIEW]
stem.usb.getCC2Current(0,  $\mu$ A) [cpp] [python] [NET] [LabVIEW]
```

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

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] [LabVIEW]
stem.usb.setDataDisable(0) [cpp] [python] [NET] [LabVIEW]
```

Manipulating HS or SS data lines is done by calling:

```
stem.usb.setHiSpeedDataEnable(0 [cpp] [python] [NET] [LabVIEW]
stem.usb.setHiSpeedDataDisable(0) [cpp] [python] [NET] [LabVIEW]
stem.usb.setSuperSpeedDataEnable(0) [cpp] [python] [NET] [LabVIEW]
stem.usb.setSuperSpeedDataDisable(0) [cpp] [python] [NET] [LabVIEW]
```

Manipulating Vbus lines are done by calling:

```
stem.usb.setPowerEnable(0) [cpp] [python] [NET] [LabVIEW]
stem.usb.setPowerDisable(0) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.usb.getPortCurrent(0,  $\mu$ A) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.usb.setPortMode(0, mode) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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
8:9	Reserved
10:11	Reserved
12:13	Reserved
14	CC Flip enable
15	Super-Speed Flip enable
16	SBU Flip enable
17	Reserved
19:18	Daughter-Card status
22:20	Reserved
23	Connection Established
24:25	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] [LabVIEW]
```

```
stem.usb.getSBUEnable(0, enabled) [cpp] [python] [NET] [LabVIEW]
```

where (enable) is a Boolean vale of 0 or 1.

Complete List of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
system[0]	slotSize	
	save	
	reset	
	setLED	
	getLED	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	setHBInterval	
	getHBInterval	
	getModule	
	getSerialNumber	
	getRouter	
	getModel	
usb[0]	setPortEnable	
	setPortDisable	
	setDataEnable	
	setDataDisable	
	setHiSpeedDataEnable	
	setHiSpeedDataDisable	
	setSuperSpeedDataEnable	
	setSuperSpeedDataDisable	
	setPowerEnable	
	setPowerDisable	
	getPortVoltage	
	getPortCurrent	
	getPortMode	
	setPortMode	
	getPortState	
	setCableFlip	
	getCableFlip	
	setConnectMode	
	getConnectMode	
	setCC1Enable	
	getCC1Enable	
	setCC2Enable	
	getCC2Enable	
	getCC1Voltage	
	getCC2Voltage	
	getCC1Current	
	getCC2Current	
	setSBUEnable	

continues on next page

Table 5 – continued from previous page

Entity Class	Entity Option	Variable(s)	Notes
mux[0]	getSBUEnable		
	setEnabled		
	getEnable		
	setChannel		
	getChannel		
	getConfiguration		
	setConfiguration		
	getSplitMode		
	setSplitMode		
	getVoltage	Channels 0-3	
equalizer[0-1]	setReceiverConfig		
	getReceiverConfig		
	setTransmitterConfig		
	getTransmitterConfig		

2.5 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.

2.5.1 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 \(BDK\)](#)⁶ 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

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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);
        return 1;
    } else { printf("Connected to BrainStem module.\n"); }

    //Basic initialization (Get LEDs turned off).
```

(continues on next page)

⁶ <https://acroname.com/software/brainstem-development-kit>

(continued from previous page)

```
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.

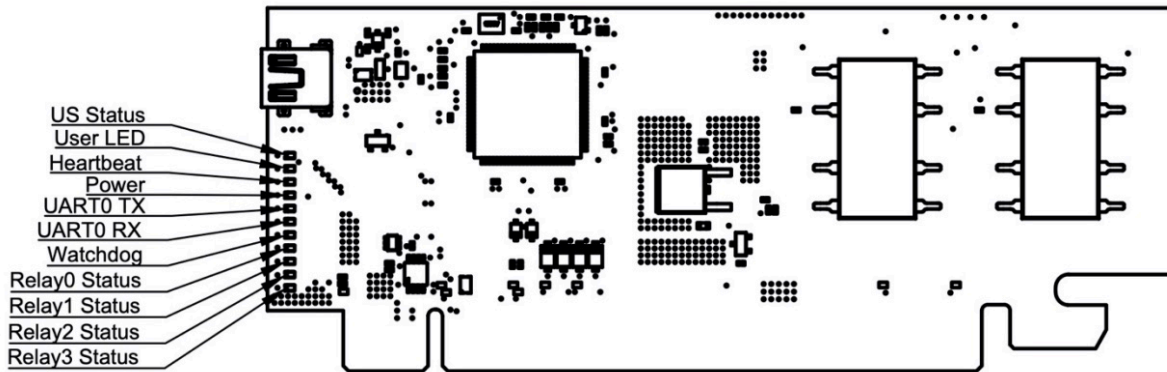


Figure 3: MTM-Relay LED Indicators

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

Relay

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

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() \[cpp\] \[python\] \[.NET\] \[LabVIEW\]
```

```
stem.relay[index].getEnabled() \[cpp\] \[python\] \[.NET\] \[LabVIEW\]
```

Get Voltage

Returns the voltage of the specified index.

```
stem.relay[index].getVoltage() \[cpp\] \[python\] \[.NET\] \[LabVIEW\]
```

Store

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

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 to load Reflex files (for details on Reflex, see [Reflex Language Reference](#)). One Reflex file can be stored per slot.

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

System

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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() \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [LabVIEW]
```

Get Input Voltage

Get the module's input voltage.

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

Complete list of Supported Entities and Functions

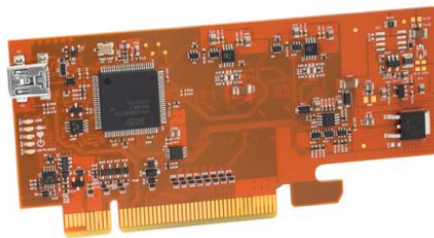
Entity Class	Entity Option	Variable(s) Notes
digital[0-3]	setConfiguration	
	getConfiguration	
	setState	
	getState	
i2c[0]	write	
	read	
relay[0-3]	setEnable	
	getEnable	

continues on next page

Table 6 – 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	

2.5.2 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 \(BDK\)](#)⁷ for you 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

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
    }
}
```

(continues on next page)

⁷ <https://acroname.com/software/brainstem-development-kit>

(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-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)

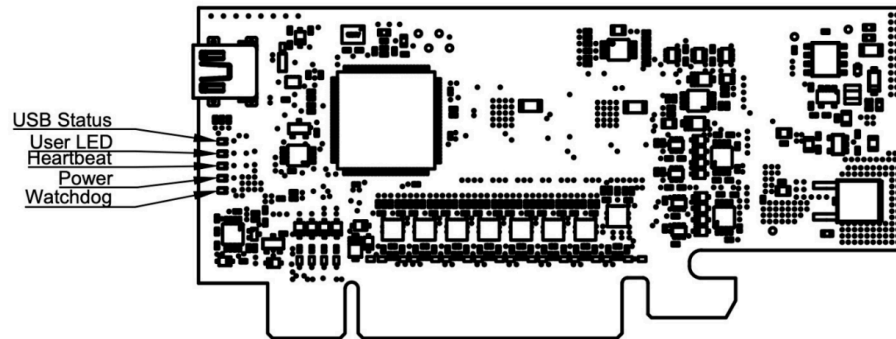
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```


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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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() \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [LabVIEW]
```

Get Input Voltage

Get the module's input voltage.

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

Store

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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 to load Reflex files (for details on Reflex, see [Reflex Language Reference](#)). One Reflex file can be stored per slot.

The MTM-DAQ-2 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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

Digital

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

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] [LabVIEW]
```

```
stem.digital[index].getConfiguration(mode) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.digital[index].getState(level) [cpp] [python] [NET] [LabVIEW]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.analog[index].getRange(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```


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] [LabVIEW]
```

```
stem.analog[index].setEnabled(mode) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.analog[index].getValue() [cpp] [python] [NET] [LabVIEW]
```

```
stem.analog[index].setVoltage(microvolts) [cpp] [python] [NET] [LabVIEW]
```

```
stem.analog[index].getVoltage() [cpp] [python] [NET] [LabVIEW]
```

I2C

API Documentation: [cpp] [python] [.NET] [LabVIEW]

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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] [LabVIEW]
```

```
stem.i2c[index].write(address,length) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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	
	getEnabled	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
system[0]	slotSize	
	save	
	reset	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	

continues on next page

Table 7 – 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		

2.5.3 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 \(BDK\)](#)⁸ 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

- Unzip the downloaded package
- Navigate to the “bin” folder
- Open HubTool

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"); }

    //Basic initialization (Get LEDs turned off).
```

(continues on next page)

⁸ <https://acroname.com/software/brainstem-development-kit>

(continued from previous page)

```
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)

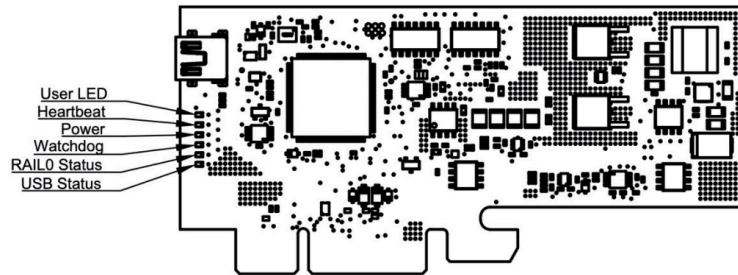
##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

# 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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

Rail

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.rail[index].getOperationalMode(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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].setOperationalState(state) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.rail[index].getOperationalState() \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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].setVoltage(microvolts) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.rail[index].getCurrentLimit(microamps) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.rail[index].getVoltage(microamps) [cpp] [python] [NET] [LabVIEW]
```

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].setKelvinSensingMode(bEnable) [cpp] [python] [NET] [LabVIEW]
```

```
stem.rail[index].getKelvinSensingMode(bEnable) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Store

API Documentation: [cpp] [python] [NET] [LabVIEW]

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 to load Reflex files (for details on Reflex, see [Reflex Language Reference](#)). One Reflex file can be stored per slot.

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

System

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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() \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [LabVIEW]
```

Get Input Voltage

Get the module's input voltage.

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [LabVIEW]
```


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] [LabVIEW]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

Temperature

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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].getTemperature(μC) [cpp] [python] [NET] [LabVIEW]
```

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	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	

continues on next page

Table 8 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	
temperature[0]	getTemperature	

2.5.4 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 \(BDK\)](#)⁹ 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

- Unzip the downloaded package
- Navigate to the “bin” folder
- Open HubTool

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;

    //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).
```

(continues on next page)

⁹ <https://acroname.com/software/brainstem-development-kit>

(continued from previous page)

```

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)

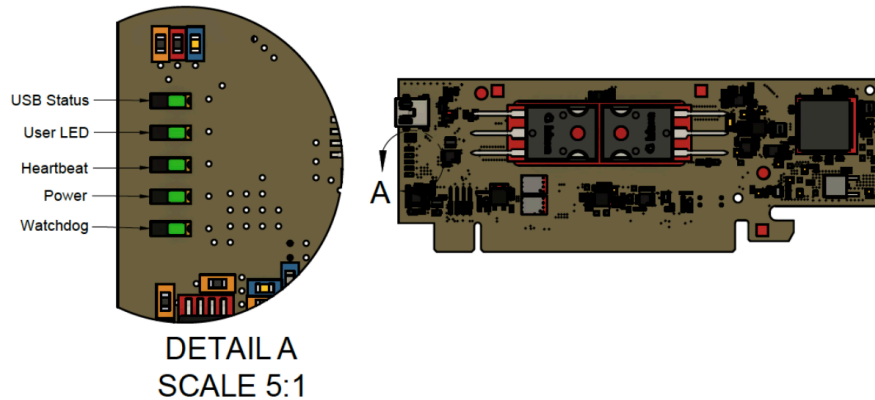
# Disconnect from device.
mtm.disconnect();
print("Disconnected from BrainStem module. \n")

```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```


Rail

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.rail[index].getOperationalMode(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.rail[index].getCurrentSetpoint(microvolts) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.rail[index].setVoltageMaxLimit (microvolts) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Store

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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 to load Reflex files (for details on Reflex, see [Reflex Language Reference](#)). One Reflex file can be stored per slot.

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

System

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

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [LabVIEW]
```

Get Input Voltage

Get the module's input voltage.

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```


Complete list of Supported Entities and Functions

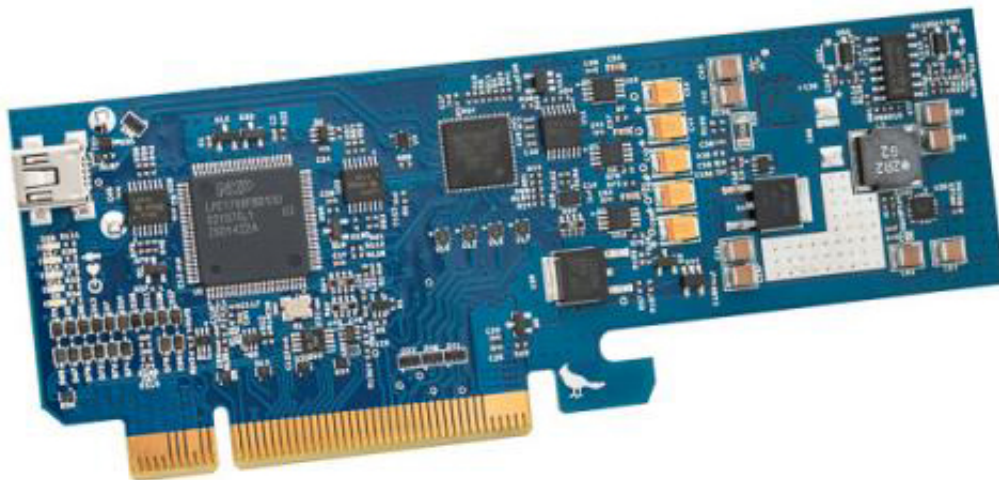
Entity Class	Entity Option	Variable(s)	Notes
digital[0-3]	setConfiguration		
	getConfiguration		
	setState		
	getState		
i2c[0]	write		
	read		
rail[0]	setPullup		Disabled by default.
	setCurrent		
	getCurrent		
	getCurrentSetpoint		
	setCurrentLimit		
	getCurrentLimit		
	getTemperature		
	setEnabled		
	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 9 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	

2.5.5 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 \(BDK\)](#)¹⁰ for you 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

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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"); }

    //Basic initialization (Get LEDs turned off).
```

(continues on next page)

¹⁰ <https://acroname.com/software/brainstem-development-kit>

(continued from previous page)

```
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)

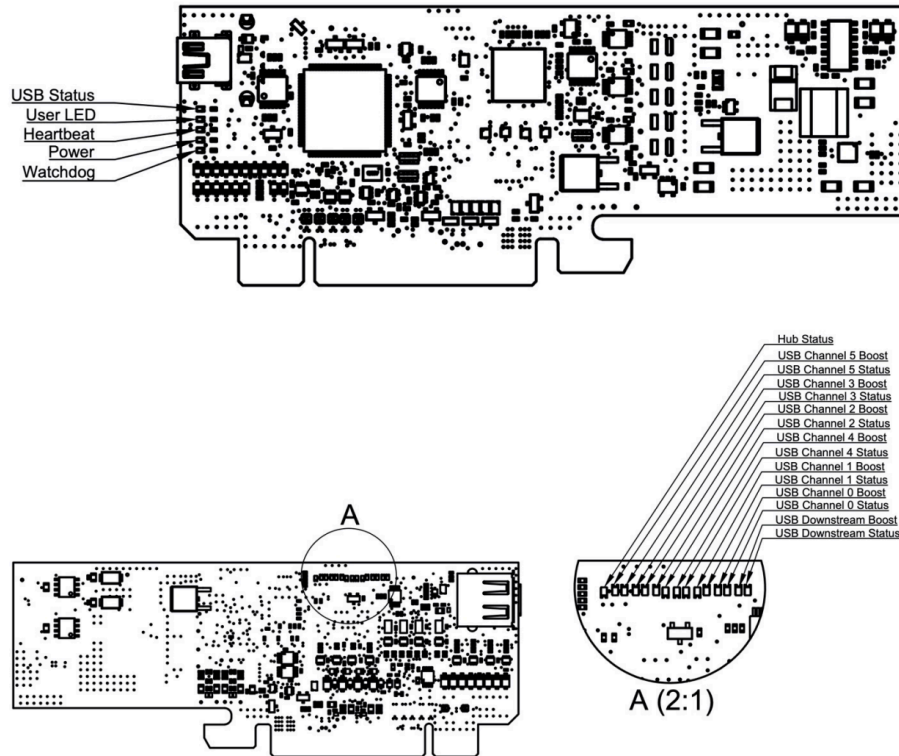
##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

# 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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

```
stem.digital[index].getState(level) [cpp] [python] [NET] [LabVIEW]
```

RCServo

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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].getConfiguration(digitalConfigurationRCServoInput/Output)
[cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.servo[index].getEnabled() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.servo[index].getPosition() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.servo[index].getReverse() [cpp] [python] [NET] [LabVIEW]
```


I2C

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

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

USB

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

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] [LabVIEW]
```

```
stem.usb.setPowerDisable(port) [cpp] [python] [NET] [LabVIEW]
```

Data

```
stem.usb.setDataEnable(port) [cpp] [python] [NET] [LabVIEW]
```

```
stem.usb.setDataDisable(port) [cpp] [python] [NET] [LabVIEW]
```

Port

```
stem.usb.setPortEnable(port) [cpp] [python] [NET] [LabVIEW]
```

```
stem.usb.setPortDisable(port) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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] [LabVIEW]
```

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.usb.setHiSpeedDataDisable(port) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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

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

To clear a Port error status

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

UART

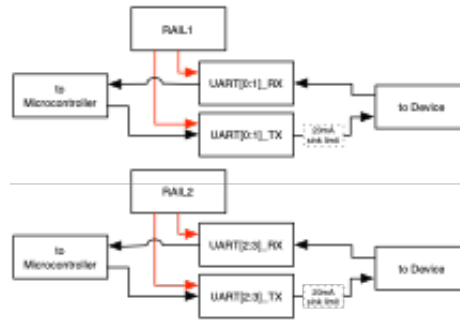
UART entities provide a mechanism to enable and disable UART data lines. All the UARTs that are passed down from the MTM-IO-Serial module can be turned on/off through software control. If a voltage is applied that is higher than the current rail voltage setpoint, each UART transmit line is current limited 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] [LabVIEW]
```

```
stem.uart[index].getEnable() [cpp] [python] [NET] [LabVIEW]
```



Rail

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.rail[index].getOperationalMode(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.rail[index].getVoltageSetpoint(microvolts) [cpp] [python] [NET] [LabVIEW]
```

Rail Voltage

Getting the rail voltage from any of the rails can be used by implementing

```
stem.rail[index].getVoltage(microvolts) [cpp] [python] [NET] [LabVIEW]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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].getConfiguration(digitalConfigurationSignalInput)      [cpp]
[python] [NET] [LabVIEW]
```

Pin	Input	Output	Rail	Hi-Z	Servo	Signal
DIO 0	Yes	Yes	1	No	Input	Input
DIO 1	Yes	Yes	1	No	Input	Input / Output
DIO 2	Yes	Yes	1	No	Input	Input / Output
DIO 3	Yes	Yes	1	No	Input	Input / Output
DIO 4	Yes	Yes	2	No	Output	Input / Output
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] [LabVIEW]
stem.signal[index].getEnable()           [cpp] [python] [NET] [LabVIEW]
```

Get/Set T3 Time

The T3 time defines the period of the waveform in nano seconds.

```
stem.signal[index].setT3Time(period) [cpp] [python] [NET] [LabVIEW]
stem.signal[index].getT3Time() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.signal[index].getT2Time() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
stem.signal[index].getInvert() [cpp] [python] [NET] [LabVIEW]
```

Store

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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 to load Reflex files (for details on Reflex, see [Reflex Language Reference](#)). One Reflex file can be stored per slot.

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

System

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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() \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [LabVIEW]
```

Get Input Voltage

Get the module's input voltage.

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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	

continues on next page

Table 10 – continued from previous page

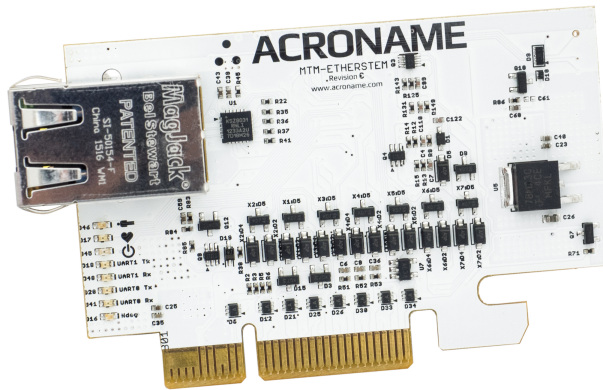
Entity Class	Entity Option	Variable(s) Notes
i2c[0]	setReverse	Index 4-7 only
	getReverse	
	write	
	read	
usb[0]	setPortEnable	
	setPortDisable	
	setDataEnable	
	setDataDisable	
	setHiSpeedDataEnable	
	setHiSpeedDataDisable	
	setPowerEnable	
	setPowerDisable	
	getPortError	
	clearPortErrorStatus	
	getSystemTemperature	In microcelsius
UART[0-3]	setUpstreamMode	
	getUpstreamMode	
	getUpstreamState	
	getDownstreamDataSpeed	
	getHubMode	
	setHubMode	
	getPortState	
	setEnabled	
rail[0]	setEnabled	
	setEnabled	
	getTemperature	In microcelsius
	setOperationalMode	
	getOperationalMode	
	getOperationalState	
	getVoltage	In microvolts
rail[1-2]	setEnabled	
	setEnabled	
	setVoltageSetpoint	In microvolts
	getVoltageSetpoint	In microvolts
signal[0-5]	getVoltage	In microvolts
	setEnabled	
	setEnabled	
	setInvert	
	getInvert	
	setT3Time	
	getT3Time	
	setT2Time	
store[0-1]	getT2Time	
	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	

continues on next page

Table 10 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
system[0]	slotCapacity	
	slotSize	
	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

2.5.6 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 \(BDK\)](#)¹¹ for you particular operating system and architecture.

¹¹ <https://acroname.com/software/brainstem-development-kit>

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

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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
```

(continues on next page)

(continued from previous page)

```
mtm.disconnect();  
}
```

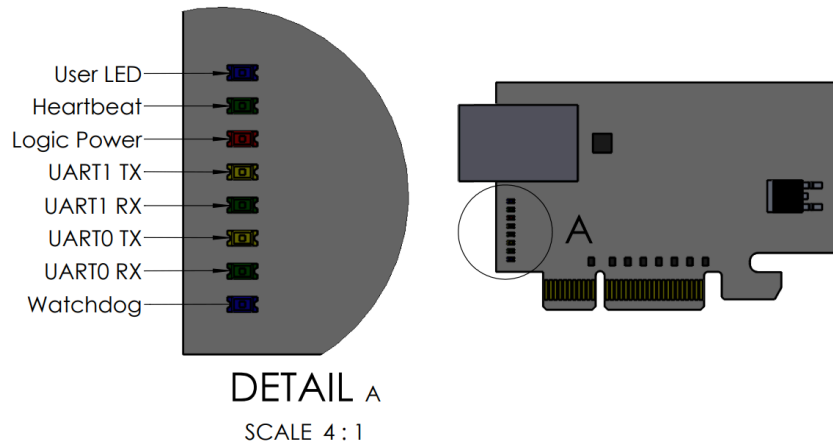
Python

```
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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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] [LabVIEW]
```

```
stem.digital[index].getState(level) [cpp] [python] [NET] [LabVIEW]
```

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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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] [LabVIEW]
```

```
stem.servo[index].getEnable() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.servo[index].getPosition() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.servo[index].getReverse() [cpp] [python] [NET] [LabVIEW]
```

I2C

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

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

Analog

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

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] [LabVIEW]
```

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] [LabVIEW]
stem.analog[index].getBulkCaptureSampleRate() [cpp] [python] [NET] [LabVIEW]
stem.analog[index].setBulkCaptureNumberOfSamples() [cpp] [python] [NET] [LabVIEW]
stem.analog[index].getBulkCaptureNumberOfSamples() [cpp] [python] [NET] [LabVIEW]
```

Get Bulk Capture State

```
stem.analog[index].getBulkCaptureState() [cpp] [python] [NET] [LabVIEW]
```

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[3]:

```
stem.analog[index].setValue() [cpp] [python] [NET] [LabVIEW]
stem.analog[index].getValue() [cpp] [python] [NET] [LabVIEW]
stem.analog[index].setVoltage(microvolts) [cpp] [python] [NET] [LabVIEW]
stem.analog[index].getVoltage() [cpp] [python] [NET] [LabVIEW]
```


Signal

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

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].getConfiguration(digitalConfigurationSignalInput)      [cpp]  
[python] [NET] [LabVIEW]
```

Get/Set Enable

```
stem.signal[index].setEnabled(bEnable)  [cpp] [python] [NET] [LabVIEW]  
stem.signal[index].getEnable()           [cpp] [python] [NET] [LabVIEW]
```

Get/Set T3 Time

The T3 time defines the period of the waveform in nano seconds.

```
stem.signal[index].setT3Time(period)  [cpp] [python] [NET] [LabVIEW]  
stem.signal[index].getT3Time()         [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]  
stem.signal[index].getT2Time()          [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.signal[index].getInvert() [cpp] [python] [NET] [LabVIEW]
```

Store

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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 to load Reflex files (for details on Reflex, see [Reflex Language Reference](#)). One Reflex file can be stored per slot.

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

System

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [LabVIEW]
```

Get Input Voltage

Get the module's input voltage.

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

Complete list of Supported Entities and Functions

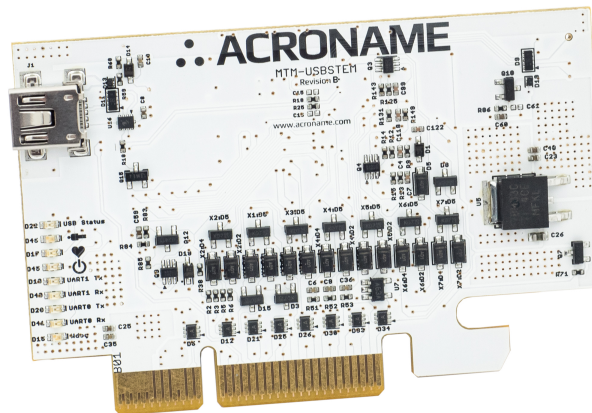
Entity Class	Entity Option	Variable(s) Notes
digital[0-14]	setConfiguration	
	getConfiguration	
	setState	
	getState	
rcservo[0-7]	setEnabled	
	getEnabled	
	setPosition	Index 4-7 only
	getPosition	
	setReverse	Index 4-7 only
	getReverse	
i2c[0-1]	write	
	read	
analog[0-2]	getValue	
	getVoltage	
	setBulkCaptureSampleRate	
	getBulkCaptureSampleRate	
	setBulkCaptureNumberOfSamples	

continues on next page

Table 11 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getBulkCaptureNumberOfSamples	
	initiateBulkCapture	
	getBulkCaptureState	
	setValue	
analog[3]	setVoltage	
signal[0-5]	setEnabled	
	getEnable	
	setInvert	
	getInvert	
store[0-2]	setT3Time	
	getT3Time	
	setT2Time	
	getT2Time	
	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
	save	
system[0]	reset	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

2.5.7 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 \(BDK\)](#)¹² for you particular operating system and architecture.

¹² <https://acroname.com/software/brainstem-development-kit>

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

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool

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"); }

    //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
```

(continues on next page)

(continued from previous page)

```
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)

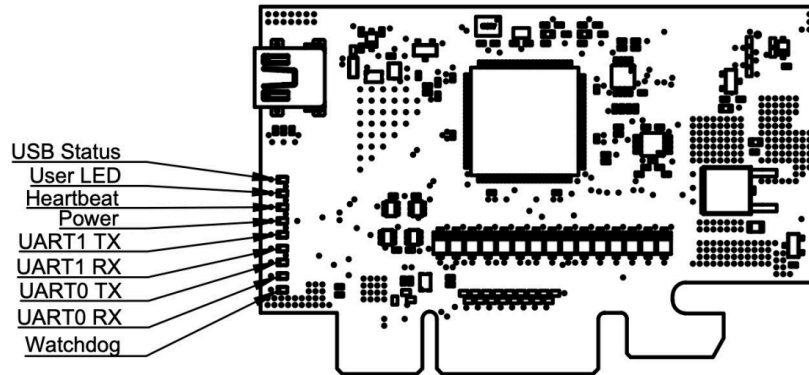
##Finished with testing.
##Turn off LED
mtm.system.setLED(0)

# 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: [\[cpp\]](#) [\[python\]](#) [\[.NET\]](#) [\[LabVIEW\]](#)

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getConfiguration(mode) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.digital[index].getState(level) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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

RCServo

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

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] [LabVIEW]
```

```
stem.servo[index].getEnable() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.servo[index].getPosition() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.servo[index].getReverse() [cpp] [python] [NET] [LabVIEW]
```

I2C

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

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

```
stem.i2c[index].write(address,length) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

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) \[cpp\] \[python\] \[NET\] \[LabVIEW\]
```

Analog

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

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] [LabVIEW]
```

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] [LabVIEW]
stem.analog[index].getBulkCaptureSampleRate() [cpp] [python] [NET] [LabVIEW]
stem.analog[index].setBulkCaptureNumberOfSamples() [cpp] [python] [NET] [LabVIEW]
stem.analog[index].getBulkCaptureNumberOfSamples() [cpp] [python] [NET] [LabVIEW]
```

Get Bulk Capture State

```
stem.analog[index].getBulkCaptureState() [cpp] [python] [NET] [LabVIEW]
```

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 ($0x0FFF \ll 4 = 0xFFFF0$) 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] [LabVIEW]
stem.analog[index].getValue() [cpp] [python] [NET] [LabVIEW]
stem.analog[index].setVoltage(microvolts) [cpp] [python] [NET] [LabVIEW]
stem.analog[index].getVoltage() [cpp] [python] [NET] [LabVIEW]
```


Signal

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

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].getConfiguration(digitalConfigurationSignalInput)      [cpp]  
[python] [NET] [LabVIEW]
```

Get/Set Enable

```
stem.signal[index].setEnabled(bEnable)  [cpp] [python] [NET] [LabVIEW]  
stem.signal[index].getEnable()  [cpp] [python] [NET] [LabVIEW]
```

Get/Set T3 Time

The T3 time defines the period of the waveform in nano seconds.

```
stem.signal[index].setT3Time(period)  [cpp] [python] [NET] [LabVIEW]  
stem.signal[index].getT3Time()  [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]  
stem.signal[index].getT2Time()  [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.signal[index].getInvert() [cpp] [python] [NET] [LabVIEW]
```

Store

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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 to load Reflex files (for details on Reflex, see [Reflex Language Reference](#)). One Reflex file can be stored per slot.

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

System

API Documentation: [cpp] [python] [.NET] [LabVIEW]

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setLED(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setBootSlot(value) [cpp] [python] [NET] [LabVIEW]
```

Get Input Voltage

Get the module's input voltage.

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

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] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setModuleSoftwareOffset(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setHBInterval(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

```
stem.system.setRouter(value) [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

Get Module

Get the address the module uses on the BrainStem network.

```
stem.system.getModule() [cpp] [python] [NET] [LabVIEW]
```

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] [LabVIEW]
```

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] [LabVIEW]
```

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
digital[0-14]	setConfiguration	
	getConfiguration	
	setState	
	getState	
rcservo[0-7]	setEnabled	
	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 12 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getBulkCaptureNumberOfSamples	
	initiateBulkCapture	
	getBulkCaptureState	
analog[3]	setValue	
	setVoltage	
signal[0-5]	setEnabled	
	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	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

API Reference

This API reference is organized by programming language. You will find product specific documentation in the [products](#) section.

3.1 BrainStem Entities

How does one describe the capabilities of an embedded system? Essentially this is the question that BrainStem entities answer. BrainStem modules are capable of I/O in the form of digitals, analogs, I²C, serial UARTs and other specialized interfaces. This section details how entities are referenced, and then describes the core Entities that BrainStem modules implement.

3.1.1 Entities

If you read the *Reflex Language* or *C++ API* sections of the reference, you will quickly see that Entities form the backbone of communication with BrainStem modules. They are the basic control mechanism for interacting with the BrainStem and the hardware to which it is connected. The following subsections describe the core entities that are available on most BrainStem modules. Less common and application specific entities will be described in the module's datasheet.

Entities usually describe a class of interaction, and usually are formed by a group of individual instances. For example; the digital entity is made up of multiple digital I/Os, which can be manipulated individually. In general the following form applies to an Entity.

```
Module . EntityClass [ Element Index ] . operation ( parameters )
```

To further the digital example, the 4th digital output of a module can be set to logic high with the Reflex language via the following syntax.

```
stem.digital[3].setState(1);
```

Indices for entities always start with a zero index.

Single individual element entities like the System entity can be addressed in the Reflex programming language without the `[]` syntax, however this is just a convenience and any individual can always be addressed explicitly.

```
// Explicit reference.
stem.system[0].setLED(1);

// Implicit reference.
stem.system.setLED(1);
```

Entities are so fundamental that they form one of the elements of the BrainStem communication protocol. The protocol specifics are detailed in the following appendices:

- *Appendix: Brainstem Universal Entity Interface*
- *Appendix: The BrainStem Communication Protocol*

3.1.2 Analog Entity

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

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 ($0x0FFF =: 0x0FFF \ll 4 = 0xFFF0$) 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 ($0x8000 =: 0x8000 \gg 5 = 0x0400$) 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);
```

Reflex

```
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
```

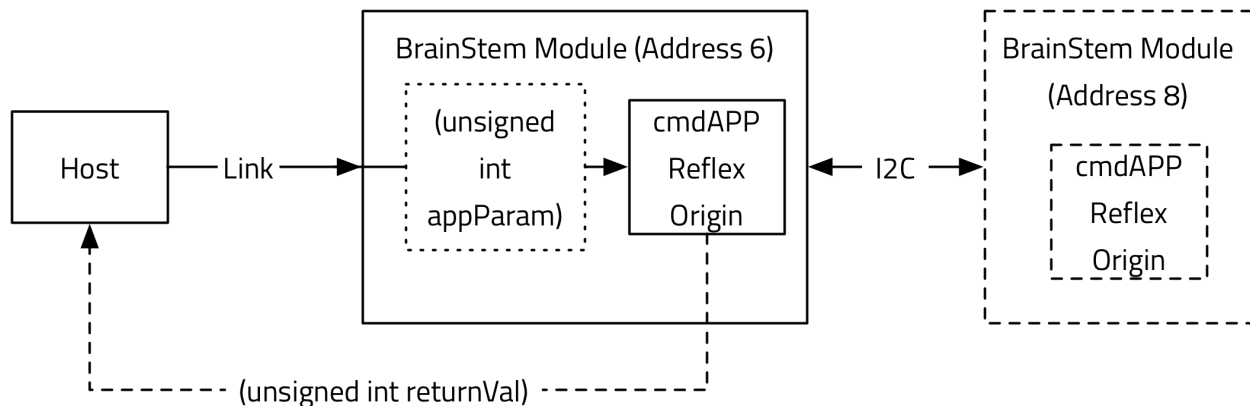
Python

```
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
```

3.1.3 App Entity

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

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
```

Reflex

```
// Somewhere in a Reflex file
reflex app[0](int appParam) {
    // do interesting things
}

stem.app[0].execute(3131948783); // triggers the App reflex handle and passes 4 bytes.
↳to it
```

Python

Implementation coming **in** future release.

3.1.4 Clock Entity

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

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);
```

(continues on next page)

(continued from previous page)

```
stem.clock.getHour(hour);
stem.clock.setHour(hour);
stem.clock.getMinute(minute);
stem.clock.setMinute(minute);
stem.clock.getSecond(second);
stem.clock.setSecond(second);
```

Reflex

```
// Get requests fill the variable with the 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);
```

3.1.5 Digital Entity

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

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

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.

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);
```

Reflex

```
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
```

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
```

3.1.6 Equalizer Entity

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

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 on 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 I2C Entity

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

BrainStem modules may have the ability to read, write data on up to 2 I2C bus's

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.

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
```

Reflex

Currently this entity is not available from within the reflex language.

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(0x42,0x02, b'\xbe\xef') # writes b'\xbe\xef' to the i2c bus.
print err
err = stem.i2c[0].setPullup(True)
print err
```

3.1.8 Mux Entity

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.

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].setEnable(1);
```

(continues on next page)

(continued from previous page)

```
err = stem.mux[0].setEnabled(0);
err = stem.mux[0].getChannel(1, &voltage);
err = stem.mux[0].setChannel(3);
```

Reflex

```
//Get calls will fill the variable with the returned value.

stem.mux[0].getChannel(&channel);
stem.mux[0].setChannel(1);
stem.mux[0].setEnabled(1);
stem.mux[0].setEnabled(0);
stem.mux[0].getChannel(1, &voltage);
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)
```

3.1.9 Pointer Entity

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

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 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 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);
```

(continues on next page)

(continued from previous page)

```
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);
```

Reflex

//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)
```

3.1.10 Port Entity

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

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 the 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.

3.1.11 Power Delivery Entity

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

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.

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 PDO's are exist: Remote Sink, Remote Source, Local Sink, and Local Source. Within each of PDO locations up to 7 PDO's 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 the 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 PDO's 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) voltage
```

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.

Cable Orientation (Get)

```
pd[x] . getCableOrientation => (unsigned char) orientation
```

Indicates which side of the connection is being using for PD negotiations. This is based on physical CC strapping within the cable.

Request (Set)

```
pd[x] . getCableOrientation <= (unsigned char) request
```

Allows access to specific request which are built into the PD specification. It's important to remember that these are requests and are not guaranteed to occur. Examples are resets, power, data, vconn role swaps etc.

Table 1: 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] . getFlagMode <= (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 2: 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 3: 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

3.1.12 Rail Entity

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

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.

```

(continues on next page)

(continued from previous page)

```

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);

```

Reflex

```

// 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].setEnable(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].setEnable(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

```

(continues on next page)

(continued from previous page)

```
state = stem.rail[0].getOperationalState(0);
print state.value
```

3.1.13 RCServo Entity

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

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  
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)
```

3.1.14 Relay Entity

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

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);
```

(continues on next page)

(continued from previous page)

```
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);
```

3.1.15 Signal Entity

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

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.

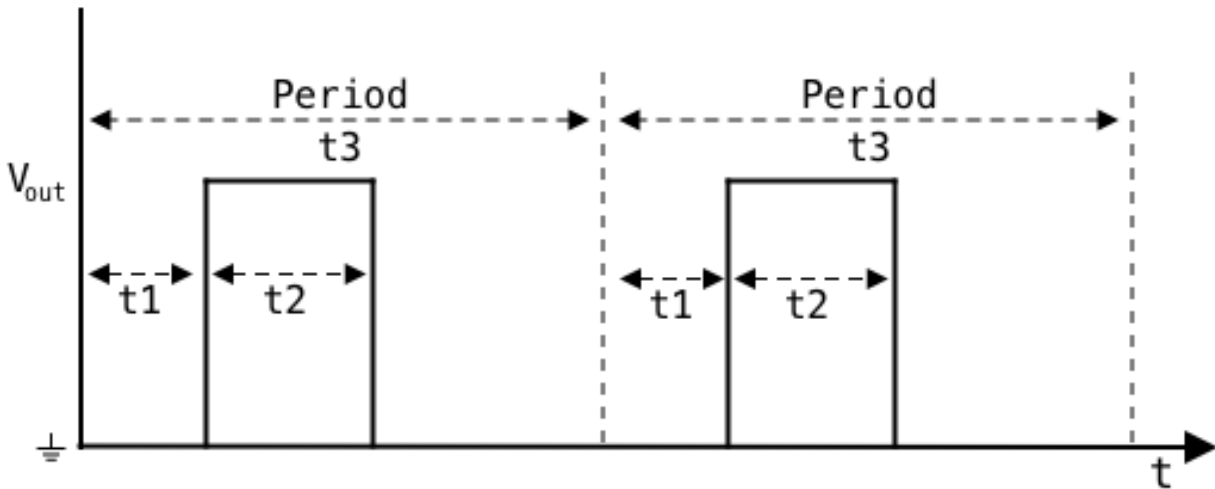
Timing

Set/Get enable

```
signal [ index ] . getEnable <= (unsigned char) enable
signal [ index ] . setEnable => (unsigned char) enable
```

Enables the Signal Entity for a given index.

Signal Entity Timing Diagram



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;
```

3.1.16 Store Entity

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

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.
```

Reflex

```
stem.store[0].getSlotState(3, state);
stem.store[0].enableSlot(3);
stem.store[0].disableSlot(3);
stem.store[0].getCapacity(1, capacity);
stem.store[0].getSize(1, size);
```

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
```

3.1.17 System Entity

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

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.

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.

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.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.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)
```

(continues on next page)

(continued from previous page)

```
hbInterval = stem.system.getHBInterval()
print hbInterval.value
stem.system.setHBInterval(12)
```

Reflex

```
// Get requests fill the parameter with the current system value upon success.

stem.system.save();
stem.system.reset();
stem.system.setLED(1);
stem.system.getLED(state);
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);
```

3.1.18 Temperature Entity

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

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.

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);
```

Reflex

```
//Get commands fill the variable with the returned value.  
stem.temperature[0].getTemperature(microcelsius);
```

Python

```
microcelsius = stem.temperature[0].getTemperature();  
print microcelsius.value
```

3.1.19 Timer Entity

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

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.

Reflex example

The following reflex code would need to be compiled with arc, loaded onto the BrainStem module and enabled to be executed. See the [Reflex Language reference](#) for more information about working with reflex files.

```
reflex timer[0].expiration(void) {
    stem.system.setLED(on);
}
```

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.
```

Reflex

```
// 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.
```

3.1.20 UART Entity

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

The UART entity is a class which allows the configuration of a specified uart port.

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 Baudrate (Get/Set)

```
uart [ index ] . setBaudRate => (unsigned int) rate  
uart [ index ] . getBaudRate <= (unsigned int) rate
```

Allows for get and set of the uart channel's baudrate.

Change Protocol (Get/Set)

```
uart [ index ] . setProtocol => (unsigned char) protocol  
uart [ index ] . getProtocol <= (unsigned char) protocol
```

Allows for get and set of the uart channel's protocol if there are different protocols.

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].setEnable(1);  
err = stem.uart[1].setEnable(1);  
  
err = stem.uart[0].getEnable(&enable);  
err = stem.uart[1].getEnable(&enable);  
  
err = stem.uart[0].setEnable(0);  
err = stem.uart[1].setEnable(0);
```

Python

```
err = stem.uart[0].setEnable(1);  
err = stem.uart[1].setEnable(1);  
  
result = stem.uart[0].getEnable()  
print result.value  
  
result = stem.uart[1].getEnable()  
print result.value  
  
err = stem.uart[0].setEnable(0);  
err = stem.uart[1].setEnable(0);
```

3.1.21 USB Entity

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

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 setSuperSpeedDataEnable/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. Options are Auto, 0 or 1

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 . setCableFlip => (unsigned char) channel, (unsigned char) bEnable  
usb . setCableFlip (unsigned char) channel <= (unsigned char) bEnable
```

Change the orientation of the common side to Mux side cable connection.

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);  
...
```

Reflex

```
// 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  
stem.usb.setPortCurrentLimit(0, limit_setting)  
state = stem.usb.getHubState()  
print state.value  
...
```

3.1.22 USB System Entity

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

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

Upstream Connection (Get/Set)

```
usbsystem . setUpstream => (unsigned char) enable
usbsystem . getUpstream <= (unsigned char) enable
```

Many acroname products have multiple upstream port selections. This function is used to access and control that functionality.

Enumeration Delay (Get/Set)

```
usbsystem . getEnumerationDelay <= (unsigned int) ms_delay
usbsystem . 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 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 . getModelList <= (unsigned int [NUM_PORTS]) list
usbsystem . setModelList => (unsigned int [NUM_PORTS]) list
```

The state list function gives you access and control to all lower level port states. It is equivalent to calling get/set state from the [PortClass](#) on all ports at once.

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.

Selector Mode (Get/Set)

```
usbsystem . getSelectorMode <= (unsigned char) mode  
usbsystem . setSelectorMode => (unsigned char) mode
```

The selector mode defines what will happen if the external selector/trigger input is used. The selector input allows for physical (button) control of the Acroname product. This feature is product specific and users should consult the manual for further details.

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](#)¹³, 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 python 2.7 and python 3.6 through 3.10. When using 2.7 it is recommended that your python version be at least 2.7.9.

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>

¹³ <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:

- *USBHub3p*
- *USBHub2x4*
- *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()
# 0xFFFFFFFF is the serial number of the module.
>>> stem.connect(0xFFFFFFFF)
```

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.

USBHub3p

class `brainstem.stem.USBHub3p` (*address=6, enable_auto_networking=True, model=19*)

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)

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)

connect (*serial_number*, ***kwargs*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

USBHub2x4

```
class brainstem.stem.USBHub2x4 (address=6, enable_auto_networking=True,  
                                model=17)
```

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)

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)

connect (*serial_number*, ***kwargs*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in brainstem.result

Return type

Result.error

[Back to the top](#)

USBCSwitch

```
class brainstem.stem.USBCSwitch (address=6, enable_auto_networking=True,  
                                model=21)
```

Concrete Module implementation for the USBC-Switch.

The module contains the USB entity as well as the following.

Entities:

- system
- app[0-3]
- pointer[0-3]
- usb
- mux
- store[0-1]
- timer[0-7]
- equalizer[0-1]

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_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)
- `usbPortStateHiSpeed` (1)
- `usbPortStateSBU` (2)
- `usbPortStateSS1` (3)
- `usbPortStateSS2` (4)
- `usbPortStateCC1` (5)
- `usbPortStateCC2` (6)
- `usbPortStateCCFlip` (13)
- `usbPortStateSSFlip` (14)
- `usbPortStateSBUFlip` (15)
- `usbPortStateErrorFlag` (19)
- `usbPortStateUSB2Boost` (20)
- `usbPortStateUSB3Boost` (21)
- `usbPortStateConnectionEstablished` (22)
- `usbPortStateCC1Inject` (26)
- `usbPortStateCC2Inject` (27)
- `usbPortStateCC1Detect` (28)
- `usbPortStateCC2Detect` (29)
- `usbPortStateCC1LogicState` (30)
- `usbPortStateCC2LogicState` (31)
- `usbPortStateOff` (0)
- `usbPortStateSideA` (1)
- `usbPortStateSideB` (2)
- `usbPortStateSideUndefined` (3)
- `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*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

MTMDAQ2

class `brainstem.stem.MTMDAQ2` (*address=10, enable_auto_networking=True, model=22*)

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` (14)
- `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*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

MTMEtherStem

```
class brainstem.stem.MTMEtherStem(address=4, enable_auto_networking=True,  
                                model=15)
```

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*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

MTMIOSerial

```
class brainstem.stem.MTMIOSerial (address=8, enable_auto_networking=True,
                                  model=13)
```

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)
- 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)

connect (*serial_number*, ****kwargs**)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

MTMLOAD1

```
class brainstem.stem.MTMLOAD1 (address=14, enable_auto_networking=True,  
                                model=23)
```

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)
```

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in brainstem.result

Return type

Result.error

[Back to the top](#)

MTMPM1

```
class brainstem.stem.MTMPM1 (address=6, enable_auto_networking=True, model=14)
```

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)
```

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in brainstem.result

Return type

Result.error

[Back to the top](#)

MTMRelay

```
class brainstem.stem.MTMRelay (address=12, enable_auto_networking=True,  
                                model=18)
```

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)
```

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in brainstem.result

Return type

Result.error

[Back to the top](#)

MTMUSBStem

```
class brainstem.stem.MTMUSBStem (address=4, enable_auto_networking=True,  
                                   model=16)
```

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*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

MTMDAQ1

class `brainstem.stem.MTMDAQ1` (*address=10, enable_auto_networking=True, model=20*)

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*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

EtherStem

```
class brainstem.stem.EtherStem (address=2, enable_auto_networking=True,  
                                model=5)
```

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*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[Back to the top](#)

USBStem

class `brainstem.stem.USBStem` (*address=2, enable_auto_networking=True, model=4*)

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*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

[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 BraiStem Modules and Entities. For more information about Brainstem Modules and Entities, please see the [Terminology](#)¹⁴ section of the [Acroname BrainStem Reference](#)¹⁵

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](#)¹⁶ section of the [Acroname BrainStem Reference](#)¹⁷

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](#)¹⁸

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](#)¹⁹

¹⁴ <https://acroname.com/reference/terms.html>

¹⁵ <https://acroname.com/reference>

¹⁶ <https://acroname.com/reference/entities>

¹⁷ <https://acroname.com/reference>

¹⁸ <https://acroname.com/reference>

¹⁹ <https://acroname.com/reference>

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](https://acroname.com/reference)²⁰

brainstem.version

Provides version access utilities.

3.2.4 Analog

class `brainstem.entity.Analog (module, index)`

The AnalogClass is the 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.

Useful constants:

- CONFIGURATION_INPUT (0)
- CONFIGURATION_OUTPUT (1)
- HERTZ_MINIMUM (7,000)
- HERTZ_MAXIMUM (200,000)
- BULK_CAPTURE_IDLE (0)
- BULK_CAPTURE_PENDING (1)
- BULK_CAPTURE_FINISHED (2)
- BULK_CAPTURE_ERROR (3)

getBulkCaptureNumberOfSamples ()

Get the current number of samples setting for this analog when bulk capturing.

Returns

Result object, containing NO_ERROR and sample number
or a non zero Error code.

Return type

Result

²⁰ <https://acroname.com/reference>

getBulkCaptureSampleRate()

Get the current sample rate setting for this analog when bulk capturing.

Sample rate is in samples per second (Hertz).

Returns

Result object, containing NO_ERROR and sample rate
or a non zero Error code.

Return type

Result

getBulkCaptureState()

Get the current bulk capture state for this analog.

Possible states of the bulk capture operation are; idle = 0 pending = 1 finished = 2 error = 3

Returns

Result object, containing NO_ERROR and bulk capture state
or a non zero Error code.

Return type

Result

getConfiguration()

Get the analog configuration.

If the configuration is 1 the analog is configured as an output, if the configuration is 0, the analog is set as an input.

Returns

Result object, containing NO_ERROR and analog configuration
or a non zero Error code.

Return type

Result

getEnable()

Get the enable state an analog output

Get a boolean value corresponding to on/off

Returns

Result object, containing NO_ERROR and enable state
or a non zero Error code.

Return type

Result

getRange()

Get the range setting of an analog input

Get a value corresponding to a discrete range option.

Returns

Result object, containing NO_ERROR and analog range
or a non zero Error code.

Return type

Result

getValue()

Get the raw ADC value in bits.

Get a 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 provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit ADC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Returns**Result object, containing NO_ERROR and analog value**

or a non zero Error code.

Return type*Result***getVoltage ()**

Get the scaled micro volt value with reference to ground.

Get a 32 bit signed integer (in microVolts) 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.

Returns**Result object, containing NO_ERROR and microVolts value**

or a non zero Error code.

Return type*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.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure. When the bulk capture is complete getBulkCaptureState() will return either finished or error.

Return type

Result.error

setBulkCaptureNumberOfSamples (value)

Set the number of samples to capture for this analog when bulk capturing.

Minimum # of Samples: 0 Maximum # of Samples: (BRAINSTEM_RAM_SLOT_SIZE / 2) = (3FFF / 2) = 1FFF = 8191

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setBulkCaptureSampleRate (value)

Set the sample rate for this analog when bulk capturing.

Sample rate is set in samples per second (Hertz).

Minimum Rate: 7,000 Hertz Maximum Rate: 200,000 Hertz

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setConfiguration (*value*)

Set the analog configuration.

Some analogs can be configured as DAC outputs. Please see your module datasheet to determine which analogs can be configured as DAC.

Param:

value (int): Set 1 for output 0 for input. Default configuration is input.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setEnabled (*enable*)

Set the enable state of an analog output.

Set a boolean value corresponding to on/off

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setRange (*value*)

Set the range of an analog input.

Set a value corresponding to a discrete range option.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setValue (*value*)

Set the value of an analog output (DAC) in bits.

Set a 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 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.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setVoltage (*value*)

Set the voltage level of an analog output (DAC) in microVolts with reference to ground.

Set a 16 bit signed integer as voltage output (in microVolts).

Note: Voltage range is dependent on the specific DAC channel range. See datasheet and setRange for options.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.5 App

class `brainstem.entity.App` (*module, index*)

The AppClass calls defined app reflexes on brainstem modules.

Calls a remote procedure defined in an active map file on a brainstem module. The remote procedure may return a value or not.

execute (*param*)

Execute an App reflex on a module.

Param:

param (int): App routine parameter.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

executeAndWaitForReturn (*param, msTimeout*)

Execute an App reflex on a module, and wait for it to return a result.

Param:

param (int): App routine parameter.

Param:

msTimeout (int): milliseconds to wait for routine to complete.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.6 Clock

class `brainstem.entity.Clock (module, index)`

The ClockClass is the interface to the realtime clock.

For modules that support realtime clocks, this class supports getting and setting clock values for year, month, day, hour, minute and second.

getDay ()

Get the current day of the month

Returns

Result object, containing NO_ERROR and current day or
a non zero Error code.

Return type

Result

getHour ()

Get the current hour

Returns

Result object, containing NO_ERROR and current hour or
a non zero Error code.

Return type

Result

getMinute ()

Get the current minute

Returns

Result object, containing NO_ERROR and current minute or
a non zero Error code.

Return type

Result

getMonth ()

Get the current month

Returns

Result object, containing NO_ERROR and current month or
a non zero Error code.

Return type

Result

getSecond ()

Get the current second

Returns

Result object, containing NO_ERROR and current second or
a non zero Error code.

Return type

Result

getYear ()

Get the current year

Returns

Result object, containing NO_ERROR and current year or
a non zero Error code.

Return type

Result

setDay (*day*)

Set the current day of the month

Param:

value (int): Current 2 digit day.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setHour (*hour*)

Set the current hour

Param:

value (int): Current 2 digit hour.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setMinute (*minute*)

Set the current minute

Param:

value (int): Current 2 digit minute.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setMonth (*month*)

Set the current month

Param:

value (int): Current 2 digit month.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setSecond (*second*)

Set the current second

Param:

value (int): Current 2 digit second.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setYear (*year*)

Set the current year

Param:

value (int): Current 4 digit year.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.7 Definitions

A module that provides defines and constants useful for working with the python library.

```
brainstem.defs.MODEL_ETHERSTEM = 5
```

EtherStem Model number

```
brainstem.defs.MODEL_MTM_DAQ_1 = 20
```

MTM-DAQ-1 Model number

```
brainstem.defs.MODEL_MTM_DAQ_2 = 22
```

MTM-DAQ-2 Model number

```
brainstem.defs.MODEL_MTM_ETHERSTEM = 15
```

MTM EtherStem Model number

```
brainstem.defs.MODEL_MTM_IO SERIAL = 13
```

MTM-IO-Serial Model number

```
brainstem.defs.MODEL_MTM_PM_1 = 14
```

MTM-PM-1 Model number

```
brainstem.defs.MODEL_MTM_RELAY = 18
```

MTM-Relay Model number

```
brainstem.defs.MODEL_MTM_USBSTEM = 16
```

MTM USBStem Model number

```
brainstem.defs.MODEL_USBHUB_2X4 = 17
```

USBHub 2x4 Model number

```
brainstem.defs.MODEL_USBHUB_3C = 24
```

USBHub3c Model number

```
brainstem.defs.MODEL_USBHUB_3P = 19
```

USBHub 3+ Model number

```
brainstem.defs.MODEL_USBSTEM = 4
```

USBStem Model number

```
brainstem.defs.MODEL_USB_C_SWITCH = 21
```

USBC-Switch Model number

```
brainstem.defs.model_info(model)
```

Get Model information.

Parameters

model (*int*) – One of the model numbers, i.e from stem.system.getModel().

Returns

Return a string containing model information.

Return type

str

```
brainstem.defs.model_name(model)
```

Get Model Name.

Parameters

model (*int*) – One of the model numbers, i.e from stem.system.getModel().

Returns

Return a string containing model name.

Return type

str

3.2.8 Digital

```
class brainstem.entity.Digital(module, index)
```

The DigitalClass is the 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.

Useful Constants:

- VALUE_LOW (0)
- VALUE_HIGH (1)
- CONFIGURATION_INPUT (0)
- CONFIGURATION_OUTPUT (1)
- CONFIGURATION_RCSERVO_INPUT (2)
- CONFIGURATION_RCSERVO_OUTPUT (3)
- CONFIGURATION_HIGHZ (4)
- CONFIGURATION_INPUT_PULL_UP (0)
- CONFIGURATION_INPUT_NO_PULL (4)
- CONFIGURATION_INPUT_PULL_DOWN (5)
- CONFIGURATION_SIGNAL_OUTPUT (6)
- CONFIGURATION_SIGNAL_INPUT (7)

```
getConfiguration()
```

Get the digital configuration.

If the configuration is 1 the digital is configured as an output, if the configuration is 0, the digital is set as an input.

Returns

Result object, containing NO_ERROR and digital configuration or a non zero Error code.

Return type

Result

getState ()

Get the digital state.

A return of 1 indicates the digital is above the logic high threshold. A return of 0 indicates the digital is below the logic low threshold.

Returns

Result object, containing NO_ERROR and digital state or a non zero Error code.

Return type

Result

getStateAll ()

Gets the digital state of all digitals in a bit mapped representation. Number of digitals varies across BrainStem modules. Refer to then datasheet for the capabilities of your module.

Returns

Result object, containing NO_ERROR and the digital state of all digitals where bit 0 = digital 0 and bit 1 = digital 1 etc. 0 = logic low and 1 = logic high. A non zero Error code is returned on error.

Return type

Result

setConfiguration (configuration)

Set the digital configuration.

Param:**configuration (int):**

- Digital Input: CONFIGURATION_INPUT = 0
- Digital Output: CONFIGURATION_OUTPUT = 1
- RCServo Input: CONFIGURATION_RCSERVO_INPUT = 2
- RCServo Output: CONFIGURATION_RCSERVO_OUTPUT = 3
- High Z State: CONFIGURATION_HIGHZ = 4
- Digital Input with pull up: CONFIGURATION_INPUT_PULL_UP = 0 (Default)
- Digital Input with no pull up or pull down: CONFIGURATION_INPUT_NO_PULL = 4
- Digital Input with pull down: CONFIGURATION_INPUT_PULL_DOWN = 5
- Digital Signal Output: CONFIGURATION_SIGNAL_OUTPUT = 6
- Digital Signal Input: CONFIGURATION_SIGNAL_INPUT = 7

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setState (state)

Set the digital state.

Param:**state (int):**

Set 1 for logic high, set 0 for logic low. configuration must be set to output.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setStateAll (state)

Sets the digital state of all digitals based on the bit mapping. Number of digitals varies across BrainStem modules. Refer to then datasheet for the capabilities of your module.

Param:**state (uint):**

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. Configuration must be set to output.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.9 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)²¹

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.
- 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)

²¹ <https://acroname.com/reference>

– 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)`

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 (*list (int)*) – A list of transports or a single transport.

Returns

A list of the Specs for all modules found.

Return type

list(Spec)

`brainstem.discover.findFirstModule (transport)`

Return the Spec for the first module found on the given transport.

TCPIP modules take a little longer to find due to the Multicast and gather necessary for finding modules on the local network segment.

args:

`transport (int)`: One of USB or TCPIP.

return:

Spec: The connection spec of the first module found on the given transport.

`brainstem.discover.findModule (transports, serial_number)`

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** (*list (int)*) – A list of transports or a single transport.
- **serial_number** (*int*) – The module serial_number to look for.

Returns

The connection spec for the module whose serial number is given in the args.

Return type

Spec

`brainstem.discover.getDownstreamDevices (list_length=128)`

Gets downstream device USB information for all Acroname hubs.

args:

`list_length`: The amount of memory to provide for the lower level C call.

Return:

Result: 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.

3.2.10 Entities

Quick Access:

- *System*
- *Analog*
- *Digital*
- *Equalizer*
- *I2C*
- *Mux*
- *Pointer*
- *Rail*
- *Store*
- *Temperature*
- *Timer*
- *USB*
- *RCServo*
- *Relay*

Entities are also accessed via the module, generally they are represented as an array member of the Module class. However entities that are singular like System can be accessed directly.

```
>>> stem.system.getBootSlot()
>>> stem.i2c[0].read(0x90, 0x02)
```

System

class brainstem.stem.**System** (*module, index*)

Access system controls configuration and information.

The system entity is available on all BrainStem modules, and provides access to system information such as module, router and serial number, as well as control over the user LED, and information such as the system input voltage.

Useful Constants:

- BOOT_SLOT_DISABLE (255)

[Back to the top](#)

Analog

class `brainstem.stem.Analog (module, index)`

The AnalogClass is the 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.

Useful constants:

- `CONFIGURATION_INPUT` (0)
- `CONFIGURATION_OUTPUT` (1)
- `HERTZ_MINIMUM` (7,000)
- `HERTZ_MAXIMUM` (200,000)
- `BULK_CAPTURE_IDLE` (0)
- `BULK_CAPTURE_PENDING` (1)
- `BULK_CAPTURE_FINISHED` (2)
- `BULK_CAPTURE_ERROR` (3)

[Back to the top](#)

Digital

class `brainstem.stem.Digital (module, index)`

The DigitalClass is the 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.

Useful Constants:

- `VALUE_LOW` (0)
- `VALUE_HIGH` (1)
- `CONFIGURATION_INPUT` (0)
- `CONFIGURATION_OUTPUT` (1)
- `CONFIGURATION_RCSERVO_INPUT` (2)
- `CONFIGURATION_RCSERVO_OUTPUT` (3)
- `CONFIGURATION_HIGHZ` (4)
- `CONFIGURATION_INPUT_PULL_UP` (0)
- `CONFIGURATION_INPUT_NO_PULL` (4)
- `CONFIGURATION_INPUT_PULL_DOWN` (5)
- `CONFIGURATION_SIGNAL_OUTPUT` (6)
- `CONFIGURATION_SIGNAL_INPUT` (7)

[Back to the top](#)

Digital

class `brainstem.stem.Equalizer (module, index)`

Equalizer Class provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

[Back to the top](#)

I2C

class `brainstem.stem.I2C (module, index)`

The I2C class is the interface the I2C busses on BrainStem modules.

The class provides a way to send read and write commands to I2C devices on the entitie's bus.

Useful Constants:

- `I2C_DEFAULT_SPEED` (0)
- `I2C_SPEED_100Khz` (1)
- `I2C_SPEED_400Khz` (2)
- `I2C_SPEED_1000Khz` (3)

[Back to the top](#)

Mux

class `brainstem.stem.Mux (module, index)`

Access MUX specialized entities on certain BrainStem modules.

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 (multiplexor) 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).

Useful Constants:

- `UPSTREAM_STATE_ONBOARD` (0)
- `UPSTREAM_STATE_EDGE` (1)
- `UPSTREAM_MODE_AUTO` (0)
- `UPSTREAM_MODE_ONBOARD` (1)
- `UPSTREAM_MODE_EDGE` (2)
- `DEFAULT_MODE` (`UPSTREAM_MODE_AUTO`)

[Back to the top](#)

Pointer

class `brainstem.stem.Pointer` (*module, index*)

Access 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 and incrementing pointer.

Useful Constants:

- `POINTER_MODE_STATIC` (0)
- `POINTER_MODE_INCREMENT` (1)

[Back to the top](#)

Rail

class `brainstem.stem.Rail` (*module, index*)

Provides power rail functionality on certain modules.

This entity is only available on certain modules. The RailClass can be used to control power to downstream devices, I has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

Useful Constants:

- `KELVIN_SENSING_OFF` (0)
- `KELVIN_SENSING_ON` (1)
- `OPERATIONAL_MODE_AUTO` (0)
- `OPERATIONAL_MODE_LINEAR` (1)
- `OPERATIONAL_MODE_SWITCHER` (2)
- `OPERATIONAL_MODE_SWITCHER_LINEAR` (3)
- `DEFAULT_OPERATIONAL_MODE` (`OPERATIONAL_MODE_AUTO`)
- `OPERATIONAL_STATE_INITIALIZING` (0)
- `OPERATIONAL_STATE_ENABLED` (1)
- `OPERATIONAL_STATE_FAULT` (2)
- `OPERATIONAL_STATE_HARDWARE_CONFIG` (8)
- `OPERATIONAL_STATE_LINEAR` (0)
- `OPERATIONAL_STATE_SWITCHER` (1)
- `OPERATIONAL_STATE_LINEAR_SWITCHER` (2)
- `OPERATIONAL_STATE_OVER_VOLTAGE_FAULT` (16)
- `OPERATIONAL_STATE_UNDER_VOLTAGE_FAULT` (17)

- OPERATIONAL_STATE_OVER_CURRENT_FAULT (18)
- OPERATIONAL_STATE_OVER_POWER_FAULT (19)
- OPERATIONAL_STATE_REVERSE_POLARITY_FAULT (20)
- OPERATIONAL_STATE_OVER_TEMPERATURE_FAULT (21)
- OPERATIONAL_STATE_OPERATING_MODE (24)
- OPERATIONAL_STATE_CONSTANT_CURRENT (0)
- OPERATIONAL_STATE_CONSTANT_VOLTAGE (1)
- OPERATIONAL_STATE_CONSTANT_POWER (2)
- OPERATIONAL_STATE_CONSTANT_RESISTANCE (3)

[Back to the top](#)

Digital

class `brainstem.stem.Signal (module, index)`

The Signal Class is the 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.

[Back to the top](#)

Store

class `brainstem.stem.Store (module, index)`

Access the store on a BrainStem Module.

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.

Useful Constants:

- INTERNAL_STORE (0)
- RAM_STORE (1)
- SD_STORE (2)
- EEPROM_STORE (3)

[Back to the top](#)

Temperature

class `brainstem.stem.Temperature (module, index)`

Provide interface to temperature sensor.

This entity is only available on certain modules, and provides a temperature reading in micro-celsius.

[Back to the top](#)

Timer

class `brainstem.stem.Timer (module, index)`

Schedules events to occur at future times.

Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

Useful Constants:

- `SINGLE_SHOT_MODE` (0)
- `REPEAT_MODE` (1)
- `DEFAULT_MODE` (`SINGLE_SHOT_MODE`)

[Back to the top](#)

USB

class `brainstem.stem.USB (module, index)`

USBClass 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.

Useful Constants:

- `UPSTREAM_MODE_AUTO` (2)
- `UPSTREAM_MODE_PORT_0` (0)
- `UPSTREAM_MODE_PORT_1` (1)
- `UPSTREAM_MODE_NONE` (255)
- `DEFAULT_UPSTREAM_MODE` (`UPSTREAM_MODE_AUTO`)
- `UPSTREAM_STATE_PORT_0` (0)
- `UPSTREAM_STATE_PORT_1` (1)
- `BOOST_0_PERCENT` (0)
- `BOOST_4_PERCENT` (1)
- `BOOST_8_PERCENT` (2)
- `BOOST_12_PERCENT` (3)
- `PORT_MODE_SDP` (0)
- `PORT_MODE_CDP` (1)

- PORT_MODE_CHARGING (2)
- PORT_MODE_PASSIVE (3)
- PORT_MODE_USB2_A_ENABLE (4)
- PORT_MODE_USB2_B_ENABLE (5)
- PORT_MODE_VBUS_ENABLE (6)
- PORT_MODE_SUPER_SPEED_1_ENABLE (7)
- PORT_MODE_SUPER_SPEED_2_ENABLE (8)
- PORT_MODE_USB2_BOOST_ENABLE (9)
- PORT_MODE_USB3_BOOST_ENABLE (10)
- PORT_MODE_AUTO_CONNECTION_ENABLE (11)
- PORT_MODE_CC1_ENABLE (12)
- PORT_MODE_CC2_ENABLE (13)
- PORT_MODE_SBU_ENABLE (14)
- PORT_MODE_CC_FLIP_ENABLE (15)
- PORT_MODE_SS_FLIP_ENABLE (16)
- PORT_MODE_SBU_FLIP_ENABLE (17)
- PORT_MODE_USB2_FLIP_ENABLE (18)
- PORT_MODE_CC1_INJECT_ENABLE (19)
- PORT_MODE_CC2_INJECT_ENABLE (20)
- PORT_SPEED_NA (0)
- PORT_SPEED_HISPEED (1)
- PORT_SPEED_SUPERSPEED (2)

[Back to the top](#)

RCServo

class `brainstem.stem.RCServo (module, index)`

Provides RCServo functionality on certain modules.

This entity is only available on certain modules. The RCServoClass can be used to interpret and control RC Servo signals and motors via the digital pins.

Useful Constants:

- SERVO_DEFAULT_POSITION (128)
- SERVO_DEFAULT_MIN (64)
- SERVO_DEFAULT_MAX (192)

[Back to the top](#)

USB

class `brainstem.stem.Relay (module, index)`

The RelayClass is the interface to relay entities on BrainStem modules.

Relay entities may be enabled (set) or disabled (cleared low). Other capabilities may be available, please see the product datasheet.

Useful Constants:

- `VALUE_LOW` (0)
- `VALUE_HIGH` (1)

[Back to the top](#)

3.2.11 Equalizer

class `brainstem.entity.Equalizer (module, index)`

Equalizer Class 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 – The equalizer receiver channel.

Returns

Result object, containing `NO_ERROR` and the receiver configuration of the supplied channel or a non zero Error code.

getTransmitterConfig ()

Gets the transmitter configuration

Returns

Result object, containing `NO_ERROR` and the current transmitter config or a non zero Error code.

setReceiverConfig (*channel, config*)

Sets the receiver configuration for a given channel.

Parameters

- **channel** – The equalizer receiver channel.
- **config** – Configuration to be applied to the receiver.

Returns

Result.error Return `NO_ERROR` on success, or one of the common sets of return error codes on failure.

setTransmitterConfig (*config*)

Sets the transmitter configuration

Parameters

config – Configuration to be applied to the transmitter.

Returns

Result.error Return `NO_ERROR` on success, or one of the common sets of return error codes on failure.

3.2.12 I2C

class `brainstem.entity.I2C (module, index)`

The I2C class is the interface the I2C busses on BrainStem modules.

The class provides a way to send read and write commands to I2C devices on the entitie's bus.

Useful Constants:

- `I2C_DEFAULT_SPEED` (0)
- `I2C_SPEED_100Khz` (1)
- `I2C_SPEED_400Khz` (2)
- `I2C_SPEED_1000Khz` (3)

getSpeed ()

Get the current speed setting of the I2C object.

Returns

returns a `Result` object containing one of the constants representing the I2C objects current speed setting.

read (*address*, *length*)

Send I2C read command, on the I2C BUS represented by the entity.

Param:

address (int): The I2C address (7bit <XXXX-XXX0>) of the device to read.

Param:

length (int): The length of the data to read in bytes.

Returns

Result object, containing `NO_ERROR` and read data or a non zero Error code.

Return type

Result

setPullup (*bEnable*)

Set software controlled I2C pullup state.

Sets the software controlled pullup on the bus for stems with software controlled pullup capabilities. Check the device datasheet for more information. This setting is saved by a `system.save`.

Param:

bEnable (bool): The desired state of the pullup.

Returns

Return `NO_ERROR` on success, or one of the common sets of return error codes on failure.

Return type

`Result.error`

setSpeed (*value*)

Set the current speed setting of the I2C object.

Param:

value (int): The constant representing the bus speed setting to apply or this object.

Returns

returns `NO_ERROR` on success or `PARAMETER_ERROR` on failure.

write(*address*, *length*, **args*)

Send I2C write command, on the I2C BUS represented by the entity.

Param:

address (int): The I2C address (7bit <XXXX-XXX0>) of the device to write.

Param:

length (int): The length of the data to write in bytes.

Param:

data (*int | list): variable number of args of either int in the range of 0 to 255 or list|tuple of ints.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.13 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)²²

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 or TCPIP.
- **serial_number** (*int*) – The module serial number.
- **module** – The module address on the Brainstem network.
- **model** – The device model number of the Brainstem module.
- ****keywords** – For TCPIP modules the possibilities are,
 - *ip_address*: (int) The IP address of the module.
 - *tcp_port*: (int) The TCP port of the module.

transport

instance attribute holding transport type.

Type

USB/TCPIP

serial_number

Module serial number.

Type

int

²² <https://acroname.com/reference>

module

Module address.

Type

int

model

Brainstem device type: See Defs module for listing.

Type

int

ip_address

IP address for TCP/IP based modules.

Type

Optional[int]

tcp_port

TCP port for TCP/IP based modules.

Type

Optional[int]

serial_port

Serial Port for SERIAL based modules.

Type

Optional[str]

baudrate

Baudrate for SERIAL based modules.

Type

Optional[int]

SERIAL = 3

SERIAL transport type.

TCPIP = 2

TCPIP transport type.

USB = 1

USB transport type.

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)

3.2.14 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](#)²³ section of the [Acroname BrainStem Reference](#)²⁴

class `brainstem.module.Entity` (*module, command, 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](#)²⁵ section of the brainstem reference for more information.

call_UEI (*option*)

Result.error: Call a set UEI on this entity.

Parameters

option (*int*) – The command option.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

check_UEIBytes (*result, valuesize*)

Helper function for UEIBytes get checks, specifically checking and fixing value sizes

Parameters

- **result** (*Result*) – The Result object from a `get_UEIBytes`
- **valuesize** (*int*) – The base value size of the elements in the set of bytes, 1 = uint8, 2 = uint16, 4 = uint32

Returns

Returns a result object, whose value is set,
or with the requested value when the results error is set to NO_ERROR

Return type

Result

property `command`

Return the entity command.

Type

int

drain_UEI (*option*)

drain UEI packets matchin option.

Parameters

option (*int*) – The command option.

²³ <https://acroname.com/reference/terms.html>

²⁴ <https://acroname.com/reference>

Returns

Returns a result object, whose value is the number of
packets drained, and the error value set to NO_ERROR

Return type

Result

get_UEI (*option*)

Result.error: Get a UEI value.

Parameters

option (*int*) – The command option.

Returns

Returns a result object, whose value is set,
or with the requested value when the results error is set to NO_ERROR

Return type

Result

get_UEIBytes (*option*)

Get a UEI Bytes 8 bit value.

Parameters

option (*int*) – The command option.

Returns

Returns a result object, whose value is set,
or with the requested value when the results error is set to NO_ERROR

Return type

Result

get_UEI_with_param (*option*, *param*)

Result.error: Get a UEI value based on a parameter.

Parameters

- **option** (*int*) – The command option.
- **param** (*byte*) – The command parameter

Returns

Returns a result object, whose value is set,
on with the requested value when the results error is set to NO_ERROR

Return type

Result

property index

Return the entity index

Type

int

property module

Return this entities module.

Type

Module

prep_UEIBytes (*buffer, valuesize*)

Helper function for UEIBytes set which converts base type to single byte tuple

Parameters

- **buffer** (*tuple*) – An array of values to be converted to single bytes
- **valuesize** (*int*) – The base value size of the elements in the set of bytes, 1 = uint8, 2 = uint16, 4 = uint32

Returns

Returns a tuple

Return type

Result

set_UEI16 (*option, value*)

Result.error: Call a set UEI with short param on this entity.

Parameters

- **option** (*int*) – The command option.
- **value** (*short*) – The short parameter to send.

Returns

Returns an error result from the list of defined error codes in brainstem.result

Return type

Result.error

set_UEI32 (*option, value*)

Result.error: Call a set UEI with int param on this entity.

Parameters

- **option** (*int*) – The command option.
- **value** (*int*) – The int parameter to send.

Returns

Returns an error result from the list of defined error codes in brainstem.result

Return type

Result.error

set_UEI32_with_subindex (*option, subindex, value*)

Result.error: Call a set UEI with a subindex.

Parameters

- **option** (*int*) – The command option.
- **subindex** (*byte*) – The subindex of the entity.
- **param** (*byte*) – The byte parameter to send.

Returns

Returns an error result from the list of defined error codes in brainstem.result

Return type

Result.error

set_UEI8 (*option*, *value*)

Result.error: Call a set UEI with byte param on this entity.

Parameters

- **option** (*int*) – The command option.
- **value** (*byte*) – The byte parameter to send.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

set_UEI8_with_subindex (*option*, *subindex*, *value*)

Result.error: Call a set UEI with a subindex.

Parameters

- **option** (*int*) – The command option.
- **subindex** (*byte*) – The subindex of the entity.
- **param** (*byte*) – The byte parameter to send.

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

set_UEIBytes (*option*, *buffer*)

Result.error: Call a set UEI with buffer and length of buffer on this entity.

Parameters

- **option** (*int*) – The command option.
- **buffer** (*byte array*) – The buffer to be sent

Returns

Returns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

class `brainstem.module.Module` (*address*, *enable_auto_networking=True*, *model=0*)

Base class for BrainStem Modules.

Provides default implementations for connecting and disconnecting from BrainStem modules via the module's serial number, a [Spec](https://acroname.com/reference/terms.html) object or through another module.

property `address`

Return the Brainstem module address.

Type

int

property `bAutoNetworking`

Return the current networking mode.

²⁵ <https://acroname.com/reference/terms.html>

Type

bool

connect (*transport*, *serial_number*)

Result.error: Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – The transport to connect over.
- **serial_number** (*int*) – Serial number of the module.

ReturnsReturns an error result from the list of defined error codes in `brainstem.result`**Return type**

Result.error

connectFromSpec (*spec*)

Result.error: Connect to a BrainStem module with a Spec.

Parameters**spec** (*Spec*) – The specifier for the connection.**Returns**Returns an error result from the list of defined error codes in `brainstem.result`**Return type**

Result.error

connectThroughLinkModule (*module*)

Result.error: Connect to network 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**Returns an error result from the list of defined error codes in `brainstem.result`**Return type**

Result.error

disconnect ()

Disconnect from the Brainstem module.

discoverAndConnect (*transport*, *serial_number=None*)

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** (*int*) – The module address to switch to for this module instance.
- **serial_number** (*int*) – The module serial_number to look for.

ReturnsReturns an error result from the list of defined error codes in `brainstem.result`

Return type

Result.error

getStatus ()

Returns the status of the BrainStem connection

See `brainstem.link.Status` for the possible states.**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

returns the model number of the object.

Type

Model

reconnect ()

Reconnect a lost connection to a Brainstem module.

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** (*int*) – The module address to switch to for this module instance.**Returns**Returns an error result from the list of defined error codes in `brainstem.result`**Return type**

Result.error

setNetworkingMode (mode)

Set the networking mode of the module object.

By default the module object is configured 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 communicate with the device. If advanced networking is required the auto networking mode can be turned off.

Parameters**mode** (*bool*) – True or 1 = Auto networking False or 0 = Manual networking**Returns**Returns an error result from the list of defined error codes in `brainstem.result`**Return type**

Result.error

property spec

Return the current spec object.

Type
Spec

3.2.15 Mux

class `brainstem.entity.Mux (module, index)`

Access MUX specialized entities on certain BrainStem modules.

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 (multiplexor) 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).

Useful Constants:

- `UPSTREAM_STATE_ONBOARD` (0)
- `UPSTREAM_STATE_EDGE` (1)
- `UPSTREAM_MODE_AUTO` (0)
- `UPSTREAM_MODE_ONBOARD` (1)
- `UPSTREAM_MODE_EDGE` (2)
- `DEFAULT_MODE` (`UPSTREAM_MODE_AUTO`)

getChannel ()

Gets the current selected channel.

Param:

`channel` (int): The channel of the mux to enable.

Returns

Return `NO_ERROR` on success, or one of the common sets of return error codes on failure.

Return type

`Result.error`

getConfiguration ()

Gets the configuration of the Mux.

Returns

Return result object with `NO_ERROR` set and the current mux voltage setting in the `Result.value` or an `Error`.

Return type

Result

getEnable ()

Gets the enable/disable status of the mux.

Returns

Return `NO_ERROR` on success, or one of the common sets of return error codes on failure.

Return type

`Result.error`

getSplitMode()

Gets the bit packed mux split configuration.

Returns

Return result object with NO_ERROR set and the current mux voltage setting in the Result.value or an Error.

Return type

Result

getVoltage(channel)

Gets the voltage of the specified channel.

On some modules this is a measured value so 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

Return result object with NO_ERROR set and the current mux voltage setting in the Result.value or an Error.

Return type

Result

setChannel(channel)

Enables the specified channel of the mux.

Param:

channel (int): The channel of the mux to enable.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setConfiguration(config)

Sets the configuration of the mux.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setEnabled(bEnable)

Enables or disables the mux based on the param.

Param:

bEnable (bool): True = Enable, False = Disable

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setSplitMode(splitMode)

Sets the mux split configuration

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.16 Pointer

class `brainstem.entity.Pointer (module, index)`

Access 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 and incrementing pointer.

Useful Constants:

- `POINTER_MODE_STATIC` (0)
- `POINTER_MODE_INCREMENT` (1)

getChar ()

Get the value of the pad at the current cursor position.

If the mode is increment this read will increment the cursor by 1 byte.

Returns

Result object, containing NO_ERROR and the value
or a non zero Error code.

Return type

Result

getInt ()

Get the value of the pad at the current cursor position.

If the mode is increment this read will increment the cursor by 4 bytes.

Returns

Result object, containing NO_ERROR and the value
or a non zero Error code.

Return type

Result

getMode ()

Get the pointer offset for this pointer.

Returns

Result object, containing NO_ERROR and the current mode
or a non zero Error code.

Return type

Result

getOffset ()

Get the pointer offset for this pointer.

Returns

Result object, containing NO_ERROR and the current offset
or a non zero Error code.

Return type

Result

getShort ()

Get the value of the pad at the current cursor position.

If the mode is increment this read will increment the cursor by 2 bytes.

Returns

Result object, containing NO_ERROR and the value
or a non zero Error code.

Return type

Result

getTransferStore ()

Get the open slot handle for this pointer.

Returns

Result object, containing NO_ERROR and the handle
or a non zero Error code.

Return type

Result

setChar (charVal)

Set a value at the current cursor position within the pad.

If the mode is increment this write will increment the cursor by 1 byte.

Param:

charVal (char): The value to set into the pad at the current
pointer position.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setInt (intVal)

Set a value at the current cursor position within the pad.

If the mode is increment this write will increment the cursor by 4 bytes.

Param:

short (short): The value to set into the pad at the current
pointer position.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setMode (mode)

Set the pointer offset for this pointer.

Param:

mode (char): The mode. One of POINTER_MODE_STATIC or
POINTER_MODE_INCREMENT

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setOffset (*offset*)

Set the pointer offset for this pointer.

Param:

offset (char): The byte offset within the pad (0 - 255).

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setShort (*shortVal*)

Set a value at the current cursor position within the pad.

If the mode is increment this write will increment the cursor by 2 bytes.

Param:

shortVal (short): The value to set into the pad at the current pointer position.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setTransferStore (*handle*)

Set store slot handle for the pad to store and store to pad transfer.

Param:

handle (char): The handle. Open slot handle id.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

transferToStore (*length*)

Transfer length bytes from the pad cursor position into the open store handle.

If the mode is increment the transfer will increment the cursor by length bytes.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

transferFromStore (*length*)

Transfer length bytes from the open store handle to the cursor position in the pad.

If the mode is increment the transfer will increment the cursor by length bytes.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.17 Port

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 allocated power envelope.

Returns

value (int): Allocated power in milli-watts (mW) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

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

value (int): Available power in milli-watts (mW) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCC1Enabled ()

Gets the current enable value of the CC1 lines. Sub-component (CC1) of `getEnabled`.

Returns

value:

- 1 = CC1 enabled
- 0 = CC1 disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCC2Enabled ()

Gets the current enable value of the CC2 lines. Sub-component (CC2) of `getEnabled`.

Returns

value:

- 1 = CC2 enabled
- 0 = CC2 disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCCEnabled ()

Gets the current enable value of the CC lines. Sub-component (CC) of `getEnabled`.

Returns

value (int):

- 1 = CC enabled
- 0 = CC disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCurrentLimit ()

Gets the current limit of the port.

Returns

value: limit of the port in microAmps (uA) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCurrentLimitMode ()

Gets the current limit mode. The mode determines how the port will react to an over current condition.

Returns

value: An enumerated representation of the active current limit mode. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataEnabled ()

Gets the current enable value of the data lines. Sub-component (Data) of getEnable.

Returns**value:**

- 1 = Data enabled
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataHS1Enabled ()

Gets the current enable value of the High Speed 1 side (HS1) data lines.: Sub-component of getDataHSEnable.

Returns**value:**

- 1 = Data enabled;
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataHS2Enabled ()

Gets the current enable value of the High Speed B side (HSB) data lines.: Sub-component of getDataHSEnable.

Returns**value:**

- 1 = Data enabled
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataHSEnable ()

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnable.

Returns**value:**

- 1 = Data enabled

- 0 = Data disabled
- error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataHSRoutingBehavior()

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Returns

value: An enumerated representation of the routing behavior. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataRole()

Gets the Port Data Role.

Returns

value: The current data role. See product datasheet for details. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataSS1Enabled()

Gets the current enable value of the Super Speed 1 side (SS1) data lines.: Sub-component of getDataSSEnable.

Returns**value:**

- 1 = Data enabled
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataSS2Enabled()

Gets the current enable value of the Super Speed 2 side (SS2) data lines. Sub-component of getDataSSEnable.

Returns**value:**

- 1 = Data enabled;
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataSSEnable()

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnable.

Returns**value:**

- 1 = Data enabled;
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataSSRoutingBehavior()

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Returns

value: An enumerated representation of the routing behavior. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataSpeed()

Gets the speed of the enumerated device.

Returns

value: Bit mapped value representing the devices speed. See product datasheet for details. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getEnabled()

Gets the current enable value of the port.

Returns**value:**

- 1 = Fully enabled port
- 0 = One or more disabled sub-components.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

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

value: Bit mapped value representing the errors of the port See product datasheet for details. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

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

value: An enumerated representation of the boost range. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getMode()

Gets current mode of the port

Returns

value: Bit mapped value representing the ports mode. See product datasheet for details. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getName()

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Returns

value: The current name of the port on success. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerEnabled()

Gets the current enable value of the power lines. Sub-component (Power) of getEnable.

Returns**value:**

- 1 = Power enabled
- 0 = Power disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerLimit()

Gets the power limit of the port.

Returns

value: Active power limit in milli-watts (mW) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerLimitMode()

Gets the power limit mode. The mode determines how the port will react to an over power condition.

Returns

value: An enumerated representation of the power limit mode Available modes are product specific. See the reference documentation. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerMode()

Gets the Port Power Mode: Convenience Function of get/setPortMode

Returns

value: The current power mode. See product datasheet for details. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

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

value: Bit mapped value representing the ports state. See product datasheet for details. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getVbusAccumulatedPower()

Gets the accumulated Vbus Power.

Returns

value: The accumulated power in mWattHours (1 == 1e-3 Wh) currently consumed on VBUS. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getVbusCurrent ()**

Gets the Vbus Current.

Returns

value: The current in microamps (1 == 1e-6A) currently present on VBUS. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getVbusVoltage ()**

Gets the Vbus Voltage.

Returns

value: The voltage in microvolts (1 == 1e-6V) currently present on Vbus. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getVconn1Enabled ()**

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled.

Returns**value:**

- 1 = Vconn1 enabled
- 0 = Vconn1 disabled.

error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getVconn2Enabled ()**

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled.

Returns**value:**

- 1 = Vconn2 enabled
- 0 = Vconn2 disabled.

error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getVconnAccumulatedPower ()**

Gets the accumulated Vconn Power.

Returns

value: The accumulated power in mWattHours (1 == 1e-3 Wh) currently consumed on Vconn. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getVconnCurrent ()**

Gets the Vconn Current.

Returns

value: The current in microamps (1 == 1e-6A) currently present on Vconn. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)

getVconnEnabled()

Gets the current enable value of the Vconn lines. Sub-component (Vconn) of getEnabled.

Returns**value:**

- 1 = Vconn enabled
- 0 = Vconn disabled.

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getVconnVoltage()

Gets the Vconn Voltage.

Returns

value: The voltage in microvolts (1 == 1e-6v) currently present on Vconn. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getVoltageSetpoint()

Gets the current voltage setpoint value of the port.

Returns**value:**

The voltage setpoint in uV

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

resetEntityToFactoryDefaults()

Resets the PortClass Entity to it factory default configuration.

Returns (int):

Non-zero BrainStem error code on failure.

resetVbusAccumulatedPower()

Resets the Vbus accumulated power.

Returns (int):

Non-zero BrainStem error code on failure.

resetVconnAccumulatedPower()

Resets the Vconn accumulated power.

Returns (int):

Non-zero BrainStem error code on failure.

setCC1Enabled(enable)

Enables or disables the CC1 lines. Sub-component (CC1) of setEnabled.

Parameters

enable (*bool*) –

- 1 = Enable CC1 lines
- 0 = Disable CC1 lines.

Returns (int):

Non-zero BrainStem error code on failure.

setCC2Enabled(enable)

Enables or disables the CC2 lines. Sub-component (CC2) of setEnabled.

Parameters

enable (*bool*) –

- 1 = Enable CC2 lines
- 0 = Disable CC2 lines.

Returns (int):

Non-zero BrainStem error code on failure.

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 (int):

Non-zero BrainStem error code on failure.

setCurrentLimit (limit)

Sets the current limit of the port.

Parameters

- limit** (*int*) – limit to be applied in microAmps (uA)

Returns (int):

Non-zero BrainStem error code on failure.

setCurrentLimitMode (mode)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters

- mode** (*int*) – An enumerated representation of the current limit mode.

Returns (int):

Non-zero BrainStem error code on failure.

setDataEnabled (enable)

Enables or disables the data lines. Sub-component (Data) of setEnable.

Parameters

- enable** (*bool*) –
- 1 = Enable data
 - 0 = Disable data

Returns (int):

Non-zero BrainStem error code on failure.

setDataHS1Enabled (enable)

Enables or disables the High Speed 1 side (HS1) data lines. Sub-component of setDataHSEnable.

Parameters

- enable** (*bool*) –
- 1 = Enable data
 - 0 = Disable data

Returns (int):

Non-zero BrainStem error code on failure.

setDataHS2Enabled (enable)

Enables or disables the High Speed 2 side (HS2) data lines. Sub-component of setDataHSEnable.

Parameters

- enable** (*bool*) –

- 1 = Enable data
- 0 = Disable data.

Returns (int):

Non-zero BrainStem error code on failure.

setDataHSEnabled (enable)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnable.

Parameters

- enable** (*bool*) –
- 1 = Enable data
 - 0 = Disable data.

Returns (int):

Non-zero BrainStem error code on failure.

setDataHSRoutingBehavior (mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

- mode** (*int*) – An enumerated representation of the routing behavior.

Returns (int):

Non-zero BrainStem error code on failure.

setDataSS1Enabled (enable)

Enables or disables the Super Speed 1 side (SS1) data lines. Sub-component of setDataEnable.

Parameters

- enable** (*bool*) –
- 1 = Enable data
 - 0 = Disable data.

Returns (int):

Non-zero BrainStem error code on failure.

setDataSS2Enabled (enable)

Enables or disables the Super Speed 2 side (SS2) data lines. Sub-component of setDataEnable.

Parameters

- enable** (*bool*) –
- 1 = Enable data
 - 0 = Disable data.

Returns (int):

Non-zero BrainStem error code on failure.

setDataSSEnabled (enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnable.

Parameters

- enable** (*bool*) –
- 1 = Enable data
 - 0 = Disable data.

Returns (int):

Non-zero BrainStem error code on failure.

setDataSSRoutingBehavior (mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode (*int*) – An enumerated representation of the routing behavior.

Returns (int):

Non-zero BrainStem error code on failure.

setEnabled (enable)

Enables or disables the entire port.

Parameters

enable (*bool*) –

- 1 = Enable the port
- 0 = Disable the port

Returns (int):

Non-zero BrainStem error code on failure.

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 (*int*) – An enumerated representation of the boost range.

Returns (int):

Non-zero BrainStem error code on failure.

setMode (mode)

Sets the mode of the port.

Parameters

mode (*int*) – Port mode to be set. See product datasheet for details.

Returns (int):

Non-zero BrainStem error code on failure.

setName (name)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

name (*string*) – User defined name to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setPowerEnabled (enable)

Enables or Disables the power lines. Sub-component (Power) of setEnable.

Parameters

enable (*bool*) –

- 1 = Enable power
- 0 = Disable disable.

Returns (int):

Non-zero BrainStem error code on failure.

setPowerLimit (limit)

Sets the power limit of the port.

Parameters

limit (*int*) – Limit to be applied in milli-watts (mW)

Returns (int):

Non-zero BrainStem error code on failure.

setPowerLimitMode (*mode*)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

- **mode** (*int*) – An enumerated representation of the power limit mode to be applied.
- **documentation.** (*Available modes are product specific. See the reference*) –

Returns (int):

Non-zero BrainStem error code on failure.

setPowerMode (*powerMode*)

Sets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

powerMode (*int*) – Power mode to be set. See product datasheet for details.

Returns (int):

Non-zero BrainStem error code on failure.

setVconn1Enabled (*enable*)

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

Parameters

- enable** (*bool*) –
- 1 = Enable Vconn1 lines
 - 0 = Disable Vconn1 lines.

Returns (int):

Non-zero BrainStem error code on failure.

setVconn2Enabled (*enable*)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

Parameters

- enable** (*bool*) –
- 1 = Enable Vconn2 lines
 - 0 = Disable Vconn2 lines

Returns (int):

Non-zero BrainStem error code on failure.

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 (int):

Non-zero BrainStem error code on failure.

setVoltageSetpoint (*value*)

Sets the current voltage setpoint value of the port

Parameters

value (*int*) – The voltage setpoint in uV

Returns (int):

Non-zero BrainStem error code on failure.

3.2.18 Power Delivery

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.

getCableCurrentMax ()

Gets the maximum current capability report by the e-mark of the attached cable

Returns

value (int): An enumerated representation of current

- Unknown/Unattached (0)
- 3 Amps (1)
- 5 Amps (2)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCableOrientation ()

Gets the current orientation being used for PD communication

Returns

value (int): An enumerated representation of the cables max speed

- Unconnected = 0
- CC1 (1)
- CC2 (2)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCableSpeedMax ()

Gets the maximum data rate capability reported by the e-mark of the attached cable.

Returns

value (int): An enumerated representation of the cables max 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)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCableType ()

Gets the cable type reported by the e-mark of the attached cable.

Returns

value (int): An enumerated representation of the cables max speed

- Invalid, no e-mark and not Vconn powered (0)
- Passive cable with e-mark (1)

- Active cable (2)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getCableVoltageMax ()

Gets the maximum voltage capability reported by the e-mark of the attached cable.

Returns

value (int): An enumerated representation of voltage

- Unknown/Unattached (0)
- 20 Volts DC (1)
- 30 Volts DC (2)
- 40 Volts DC (3)
- 50 Volts DC (4)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getConnectionState ()

Gets the current state of the connection in the form of an enumeration.

Returns

value (int): An enumerated representation of the current state. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

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

value (int): An enumerated value referring to current swap value

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

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 (*int*) – Flag/Advertisement to be modified

Returns

value (int): The current mode of the provided flag

- Disabled (0)
- Enable (1)
- Auto (2) default

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getNumberOfPowerDataObjects (*partner, powerRole*)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

Parameters

- **partner** (*int*) –
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** (*int*) –
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

Returns

value (*int*): The number of of Power Data Objects (PDO) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getOverride ()

Gets the current enabled overrides

Returns

value (*int*): Bit mapped representation of the current override configuration. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

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

value (*int*): An enumerated value referring to the current configuration.

- Allowable values are 0 - 4
- error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerDataObject (*partner, powerRole, ruleIndex*)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

Parameters

- **partner** (*int*) –
 - Local = 0 = powerdeliveryPartnerLocal
 - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** (*int*) –
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*int*) –

Returns

value (*int*): Power Data Object (PDO) for the given partner and power role. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerDataObjectEnabled (*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).

Parameters

- **powerRole** (*int*) –
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*int*) – The index of the PDO in question. Valid index are 1-7.

Returns

value (bool): Represents the enabled state. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerDataObjectEnabledList (*powerRole*)

Gets all Power Data Object enables for a given power role. Equivalent of calling PowerDelivery.getPowerDataObjectEnabled() for all indexes.

Parameters

- **powerRole** (*int*) –
 - Source = 1 = powerdeliveryPowerRoleSource (1) Source
 - Sink = 2 = powerdeliveryPowerRoleSink (2) Sink

Returns

value (bool): Bit 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). error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerDataObjectList ()

Gets all Power Data Objects (PDO) for a given partner and power role. Equivalent to calling PowerDelivery.getPowerDataObject() on all partners, power roles, and index's.

Returns

value (tuple(int)): All Power Data Objects (PDOs) On success the length should be 28 (7 rules * 2 partners * 2 power roles) The order of which is:

- Rules 1-7 Local Source
- Rules 1-7 Local Sink
- Rules 1-7 Remote Source
- Rules 1-7 Remote Sink

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerRole ()

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

Returns

value (int): The current power role

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerRolePreferred ()

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

Returns

value (int): The current power role

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getRequestDataObject (partner)

Gets the current Request Data Object (RDO) for a given partner.

RDOs:

- Are provided by the sinking device.
- 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 (*int*) -

- Local = 0 = powerdeliveryPartnerLocal
- Remote = 1 = powerdeliveryPartnerRemote

Returns

value (int): Value of the request RDO. (Zero indicates the RDO is not active)

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

request (request)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

Parameters

request (*int*) -

- pdRequestHardReset (0)
- pdRequestSoftReset (1)
- pdRequestDataReset (2)
- pdRequestPowerRoleSwap (3)
- pdRequestPowerFastRoleSwap (4)
- pdRequestDataRoleSwap (5)
- pdRequestVconnSwap (6)
- pdRequestSinkGoToMinimum (7)
- pdRequestRemoteSourcePowerDataObjects (8)
- pdRequestRemoteSinkPowerDataObjects (9)

Returns (int):

Non-zero BrainStem error code on failure.

requestStatus ()

Gets the status of the last request command sent.

Returns

value (int): the request status error: Non-zero BrainStem error code on failure.

Return type

Result (object)

resetEntityToFactoryDefaults ()

Resets the PowerDelivery Entity to it factory default configuration.

Returns (int):

Non-zero BrainStem error code on failure.

resetPowerDataObjectToDefault (powerRole, ruleIndex)

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

Parameters

- **powerRole** (*int*) –
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*int*) – The index of the PDO in question. Valid index are 1-7.

Returns (int):

Non-zero BrainStem error code on failure.

setFastRoleSwapCurrent (*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

- **current** (*int*) – An enumerated value referring to value to be set. - 0A (0) - 900mA (1) - 1.5A (2) - 3A (3)

Returns (int):

Non-zero BrainStem error code on failure.

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** (*int*) – Flag/Advertisement to be modified
- **mode** (*int*) –
 - Disabled (0)
 - Enable (1)
 - Auto (2) default

Returns (int):

Non-zero BrainStem error code on failure.

setOverride (*overrides*)

Sets the current overrides.

Parameters

- **overrides** (*int*) – Overrides to be set in a bit mapped representation.

Returns (int):

Non-zero BrainStem error code on failure.

setPeakCurrentConfiguration (*config*)

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

- **config** (*int*) – An enumerated value referring to the configuration to be set - Allowable values are 0 - 4

Returns (int):

Non-zero BrainStem error code on failure.

setPowerDataObject (*powerRole, ruleIndex, pdo*)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

Parameters

- **powerRole** (*int*) –
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*int*) – The index of the PDO in question. Valid index are 1-7.
- **pdo** (*int*) – Power Data Object to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setPowerDataObjectEnabled (*powerRole, ruleIndex, enable*)

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** (*int*) –
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*int*) – The index of the PDO in question. Valid index are 1-7.
- **enable** (*bool*) – The enabled state to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setPowerRole (*powerRole*)

Sets the power role to be advertised by the Local partner. (CC Strapping).

Parameters

- **powerRole** (*int*) –
 - Disabled = 0 = powerdeliveryPowerRoleDisabled
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink
 - Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Returns (int):

Non-zero BrainStem error code on failure.

setPowerRolePreferred (*powerRole*)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

Parameters

- **powerRole** (*int*) –
 - Disabled = 0 = powerdeliveryPowerRoleDisabled
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

Returns (int):

Non-zero BrainStem error code on failure.

setRequestDataObject (*partner, 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.
- 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** (*int*) - Indicates which side of the PD connection is in question.
Local = 0 = powerdeliveryPartnerLocal
- **rdo** (*int*) - RDO to be applied.

Returns (int):

Non-zero BrainStem error code on failure.

3.2.19 Rail

class `brainstem.entity.Rail` (*module, index*)

Provides power rail functionality on certain modules.

This entity is only available on certain modules. The RailClass can be used to control power to downstream devices, I has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

Useful Constants:

- KELVIN_SENSING_OFF (0)
- KELVIN_SENSING_ON (1)
- OPERATIONAL_MODE_AUTO (0)
- OPERATIONAL_MODE_LINEAR (1)
- OPERATIONAL_MODE_SWITCHER (2)
- OPERATIONAL_MODE_SWITCHER_LINEAR (3)
- DEFAULT_OPERATIONAL_MODE (OPERATIONAL_MODE_AUTO)
- OPERATIONAL_STATE_INITIALIZING (0)
- OPERATIONAL_STATE_ENABLED (1)
- OPERATIONAL_STATE_FAULT (2)
- OPERATIONAL_STATE_HARDWARE_CONFIG (8)
- OPERATIONAL_STATE_LINEAR (0)
- OPERATIONAL_STATE_SWITCHER (1)
- OPERATIONAL_STATE_LINEAR_SWITCHER (2)
- OPERATIONAL_STATE_OVER_VOLTAGE_FAULT (16)
- OPERATIONAL_STATE_UNDER_VOLTAGE_FAULT (17)
- OPERATIONAL_STATE_OVER_CURRENT_FAULT (18)
- OPERATIONAL_STATE_OVER_POWER_FAULT (19)
- OPERATIONAL_STATE_REVERSE_POLARITY_FAULT (20)
- OPERATIONAL_STATE_OVER_TEMPERATURE_FAULT (21)
- OPERATIONAL_STATE_OPERATING_MODE (24)
- OPERATIONAL_STATE_CONSTANT_CURRENT (0)
- OPERATIONAL_STATE_CONSTANT_VOLTAGE (1)
- OPERATIONAL_STATE_CONSTANT_POWER (2)
- OPERATIONAL_STATE_CONSTANT_RESISTANCE (3)

clearFaults()

Clears the current fault state of the rail.

Refer to the module datasheet for definition of the rail faults.

Returns

Return result object with NO_ERROR set or an Error.

Return type

Result

getCurrent()

Get the rail current.

Returns

Result object, containing NO_ERROR and the current in microamps
or a non zero Error code.

Return type

Result

getCurrentLimit()

Get 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

Return result object with NO_ERROR set and the current
limit setting in the Result.value or an Error condition.

Return type

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

Return result object with NO_ERROR set and the current rail current setting in the Result.value or an Error.

Return type

Result

getEnable()

Get the state of the rail switch.

Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Returns

Return result object with NO_ERROR set and the current rail enable state in the Result.value or an Error condition.

Return type

Result

getKelvinSensingEnable()

Determine whether kelvin sensing is enabled or disabled.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

bEnable (*bool*) – Kelvin sensing is enabled or disabled.

Returns

Return result object with NO_ERROR set and the current rail kelvin sensing mode setting in the Result.value or an Error.

Return type

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

Return result object with NO_ERROR set and the current rail kelvin sensing state setting in the Result.value or an Error.

Return type

Result

getOperationalMode()

Determine the current operational mode of the system.

Refer to the module datasheet for definition of the rail operational mode capabilities.

Returns

Return result object with NO_ERROR set and the current rail operational mode setting in the Result.value or an Error.

Return type

Result

getOperationalState()

Determine the current operational state of the system.

Refer to the module datasheet for definition of the rail operational states.

Returns

Return result object with NO_ERROR set and the current rail operational state in the Result.value or an Error.

Return type

Result

getPower()

Get the rail power.

Returns

Result object, containing NO_ERROR and the power in milliwatts
or a non zero Error code.

Return type

Result

getPowerLimit()

Get the rail power limit setting.

Check product datasheet to see if this feature is available.

Parameters

milliwatts (*int*) – The power in milli-watts (1 == 1e-3W).

Returns

Return result object with NO_ERROR set and the power
limit setting in the Result.value or an Error condition.

Return type

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

Return result object with NO_ERROR set and the power rail power setting in the Result.value or an Error.

Return type

Result

getResistance()

Get the rail resistance.

Returns

Result object, containing NO_ERROR and the resistance in milliohms
or a non zero Error code.

Return type

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

Return result object with NO_ERROR set and the resistance rail resistance setting in the Result.value or an Error.

Return type

Result

getTemperature()

Get the rail temperature.

Returns

Return result object with NO_ERROR set and the rail temperature in the Result.value or an Error condition.

Return type

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.

On some modules this is a measured value so 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

Return result object with NO_ERROR set and the current rail voltage setting in the Result.value or an Error.

Return type

Result

getVoltageMaxLimit()

Get the rail voltage maximum limit setting.

Check product datasheet to see if this feature is available.

Parameters

microvolts (*int*) – The voltage maximum in micro-volts (1 == 1e-6V).

Returns

Return result object with NO_ERROR set and the voltage minimum
limit setting in the Result.value or an Error condition.

Return type

Result

getVoltageMinLimit ()

Get the rail voltage minimum limit setting.

Check product datasheet to see if this feature is available.

Parameters

microvolts (*int*) – The voltage minimum in micro-volts (1 == 1e-6V).

Returns

Return result object with NO_ERROR set and the voltage minimum
limit setting in the Result.value or an Error condition.

Return type

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

Return result object with NO_ERROR set and the current rail voltage setting in the Result.value or an Error.

Return type

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

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

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

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

setEnabled (bEnable)

Set the state of the rail switch.

Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Parameters

bEnable (*bool*) – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

Returns

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

setKelvinSensingEnable (*bEnable*)

Enable or Disable kelvin sensing on the module.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

bEnable (*bool*) – enable or disable kelvin sensing.

Returns

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

setOperationalMode (*mode*)

Set the operational mode of the rail.

Refer to the module datasheet for definition of the rail operational capabilities.

Parameters

mode (*int*) – The operational mode to employ.

Returns

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

setPowerLimit (*milliwatts*)

Set the rail power limit setting.

Check product datasheet to see if this feature is available.

Parameters

milliwatts (*int*) – The power in milli-watts (1 == 1e-3W).

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setPowerSetpoint (*milliwatts*)

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

milliwatts (*int*) – The power in milli-watts (1 == 1e-3W) to be supply by the rail.

Returns

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

setResistanceSetpoint (*milliohms*)

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

milliohms (*int*) – The resistance in milli-ohms (1 == 1e-3Ohms) to be applied to the rail.

Returns

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

setVoltageMaxLimit (*microvolts*)

Set the rail voltage maximum limit setting.

Check product datasheet to see if this feature is available.

Parameters

microvolts (*int*) – The voltage maximum in micro-volts (1 == 1e-6V).

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setVoltageMinLimit (*microvolts*)

Set the rail voltage minimum limit setting.

Check product datasheet to see if this feature is available.

Parameters

microvolts (*int*) – The voltage minimum in micro-volts (1 == 1e-6V).

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

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 supply by the rail.

Returns

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

3.2.20 RCServo

class `brainstem.entity.RCServo (module, index)`

Provides RCServo functionality on certain modules.

This entity is only available on certain modules. The RCServoClass can be used to interpret and control RC Servo signals and motors via the digital pins.

Useful Constants:

- `SERVO_DEFAULT_POSITION` (128)
- `SERVO_DEFAULT_MIN` (64)
- `SERVO_DEFAULT_MAX` (192)

getEnable ()

Get the enable status of the servo channel.

If the enable status is 0 the servo channel is disabled, if the status is 1 the channel is enabled.

Returns

Result object, containing NO_ERROR and servo enable status
or a non zero Error code.

Return type

Result

getPosition ()

Get the position of the servo channel.

Default configuration: 1ms = 64 and 2ms = 192

Returns

Result object, containing NO_ERROR and the servo position
or a non zero Error code.

Return type

Result

getReverse ()

Get the reverse status of the channel.

0 = not reversed, 1 = reversed

Returns

Result object, containing NO_ERROR and the reverse status
or a non zero Error code.

Return type

Result

setEnabled (enable)

Enable the servo channel.

Param:

enable (bool): The state to be set. 0 is disabled, 1 is enabled

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setPosition (*position*)

Set the position of the servo channel.

Param:

position (int): The position to be set. With the default configuration 64 = a 1 ms pulse and 192 = a 2ms pulse.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setReverse (*reverse*)

Se the output to be reverse on the servo channel.

Param:

reverse (bool): Reverse mode: 0 = not reversed, 1 = reversed. ie: setPosition of 64 would actually apply 192 tot he servo output; however, getPosition will return 64.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.21 Relay

class `brainstem.entity.Relay` (*module, index*)

The RelayClass is the interface to relay entities on BrainStem modules.

Relay entities may be enabled (set) or disabled (cleared low). Other capabilities may be available, please see the product datasheet.

Useful Constants:

- VALUE_LOW (0)
- VALUE_HIGH (1)

getEnable ()

Get the relay enable state.

A return of 1 indicates the relay is enabled. A return of 0 indicates the relay is disabled.

Returns

Result object, containing NO_ERROR and digital state
or a non zero Error code.

Return type

Result

getVoltage ()

Get the scaled micro volt value with refrence to ground.

Get a 32 bit signed integer (in micro Volts) based on the boards ground and refrence 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.

Returns

Result object, containing NO_ERROR and microVolts value
or a non zero Error code.

Return type

Result

setEnabled (*bEnable*)

Enables or disables the relay based on bEnable.

Param:

bEnable (int): Set 1 for enable, set 0 for disable.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.22 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](#)²⁶

class `brainstem.result.Result` (*error, value*)

Result class for returning results of commands

Instances of Result represent the response to a command. The Result class also contains constants representing the possible errors that may be encountered during interaction with a BrainStem module.

property error

Return the error attribute

property value

Return the value attribute

3.2.23 Signal

See the [Signal Entity](#) for generic information.

class `brainstem.entity.Signal` (*module, index*)

The Signal Class is the 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.

setEnabled ()

Get the Enable/Disable of the signal.

Returns

Result object, containing NO_ERROR and boolean value True for enabled
False for disabled or a non zero Error code.

²⁶ <https://acroname.com/reference>

getInvert ()

Get the invert status of the signal.

Returns

Result object, containing NO_ERROR and boolean value True for inverted False for normal or a non zero Error code.

getT2Time ()

Get the current wave active period (T2) in nanoseconds.

Returns

Result object, containing NO_ERROR and an integer value not larger than the max unsigned 32bit value. or a non zero Error code.

getT3Time ()

Get the current wave period (T3) in nanoseconds.

Returns

Result object, containing NO_ERROR and an integer value not larger than the max unsigned 32bit value. or a non zero Error code.

setEnabled (enable)

Enable/Disable the signal output.

Parameters

enable – True to enable, false to disable

Returns

Result.error Return NO_ERROR on success, or one of the common sets of return error codes on failure.

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 – True to invert, false for normal mode.

Returns

Result.error Return NO_ERROR on success, or one of the common sets of return error codes on failure.

setT2Time (t2_nsec)

Set the signal active period or T2 in nanoseconds.

Parameters

t2_nsec – Tnteger not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Returns

Result.error Return NO_ERROR on success, or one of the common sets of return error codes on failure.

setT3Time (t3_nsec)

Set the signal period or T3 in nanoseconds.

Parameters

t3_nsec – Tnteger not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Returns

Result.error Return NO_ERROR on success, or one of the common sets of return error codes on failure.

3.2.24 System

class `brainstem.entity.System (module, index)`

Access system controls configuration and information.

The system entity is available on all BrainStem modules, and provides access to system information such as module, router and serial number, as well as control over the user LED, and information such as the system input voltage.

Useful Constants:

- `BOOT_SLOT_DISABLE` (255)

getBootSlot ()

Get the store slot which is mapped when the module boots.

Returns

Result object, containing NO_ERROR and slot number
or a non zero Error code.

Return type

Result

getErrors ()

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

Returns

value: Bit mapped value representing the errors. See product datasheet for details. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getHBInterval ()

Get 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.

Returns

Result object, containing NO_ERROR and the Heartbeat interval
or a non zero Error code.

Return type

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

Result object, containing NO_ERROR and hardware revision
or a non zero Error code.

Return type

Result

getInputCurrent ()

Get the module's input current.

Returns

Result object, containing NO_ERROR and input current
or a non zero Error code.

Return type*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

value (int): an enumerated value representing behavior. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getInputPowerBehaviorConfig()****Gets the input power behavior configuration**

Certain behaviors use a list of ports to determine priority when budgeting power.

Returns

value (tuple(int)): A list of ports which indicate priority sequencing. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getInputPowerSource()**

Provides the source of the current power source in use.

Returns

value (int): An enumerated value representing the current input power source. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getInputVoltage()**

Get the module's input voltage.

Returns

Result object, containing NO_ERROR and input voltage
or a non zero Error code.

Return type*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

Result object, containing NO_ERROR and the LED State
or a non zero Error code.

Return type*Result***getLinkInterface()****Gets the link interface configuration.**

This refers to which interface is being used for control by the device.

Returns

value (int): an enumerated value representing interface.

- 0 = Auto

- 1 = Control Port
- 2 = Hub Upstream Port

error: Non-zero BrainStem error code on failure.

Return type

Result (object)

`getMaximumTemperature ()`

Get the module's maximum temperature ever recorded in micro-C (uC) his value will persists through a power cycle.

Returns

Result object, containing NO_ERROR and max temperature in micro-C
or a non zero Error code.

Return type

Result

`getMinimumTemperature ()`

Get the module's minimum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

Returns

Result object, containing NO_ERROR and max temperature in micro-C
or a non zero Error code.

Return type

Result

`getModel ()`

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "Brain-Stem model codes". Other codes are be used by Acroname for proprietary module types.

Returns

Result object, containing NO_ERROR and model number
or a non zero Error code.

Return type

Result

`getModule ()`

Get the address the module uses on the BrainStem network.

Returns

Result object, containing NO_ERROR and the current module address
or a non zero Error code.

Return type

Result

`getModuleBaseAddress ()`

Get the base address the module.

The software and hardware addresses are added to the base address to produce the effective module address.

Returns

Result object, containing NO_ERROR and the current module address
or a non zero Error code.

Return type

Result

`getModuleHardwareOffset ()`

Get the module address hardware 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

Result object, containing NO_ERROR and module offset
or a non zero Error code.

Return type

Result

getModuleSoftwareOffset ()

Get the module address software offset.

The address offset that is added to the module base address, and potentially the hardware offset to produce the module effective address.

Returns

Result object, containing NO_ERROR and the current module address
or a non zero Error code.

Return type

Result

getName ()

Gets a user defined name of the port.

Helpful for identifying ports/devices in a static environment.

Returns

value: The current name of the port on success. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerLimit ()

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

Returns

value (int): Power limit in milli-watts (mW) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerLimitMax ()

Gets the user defined maximum power limit for the system.

Provides mechanism for defining an unregulated power supplies capability.

Returns

value (int): Power limit in milli-watts (mW) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getPowerLimitState ()

Gets a bit mapped representation of the factors contributing to the power limit.

Active limit can be found through PowerDeliveryClass::getPowerLimit().

Returns

value (int): The current power limit state. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getRouter ()**

Get the router address the module uses to communicate with the host.

Returns

Result object, containing NO_ERROR and the current router address
or a non zero Error code.

Return type*Result***getRouterAddressSetting ()**

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.

Returns

Result object, containing NO_ERROR and the current router address
or a non zero Error code.

Return type*Result***getSerialNumber ()**

Get the module's serial number.

The serial number is a unique 32bit integer which is usually communicated in hexadecimal format.

Returns

Result object, containing NO_ERROR and serial number
or a non zero Error code.

Return type*Result***getTemperature ()**

Get the module's current temperature in micro-C

Returns

Result object, containing NO_ERROR and max temperature in micro-C
or a non zero Error code.

Return type*Result***getUnregulatedCurrent ()**

Gets the current present at the unregulated port.

Returns

value (int): Unregulated current in micro-amps (mA) error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getUnregulatedVoltage ()**

Gets the voltage present at the unregulated port.

Returns

value (int): Unregulated Voltage in micro-volts (mV) error: Non-zero BrainStem error code on failure.

Return type*Result* (object)

getUptime()

Get accumulated system uptime.

This is the total time the system has been powered up with the firmware running. The returned uptime is a count of minutes of uptime, or may be a module dependent counter.

Returns

Result object, containing NO_ERROR and uptime in minutes
or a non zero Error code.

Return type

Result

getVersion()

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.

Returns

Result object, containing NO_ERROR packed version number
or a non zero Error code.

Return type

Result

logEvents()

Save system log entries to slot defined by module.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

reset()

Reset the system.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

resetDeviceToFactoryDefaults()

Resets the device to it factory default configuration.

Returns (int):

Non-zero BrainStem error code on failure.

resetEntityToFactoryDefaults()

Resets the SystemClass Entity to it factory default configuration.

Returns (int):

Non-zero BrainStem error code on failure.

routeToMe(value)

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

value (*int*) – Enable or disable of the route to me function 1 = enable.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

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

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setBootSlot (value)

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

value (*int*) – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setHBInterval (value)

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. Valid values are 1-255; default is 10 (256 milliseconds).

Parameters

value (*int*) – Heartbeat interval settings.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

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 (*int*) – An enumerated representation of behavior to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setInputPowerBehaviorConfig (*config*)

Sets the input power behavior configuration

Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

config (*tuple* (*int*)) – List of ports which indicate priority sequencing.

Returns (int):

Non-zero BrainStem error code on failure.

setLED (*value*)

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

value (*int*) – LED State setting.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setLinkInterface (*linkInterface*)

Sets the link interface configuration.

This refers to which interface is being used for control by the device.

Parameters

interface (*int*) – An enumerated representation of interface to be set. * 0 = Auto= systemLinkAuto * 1 = Control Port = systemLinkUSBControl * 2 = Hub Upstream Port = systemLinkUSBHub

Returns (int):

Non-zero BrainStem error code on failure.

setModuleSoftwareOffset (*value*)

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.

Warning:

changing the module address may cause the module to “drop off” the BrainStem network if the module is also the router. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

value (*int*) – The module address offset.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setName (*name*)

Sets a user defined name of the port.

Helpful for identifying ports/devices in a static environment.

Parameters

name (*string*) – User defined name to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setPowerLimitMax (*limit*)

Sets a user defined maximum power limit for the system.

Provides mechanism for defining an unregulated power supplies capability.

Parameters

limit (*int*) – Limit in milli-watts (mW) to be set

Returns (int):

Non-zero BrainStem error code on failure.

setRouter (*value*)

Set the router address the module uses to communicate with the host.

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

value (*int*) – The module address of the router module on the network.

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.25 Store

class `brainstem.entity.Store` (*module, index*)

Access the store on a BrainStem Module.

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.

Useful Constants:

- INTERNAL_STORE (0)
- RAM_STORE (1)
- SD_STORE (2)
- EEPROM_STORE (3)

getSlotCapacity (*slot*)

Get the slot capacity.

Returns the Capacity of the slot, i.e. The number of bytes it can hold.

return: Result:

Either the capacity of the slot in Result.value or an error.

getSlotLocked (*slot*)**Gets the current lock state of the slot**

Allows for write protection on a slot.

Parameters

slot (*int*) – The slot number

Returns

value: The current locked state of the provided slot. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getSlotSize (*slot*)

Get the slot size.

Returns the size of the data currently filling the slot in bytes.

return: Result:

Either the size of the slot in Result.value or an error.

getSlotState (*slot*)

Get slot state.

Slots which contain reflexes may be “enabled,” i.e. the reflexes contained in the slot are active.

Parameters

slot (*int*) – The slot number.

return: Result:

Return result object with NO_ERROR set and the current state of the slot in the Result.value or an Error.

loadSlot (*slot*, *data*, *_*=None)

Load the slot.

Parameters

- **slot** (*int*) – The slot number.
- **data** (*str*, *bytes*) – The data.
- **_** (*int*) – (length Deprecated) Unused parameter, and will be removed in next minor release.

return: Result.error:

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

setSlotLocked (*slot*, *lock*)**Sets the locked state of the slot**

Allows for write protection on a slot.

Parameters

- **slot** (*int*) – The slot number
- **lock** (*bool*) – Locked state to set.

Returns (int):

Non-zero BrainStem error code on failure.

slotDisable (*slot*)

Disable the slot

slotEnable (*slot*)

Enable the slot

unloadSlot (*slot*)

Unload the slot data.

Parameters

slot (*int*) – The slot number.

return: Result:

Either Returns Result object with NO_ERROR set and its value attribute set with an object of type bytes containing the unloaded data. Or a Result object with a non-zero error.

3.2.26 Temperature

class `brainstem.entity.Temperature` (*module, index*)

Provide interface to temperature sensor.

This entity is only available on certain modules, and provides a temperature reading in micro-celsius.

getValue ()

Get the current temperature in micro-C.

Returns

value: The temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getValueMax ()

Get the module's maximum temperature in micro-C since the last power cycle.

Returns

value: The maximum temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getValueMin ()

Get the module's minimum temperature in micro-C since the last power cycle.

Returns

value: The minimum temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

Return type

Result (object)

reset ()

Get the module's maximum temperature in micro-C since the last power cycle.

Returns

value: The maximum temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**resetEntityToFactoryDefaults()**

Resets the TemperatureClass Entity to it factory default configuration.

Returns (int):

Non-zero BrainStem error code on failure.

3.2.27 Timer

class `brainstem.entity.Timer (module, index)`

Schedules events to occur at future times.

Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

Useful Constants:

- `SINGLE_SHOT_MODE` (0)
- `REPEAT_MODE` (1)
- `DEFAULT_MODE` (`SINGLE_SHOT_MODE`)

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

Return result object with `NO_ERROR` set and the timer expiration in `uSeconds` in the `Result.value` or an `Error`.

Return type*Result***getMode()**

Get the mode of the timer.

Valid timer modes are single mode and repeat mode.

Returns

Return result object with `NO_ERROR` set and the timer mode either single (0) or repeat (1) in the `Result.value` or an `Error`.

Return type*Result***setExpiration (uSecDuration)**

Set the expiration time for the timer entity.

When the timer expires, it will fire the associated `timer[index]()` reflex.

Parameters

`uSecDuration (int)` – The duration before timer expiration in microseconds.

Returns

Return `NO_ERROR` on success, or one of the common
sets of return error codes on failure.

Return type`Result.error`

setMode (*mode*)

Set the mode of the timer.

Parameters

mode (*int*) - The mode of the timer. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Returns

Return NO_ERROR on success, or one of the common
sets of return error codes on failure.

Return type

Result.error

3.2.28 UART

class brainstem.entity.**UART** (*module, index*)

Provides UART entity access on certain BrainStem modules.

A UART is a “Universal Asynchronous Reciever/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

getBaudRate ()

Get the Baud rate of the UART.

Returns

Result object, containing NO_ERROR and the UART baud rate
or a non zero Error code.

Return type

Result

getEnable ()

Get the enable status of the UART.

Returns

Result object, containing NO_ERROR and the UART state
or a non zero Error code.

Return type

Result

getProtocol ()

Get the protocol format of the UART.

Returns

Result object, containing NO_ERROR and the UART protocol
or a non zero Error code.

Return type

Result

setBaudRate (*rate*)

Set the Baud rate of the UART.

Param:

rate (int):

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setEnabled (*bEnable*)

Enable the UART.

Param:

bEnable (bool): True = Enable, False = Disable

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

setProtocol (*protocol*)

Set the protocol format of the UART.

Param:

protocol (int):

Returns

Return NO_ERROR on success, or one of the common sets of return error codes on failure.

Return type

Result.error

3.2.29 USB

class `brainstem.entity.USB` (*module, index*)

USBClass 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.

Useful Constants:

- UPSTREAM_MODE_AUTO (2)
- UPSTREAM_MODE_PORT_0 (0)
- UPSTREAM_MODE_PORT_1 (1)
- UPSTREAM_MODE_NONE (255)
- DEFAULT_UPSTREAM_MODE (UPSTREAM_MODE_AUTO)
- UPSTREAM_STATE_PORT_0 (0)
- UPSTREAM_STATE_PORT_1 (1)
- BOOST_0_PERCENT (0)
- BOOST_4_PERCENT (1)
- BOOST_8_PERCENT (2)
- BOOST_12_PERCENT (3)
- PORT_MODE_SDP (0)
- PORT_MODE_CDP (1)
- PORT_MODE_CHARGING (2)
- PORT_MODE_PASSIVE (3)
- PORT_MODE_USB2_A_ENABLE (4)

- PORT_MODE_USB2_B_ENABLE (5)
- PORT_MODE_VBUS_ENABLE (6)
- PORT_MODE_SUPER_SPEED_1_ENABLE (7)
- PORT_MODE_SUPER_SPEED_2_ENABLE (8)
- PORT_MODE_USB2_BOOST_ENABLE (9)
- PORT_MODE_USB3_BOOST_ENABLE (10)
- PORT_MODE_AUTO_CONNECTION_ENABLE (11)
- PORT_MODE_CC1_ENABLE (12)
- PORT_MODE_CC2_ENABLE (13)
- PORT_MODE_SBU_ENABLE (14)
- PORT_MODE_CC_FLIP_ENABLE (15)
- PORT_MODE_SS_FLIP_ENABLE (16)
- PORT_MODE_SBU_FLIP_ENABLE (17)
- PORT_MODE_USB2_FLIP_ENABLE (18)
- PORT_MODE_CC1_INJECT_ENABLE (19)
- PORT_MODE_CC2_INJECT_ENABLE (20)
- PORT_SPEED_NA (0)
- PORT_SPEED_HISPEED (1)
- PORT_SPEED_SUPERSPEED (2)

clearPortErrorStatus (*channel*)

Clear the error status for the given channel.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getAltModeConfig (*channel*)

Gets alt mode configuration for defined USB channel.

See the product datasheet for device specific details.

Parameters

channel (*int*) – The USB channel

Returns: Result object

getCC1Current (*channel*)

Get the current through the CC1 line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getCC1Voltage (*channel*)

Get the voltage on the CC1 line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getCC2Current (*channel*)

Get the current through the CC2 line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getCC2Voltage (*channel*)

Get the voltage on the CC2 line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getCableFlip (*channel*)

Get the status of cable orientation flip within the S85 switch.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getConnectMode (*channel*)

Get The connection mode for the Switch.

Parameters

channel (*int*) – Upstream port to be applied.

Returns

value (int): The connect mode error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDownstreamBoostMode ()

Get the downstream boost mode.

Returns: Result object

Result value 0 - no boost, 1 - 4%% boost, 2 - 8%% boost, 3 - 12%% boost.

getDownstreamDataSpeed (*channel*)

Get the downstream port data speed.

Returns: Result object**Result value:**

- N/A: PORT_SPEED_NA = 0
- Hi Speed: PORT_SPEED_HISPEED = 1
- SuperSpeed: PORT_SPEED_SUPERSPEED = 2

getEnumerationDelay ()

Get the interport enumeration delay in milliseconds.

Returns: Result object

getHubMode ()**Get a bit mapped representation of the hub mode.**

Usually represents the port data and power lines enable/disable state in one bit packed result. See the product datasheet for state mapping.

Returns: Result object

getPortCurrent (*channel*)

Get the current through the power line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getPortCurrentLimit (*channel*)

Get the current limit for the port.

Returns: Result object

getPortError (*channel*)

Get the error for the Port.

Returns: Result object

getPortMode (*channel*)

Get 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 at

ef usbPortMode

Returns: Result object

getPortState (*channel*)

Get the state for the Port.

Returns: Result object

getPortVoltage (*channel*)

Get the voltage on the power line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getSBU1Voltage (*channel*)

Get the voltage on the SBU1 line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getSBU2Voltage (*channel*)

Get the voltage on the SBU2 line for a port.

Parameters

channel (*int*) – The USB port number

Returns: Result object

getUpstreamBoostMode ()

Get the upstream boost mode.

Returns: Result object

Result value 0 - no boost, 1 - 4%% boost, 2 - 8%% boost, 3 - 12%% boost.

getUpstreamMode ()

Get the upstream switch mode for the USB upstream ports.

Returns: Result object

getUpstreamState ()

Get the upstream switch state for the USB upstream ports.

Returns: Result object

Result value 2 if no ports plugged in; 0 if port0 is active, 1 if port1 is active.

setAltModeConfig (*channel*, *configuration*)

Sets alt mode configuration for defined USB channel.

See the product datasheet for device specific details

Parameters

- **channel** (*int*) – The USB channel
- **configuration** (*uint*) – The configuration to set

Returns (int):

Non-zero BrainStem error code on failure.

setCC1Enable (*channel*, *enable*)

Enable CC1 lines for a Type C USB port

Parameters

- **channel** (*int*) – The USB port number
- **enable** (*int*) – enable (0 = disable, 1 = enable)

Returns (int):

Non-zero BrainStem error code on failure.

setCC2Enable (*channel*, *enable*)

Enable CC2 lines for a Type C USB port

Parameters

- **channel** (*int*) – The USB port number
- **enable** (*int*) – enable [0 = disable, 1 = enable]

Returns (int):

Non-zero BrainStem error code on failure.

setCableFlip (*channel*, *enable*)

Enables a cable orientation flip within the S85 switch.

Parameters

- **channel** (*int*) – The USB port number
- **enable** (*int*) – enables cable flip. [0 = disable, 1 = enable]

Returns (int):

Non-zero BrainStem error code on failure.

setConnectMode (*channel*, *mode*)

Set The connection mode for the Switch.

Parameters

- **channel** (*int*) – Upstream port to be applied.
- **mode** (*int*) – 0 = Manual mode, 1 = Auto mode.

Returns (int):

Non-zero BrainStem error code on failure.

setDataDisable (*channel*)

Disable just the data lines for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

setDataEnable (*channel*)

Enable just the data lines for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

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 (*int*) – Downstream boost setting 0, 1, 2, or 3.

Returns (int):

Non-zero BrainStem error code on failure.

setEnumerationDelay (*ms_delay*)**Set the interport enumeration delay in milliseconds.**

This setting must be saved with a `stem.system.save()` call for it to be active. This setting is persistent across hub power down. Resetting the hub will return this setting to the default value of 0ms.

Parameters

ms_delay (*int*) – Interport delay in milliseconds

Returns (int):

Non-zero BrainStem error code on failure.

setHiSpeedDataDisable (*channel*)

Disable Hi-Speed (USB2.0) data transfer for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

setHiSpeedDataEnable (*channel*)

Enable Hi-Speed (USB2.0) data transfer for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

setHubMode (*mode*)**Set a bit mapped representation of the hub mode.**

Usually represents the port data and power lines enable/disable. See the product datasheet for state mapping.

Parameters

mode (*int*) – The hub state

Returns (int):

Non-zero BrainStem error code on failure.

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 to make it persistent.

Parameters

- **channel** (*int*) – Port index.
- **microAmps** (*int*) – The current limit setting in microAmps (1A=10e6)

Returns (int):

Non-zero BrainStem error code on failure.

setPortDisable (*channel*)

Disable both power and data lines for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

setPortEnable (*channel*)

Enable both power and data lines for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

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 at

ef usbPortMode

Args:

channel (*int*): Port Index. **mode** (*int*): The port mode setting as packed bit field.

Returns (int):

Non-zero BrainStem error code on failure.

setPowerDisable (*channel*)

Disable just the power line for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

setPowerEnable (*channel*)

Enable just the power line for a USB port.

Parameters

channel (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

setSBUEnable (*channel*, *enable*)

Enable SBU1/SBU2 lines for a Type C USB port based on `usbPortMode` settings.

Parameters

- **channel** (*int*) – The USB port number
- **enable** (*int*) – enables SBU1/SBU2 [0 = disable, 1 = enable]

Returns (int):

Non-zero BrainStem error code on failure.

setSuperSpeedDataDisable (*channel*)

Disable SuperSpeed (USB3.0) data transfer for a USB port.

Parameters

- **channel** (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

setSuperSpeedDataEnable (*channel*)

Enable SuperSpeed (USB3.0) data transfer for a USB port.

Parameters

- **channel** (*int*) – The USB port number

Returns (int):

Non-zero BrainStem error code on failure.

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** (*int*) – Upstream boost setting 0, 1, 2, or 3.

Returns (int):

Non-zero BrainStem error code on failure.

setUpstreamMode (*mode*)

Set the upstream switch mode for the USB upstream ports

Parameters

- **mode** (*int*) –
 - Auto: `UPSTREAM_MODE_AUTO` = 2
 - Port 0: `UPSTREAM_STATE_PORT_0` = 0
 - Port 1: `UPSTREAM_STATE_PORT_1` = 1

Returns (int):

Non-zero BrainStem error code on failure.

3.2.30 USB System

class `brainstem.entity.USBSystem (module, index)`

The USBSystem class provides high level control of the lower level Port Class.

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

value (int): An enumerated representation of the current behavior. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataRoleBehaviorConfig ()

Gets the current data role behavior configuration Certain data role behaviors use a list of ports to determine priority host priority.

Returns

value (tuple(int)): A list of ports which indicate priority sequencing. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getDataRoleList ()

Gets the data role of all ports with a single call Equivalent to calling Port.getDataRole() on each individual port.

Returns

value (int): Bit packed representation of the data role for all ports. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getEnabledList ()

Gets the current enabled status of all ports with a single call. Equivalent to calling Port-Class::setEnabled() on each port.

Returns

value (int): Bit packed representation of the enabled status for all ports. error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getEnumerationDelay ()

Gets the inter-port enumeration delay in milli-seconds (mS). Delay is applied upon hub enumeration.

Returns

value (int): Current inter-port delay in milli-seconds (mS). error: Non-zero BrainStem error code on failure.

Return type

Result (object)

getModeList ()

Gets the current mode of all ports with a single call. Equivalent to calling Port-Class:getMode() on each port.

Returns

value (tuple(int)): List of modes of each port. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**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

value (int): An enumerated representation of the current behavior. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getPowerBehaviorConfig ()**

Gets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Returns

value (tuple(int)): A list of ports which indicate priority sequencing. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**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

value (int): The current mode error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getStateList ()**

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port.

Returns

value (tuple(int)): List of states for each port. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**getUpstream ()**

Gets the current upstream port.

Returns

value: Index of upstream port. error: Non-zero BrainStem error code on failure.

Return type*Result* (object)**resetEntityToFactoryDefaults ()**

Resets the USBSystemClass Entity to it factory default configuration.

Returns (int):

Non-zero BrainStem error code on failure.

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 (*int*) – An enumerated representation of the behavior to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setDataRoleBehaviorConfig (config)

Sets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine host priority.

Parameters

config (*tuple* (*int*)) – List of ports which indicate priority sequencing.

Returns (int):

Non-zero BrainStem error code on failure.

setEnabledList (enabledList)

Sets the enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

Parameters

enabledList (*int*) – Bit packed representation of the enabled status for all ports to be applied.

Returns (int):

Non-zero BrainStem error code on failure.

setEnumerationDelay (delay)

Set the inter-port enumeration delay in milliseconds. This setting should be saved with a stem.system.save() call. Delay is applied upon hub enumeration.

Parameters

delay (*int*) – Delay in milli-seconds (mS) to be applied between port enables.

Returns (int):

Non-zero BrainStem error code on failure.

setModeList (modeList)

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port.

Parameters

modeList (*tuple* (*int*)) – List of modes to be set for each port.

Returns (int):

Non-zero BrainStem error code on failure.

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 (*int*) – An enumerated representation of the behavior to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setPowerBehaviorConfig (config)

Sets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

config (*tuple* (*int*)) – List of ports which indicate priority sequencing.

Returns (int):

Non-zero BrainStem error code on failure.

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 (*mode*) – Mode to be set.

Returns (int):

Non-zero BrainStem error code on failure.

setUpstream (*port*)

Sets the upstream port.

Parameters

port (*int*) – Upstream port to be applied.

Returns (int):

Non-zero BrainStem error code on failure.

3.2.31 Version

Provides version access utilities.

`brainstem.version.get_version_string (packed_version=None)`

Returns the library version as a string

Parameters

packed_version (*int (Optional)*) – 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.

`brainstem.version.unpack_version (packed_version)`

Returns the library version as a 3-tuple (major, minor, patch)

Parameters

packed_version (*int (Optional)*) – The packed version number.

3.3 C++ API Reference

Welcome to the BrainStem C++ API reference documentation. This documentation covers the c++ Acroname BrainStem library. 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.](#)

The Getting started guide is particularly useful for learning how to use the application tools we provide to communicate with your hardware.

3.3.1 Errors

group **aErrors**

Unified list of Error codes for BrainStem module Iteration.

aError.h provides a unified list of error codes. These error codes apply accross all API's. Library functions will return one of these error codes when appropriate.

3.3.2 Classes

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.

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.

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.

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.

class EntityClass

Subclassed by *Acroname::BrainStem::AnalogClass*, *Acroname::BrainStem::AppClass*,
Acroname::BrainStem::ClockClass, *Acroname::BrainStem::DigitalClass*, *Acroname::BrainStem::EqualizerClass*,
Acroname::BrainStem::I2CClass, *Acroname::BrainStem::MuxClass*,
Acroname::BrainStem::PointerClass, *Acroname::BrainStem::PortClass*, *Acroname::BrainStem::PowerDeliveryClass*,
Acroname::BrainStem::RailClass, *Acroname::BrainStem::RCServoClass*, *Acroname::BrainStem::RelayClass*,
Acroname::BrainStem::SignalClass, *Acroname::BrainStem::StoreClass*, *Acroname::BrainStem::SystemClass*,
Acroname::BrainStem::TemperatureClass, *Acroname::BrainStem::TimerClass*, *Acroname::BrainStem::UARTClass*,
Acroname::BrainStem::USBClass, *Acroname::BrainStem::USBSystemClass*

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.

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.

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.

- The link can be fully specified with a transport and module serial number
- 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.

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*, *aUSBCSwitch*, *aUSBHub2x4*, *aUSBHub3c*, *aUSBHub3p*

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).

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 and incrementing pointer.

class **PortClass** : public Acroname::BrainStem::EntityClass

Port Class: 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.

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.

class **RailClass** : public Acroname::BrainStem::EntityClass

RailClass: Provides power rail functionality on certain modules. This entity is only available on certain modules. The **RailClass** can be used to control power to downstream devices, I has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

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.

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.

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.

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.

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. The most common brainstem example uses the system entity to blink the User LED.

class **TemperatureClass** : public Acroname::BrainStem::EntityClass

TemperatureClass: This entity is only available on certain modules, and provides a temperature reading in microcelsius.

class **TimerClass** : public Acroname::BrainStem::EntityClass

TimerClass: The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

class **UARTClass** : public Acroname::BrainStem::EntityClass

UART Class: 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.

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.

class **USBSystemClass** : public Acroname::BrainStem::EntityClass

USBSystem Class: The USBSystem class provides high level control of the lower level Port Class.

Subclassed by *aUSBHub3c::HubClass*

3.3.3 Acroname Modules

Hubs & Switches

USBHub3c

USBHub3p

USBHub2x4

USBCSwitch

MTM Modules

MTM-DAQ-2

MTM-EtherStem

MTM-IO-Serial

MTM-Load-1

MTM-PM-1

MTM-Relay

MTM-USBStem

Abstract Classes

MTM-Stem Module

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::*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

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::*USBSystemClass*

Hub class implementation for use with USBHub3c.

Defines

aUSBHUB3C_MODULE 6

USBHub3c module number

aUSBHUB3C_NUM_APPS 4

Number of App instances available

aUSBHUB3C_NUM_POINTERS 4

Number of Pointer instances available

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_TIMERS 8

Number of Timer instances available

aUSBHUB3C_NUM_USB 1

Number of USB instances available

aUSBHUB3C_NUM_USB_PORTS 8

Number of USB 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 1

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 Members

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

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

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 Members

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

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

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

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::*AppClass* **app**[aUSBCSWITCH_NUM_APPS]

App Class

Acroname::BrainStem::*MuxClass* **mux**

Mux Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBCSWITCH_NUM_POINTERS]

Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBCSWITCH_NUM_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBCSWITCH_NUM_TIMERS]
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_APPS 4
Number of App instances available

aUSBCSWITCH_NUM_POINTERS 4
Number of Pointer instances available

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_TIMERS 8
Number of Timer 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

set_usbPortStateCOM_ORIENT_STATUS (var, state) ((var & ~(3 << 8)) | (state << 8))

Common side orientation status

get_usbPortStateCOM_ORIENT_STATUS (var) ((var & (3 << 8)) >> 8)

Common side orientation status

set_usbPortStateMUX_ORIENT_STATUS (var, state) ((var & ~(3 << 10)) | (state << 10))

Mux side orientation status

get_usbPortStateMUX_ORIENT_STATUS (var) ((var & (3 << 10)) >> 10)

Mux side orientation status

set_usbPortStatesPEED_STATUS (var, state) ((var & ~(3 << 12)) | (state << 12))

USB speed status

get_usbPortStatesPEED_STATUS (var) ((var & (3 << 12)) >> 12)

USB speed status

usbPortStateCCFlip 14

CC flip status

usbPortStateSSFlip 15

SS flip status

usbPortStateSBUFlip 16

SBU flip status

usbPortStateUSB2Flip 17

USB2 flip status

get_usbPortStateDaughterCard (var) ((var & (3 << 18)) >> 18)

Daughter card status

usbPortStateErrorFlag 20

Error indicator for this port

usbPortStateUSB2Boost 21

USB2 boost current state

usbPortStateUSB3Boost 22

USB3 boost current state

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

Port State Error Defines

usbPortStateOff 0

Indicator for port state off

usbPortStateSideA 1

Indicator for port side A

usbPortStateSideB 2

Indicator for port side B

usbPortStateSideUndefined 3

Indicator for port side undefined

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 Members

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

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_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_MILLIOHMS 1000000000

Rail: Max resistance in milliohms

aMTMLOAD1_MIN_MILLIOHMS 0

Rail: Min resistance in milliohms

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

3.3.4 Analog Class

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.

~AnalogClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the analog entity being initialized.

aErr **getValue** (uint16_t *value)

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.

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.

Return values

aErrConfiguration – - Entity does not support this configuration.

Returns

EntityReturnValues “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.

Returns

Returns common entity return values. When the bulk capture is complete *getBulkCaptureState()* will return either bulkCaptureFinished or bulkCaptureError.

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

3.3.5 App Class

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.

~AppClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module.
- **index** – The cmdAPP reflex index to be addressed.

aErr **execute** (const uint32_t appParam)

Execute the app reflex on the module. Don'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.

Returns

aErrNone - success.

Returns

aErrTimeout - The request timed out waiting to start execution.

Returns

aErrConnection - No active link connection.

Returns

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. Wait 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.

Returns

aErrNone - success.

Returns

aErrTimeout - The request timed out waiting for a response.

Returns

aErrConnection - No active link connection.

Returns

aErrNotFound - the app reflex was not found or not enabled on the module.

3.3.6 Clock Class

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 class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the clock entity being initialized.

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 *min)

Get the two digit minute value (0-59).

Parameters

min – The two digit minute portion of the real-time clock value.

Returns

Returns common entity return values

aErr **setMinute** (const uint8_t min)

Set the two digit minute value (0-59).

Parameters

min – The two digit minute portion of the real-time clock value.

Returns

Returns common entity return values

aErr **getSecond** (uint8_t *sec)

Get the two digit second value (0-59).

Parameters

sec – The two digit second portion of the real-time clock value.

Returns

Returns common entity return values

aErr **setSecond** (const uint8_t sec)

Set the two digit second value (0-59).

Parameters

sec – The two digit second portion of the real-time clock value.

Returns

Returns common entity return values

3.3.7 Digital Class

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 class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the digital entity being initialized.

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 –

- 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

Returns

aErrConfiguration - Entity does not support this configuration.

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.

Parameters

state – 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.

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

3.3.8 Entity Class

class **EntityClass**

Subclassed by *Acroname::BrainStem::AnalogClass*, *Acroname::BrainStem::AppClass*,
Acroname::BrainStem::ClockClass, *Acroname::BrainStem::DigitalClass*, *Acron-*
ame::BrainStem::EqualizerClass, *Acroname::BrainStem::I2CClass*, *Acron-*
ame::BrainStem::MuxClass, *Acroname::BrainStem::PointerClass*, *Acron-*
ame::BrainStem::PortClass, *Acroname::BrainStem::PowerDeliveryClass*, *Acron-*
ame::BrainStem::RailClass, *Acroname::BrainStem::RCServoClass*, *Acron-*
ame::BrainStem::RelayClass, *Acroname::BrainStem::SignalClass*, *Acron-*
ame::BrainStem::StoreClass, *Acroname::BrainStem::SystemClass*, *Acron-*
ame::BrainStem::TemperatureClass, *Acroname::BrainStem::TimerClass*, *Acron-*
ame::BrainStem::UARTClass, *Acroname::BrainStem::USBCClass*, *Acron-*
ame::BrainStem::USBSystemClass

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 **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** (std::map<uint64_t, uint32_t> *status)

Gets all available stream values associated with the cmd and index of the called API. Keys can be decoded with [Link::getStreamKeyElement](#).

Parameters

status – map of option value pairs to be filled. Option codes are based on the cmd and index of the calling API.

Returns

aErrParam if status is null

Returns

aErrResource - if the link is not valid

Returns

aErrNone - success

Public Static Functions

static uint8_t **getUEIBytesSequence** (const uint8_t sequence)

Parameters

sequence – - Sequence byte to be checked.

Returns

The sequence number of the byte.

static bool **getUEIBytesContinue** (const uint8_t sequence)

Parameters

sequence – - Sequence byte to be checked.

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

3.3.9 Equalizer Class

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.

~EqualizerClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module.
- **index** – The index.

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.

3.3.10 I2C Class

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 class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the digital entity being initialized.

aErr **read** (const uint8_t address, const uint8_t length, uint8_t *result)

Read from a device on this I2C bus.

Parameters

- **address** – - The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** – - The length of the data to read in bytes.
- **result** – - 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 length, const uint8_t *data)

Write to a device on this I2C bus.

Parameters

- **address** – - The I2C address (7bit <XXXX-XXX0>) of the device to write.
- **length** – - The length of the data to write in bytes.
- **data** – - 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 bool bEnable)

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

- **bEnable** – - 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 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**

typedef enum Acroname::BrainStem::Link::STREAM_PACKET STREAM_PACKET_t

Enumeration of stream packet types.

typedef std::function<aErr(const aPacket *packet, void *pRef)> streamCallback_t

Function signature for streaming callbacks.

Param packet

reference to streaming packet

Param pRef

User provided reference

Return

aErrNone - Success. Return value is not currently used.

typedef enum Acroname::BrainStem::Link::STREAM_KEY STREAM_KEY_t

Enumeration for element types within a stream key.

Public Functions

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.

Returns

aErrBusy - if the module is already in use.

Returns

aErrParam - if the transport type is undefined.

Returns

aErrNotFound - if the module cannot be found or if no modules found.

Returns

aErrNone - If the connect was successful.

aErr **connect** (void)

Connect to a link with a fully defined specifier.

Returns

aErrBusy - if the module is running, starting or stopping. Try again in a bit.

Returns

aErrDuplicate - If the module is already connected and running.

Returns

aErrConnection - If there was an error with the connection. User needs to disconnect, then reconnect.

Returns

aErrConfiguration - If the link has an invalid *linkSpec*.

Returns

aErrNotFound - if the module cannot be found.

Returns

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.

Returns

aErrResource - If the there is no valid connection.

Returns

aErrConnection - If the disconnect failed, due to a communication issue.

Returns

aErrNone If the disconnect was successful.

aErr **reset** (void)

Reset The underlying link stream.

Returns

aErrResource - If the there is no valid connection.

Returns

aErrConnection - If the reset failed, due to a communication issue.

Returns

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 specifcaiton.

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.

Returns

aErrConnection - link not connected.

Returns

aErrParam - data too long or short.

Returns

aErrPacket - invalid module address.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrParam - data too long or short.

Returns

aErrPacket - invalid module address.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrPacket - invalid module address.

Returns

aErrTimeout - no packet available.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrPacket - invalid module address.

Returns

aErrTimeout - no packet available.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrPacket - invalid module address.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrParam - data too long or short.

Returns

aErrPacket - invalid module address.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrPacket - invalid module address.

Returns

aErrTimeout - no packet available.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrParam - invalid module address.

Returns

aErrCancel - The write process is closing and this call was unable to successfully complete.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrParam - invalid module address.

Returns

aErrCancel - The write process is closing and this call was unable to successfully complete.

Returns

aErrOverrun - The read would overrun the buffer, i.e there is more data in the slot than the buffer can handle.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrParam - invalid module address.

Returns

aErrCancel - The write process is closing and this request was unable to successfully complete.

Returns

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.

Returns

aErrConnection - link not connected.

Returns

aErrParam - invalid module address.

Returns

aErrCancel - The write process is closing and this request was unable to successfully complete.

Returns

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, std::map<uint64_t, uint32_t> *status)

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)
- **status** – map of key value pairs to be filled based on the parameters.
Link::getStreamKeyElement should be used to decode the keys

Returns

aErrParam if status 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 **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)

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.

Returns

aErrNotFound if no devices were found.

Returns

aErrNone on success.

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.

Returns

true To continue processing, or false to stop processing.

static inline *aErr* **sDiscover** (const *linkType* type, list<*linkSpec*> *devices)

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.

Returns

aErrNotFound if no devices were found.

Returns

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.

Returns

aErrPacket - Not a stream packet

Returns

aErrUnknown - Unknown decoding issue.

Returns

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

3.3.12 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](#), [aUSBCSwitch](#), [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.

Returns

[aErrBusy](#) - if the module is already in use.

Returns

aErrParam - if the type is incorrect or serialNum is not specified

Returns

aErrNotFound - if the module cannot be found.

Returns

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.

Returns

aErrInitialization - If there is currently no link object.

Returns

aErrBusy - If the link is currently connected.

Returns

aErrParam - if the specifier is incorrect.

Returns

aErrNotFound - if the module cannot be found.

Returns

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.

Returns

aErrBusy - if the module is already in use.

Returns

aErrParam - if the transport type is undefined.

Returns

aErrNotFound - if the module cannot be found.

Returns

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

Parameters

pModule -- Pointer to a valid *Module* class object.

Returns

aErrParam - if the module is undefined.

Returns

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.

Returns

aErrResource - If there is no valid connection.

Returns

aErrConnection - If the disconnect failed, due to a communication issue.

Returns

aErrNone If the disconnect was successful.

aErr **reconnect** ()

Reconnect using the current link specifier.

Returns

aErrBusy - if the module is already in use.

Returns

aErrParam - if the specifier is incorrect.

Returns

aErrNotFound - if the module cannot be found.

Returns

aErrNone If the connect was successful.

Link ***getLink** (void) const

Get the current link object.

Returns

The link associated with the module.

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 **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).

Returns

aErrNone - The module supports this command and access flags.

Returns

aErrMode - The module supports the command but not the access flag.

Returns

aErrNotFound - The module does not support the command, option, or index.

Returns

aErrTimeout - The request timed out without a response.

Returns

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.

Returns

aErrNone - Success.

Returns

aErrTimeout - The request timed out without a response.

Returns

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.

Returns

aErrNone - Success.

Returns

aErrTimeout - The request timed out waiting for response.

Returns

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.

Returns

aErrNone - Success.

Returns

aErrTimeout - The request timed out without response.

Returns

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.

Returns

aErrNone - Success.

Returns

aErrTimeout - Timeout occurred without response.

Returns

aErrConnection - No active link exists.

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

3.3.13 Mux Class

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 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).

Public Functions

MuxClass (void)

Constructor.

~MuxClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity, i.e. aMUX_UART or aMUX_USB.

aErr **getEnable** (uint8_t *bEnabled)

Get the mux enable/disable status

Parameters

- **bEnabled** – true: mux is enabled, false: the mux is disabled.

Returns

Returns common entity return values

aErr **setEnable** (const uint8_t bEnable)

Enable the mux.

Parameters

- **bEnable** – 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 muxConfig_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

3.3.14 Pointer Class

```
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 and incrementing pointer.

Public Functions

PointerClass (void)

Constructor.

~PointerClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index the pointer element index.

aErr **getOffset** (uint16_t *offset)

Get the offset of the pointer

Parameters

offset – The value of the offset.

Returns

All possible standard UEI return values.

aErr **setOffset** (uint16_t offset)

Set the offset of the pointer

Parameters

offset – The value of the offset.

Returns

All possible standard UEI 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

All possible standard UEI return values.

aErr **setMode** (uint8_t mode)

Set the mode of the pointer

Parameters

mode – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Returns

All possible standard UEI return values.

aErr **getTransferStore** (uint8_t *handle)

Get the handle to the store.

Parameters

handle – The handle of the store.

Returns

All possible standard UEI return handles.

aErr **setTransferStore** (uint8_t handle)

Set the handle to the store.

Parameters

handle – The handle of the store.

Returns

All possible standard UEI return handles.

aErr **initiateTransferToStore** (uint8_t length)

Transfer data to the store.

Parameters

length – The length of the data transfer.

Returns

All possible standard UEI return values.

aErr **initiateTransferFromStore** (uint8_t length)

Transfer data from the store.

Parameters

length – The length of the data transfer.

Returns

All possible standard UEI 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

All possible standard UEI 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

All possible standard UEI 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

All possible standard UEI 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

All possible standard UEI 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

All possible standard UEI 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

All possible standard UEI return values.

3.3.15 Port Class

class **PortClass** : public Acroname::BrainStem::EntityClass

Port Class: 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.

~PortClass (void)

Destructor.

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 setDataHSEnabled.

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 setDataHSEnabled.

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 setDataEnabled.

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 (SS) 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.

aErr **getDataSpeed** (uint8_t *speed)

Gets the speed of the enumerated device.

Parameters

speed – Bit mapped value representing the devices speed. See product datasheet 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 product datasheet 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 product datasheet 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 bufLength, 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
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr **setName** (uint8_t *buffer, const size_t bufLength)

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.
- **bufLength** – Length of the buffer to be transferred.

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 **resetVbusAccumulatedPower** (void)

Resets the Vbus Accumulated Power to zero.

Returns

Returns common entity return values

aErr **getVconnAccumulatedPower** (int32_t *milliwatthours)

Gets the Vconn Accumulated Power

Parameters

milliwatthours – The accumuled power on Vconn in milliwatt-hours.

Returns

Returns common entity return values

aErr **resetVconnAccumulatedPower** (void)

Resets the Vconn Accumulated Power to zero.

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 **resetEntityToFactoryDefaults** (void)

Resets the *PortClass* Entity to it factory default configuration.

Returns

Returns common entity return values

3.3.16 Power Delivery Class

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.

~PowerDeliveryClass (void)

Destructor.

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 bufLength, 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.
- **bufLength** – 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. 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 uint8_t partner, 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. 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
- **rdo** – Request Data Object to be set.

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 **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)

Returns

The returned error represents the success of the request being sent to the partner only. The success of the request being serviced by the remote partner can be obtained through [PowerDeliveryClass::requestStatus\(\)](#) 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 **resetEntityToFactoryDefaults** (void)

Resets the *PowerDeliveryClass* Entity to it factory default configuration.

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

aErrNone on success; aErrParam with bad input.

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

aErrNone on success; aErrParam with bad input.

3.3.17 Rail Class

class **RailClass** : public Acroname::BrainStem::EntityClass

RailClass: Provides power rail functionality on certain modules. This entity is only available on certain modules. The *RailClass* can be used to control power to downstream devices, I has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

Public Functions

RailClass (void)

Constructor.

~RailClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity. Each rail index refers to a specific hardware voltage plane or “rail”. Refer to the module datasheet for definition of the hardware voltage planes and specific capabilities.

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 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 *bEnable)

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

bEnable – true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage

Returns

Returns common entity return values

aErr **setEnabled** (const uint8_t bEnable)

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

bEnable – 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 *milliwatts)

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

milliwatts – The power in milli-watts (1 == 1e-3W) 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 milliwatts)

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

milliwatts – The power in milli-watts (1 == 1e-3W) to be supplied by the rail.

Returns

Returns common entity return values

aErr **getPowerSetpoint** (int32_t *milliwatts)

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

milliwatts – The power in milli-watts (1 == 1e-3W) 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 milliwatts)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

milliwatts – The power in milli-watts (mW).

Returns

Returns common entity return values

aErr **getPowerLimit** (int32_t *milliwatts)

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

milliwatts – The power in milli-watts (mW).

Returns

Returns common entity return values

aErr **getResistance** (int32_t *milliohms)

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

milliohms – The resistance in milli-ohms (1 == 1e-3Ohms) 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 milliohms)

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

milliohms – The resistance in milli-ohms (1 == 1e-3Ohms) to be drawn by the rail.

Returns

Returns common entity return values

aErr **getResistanceSetpoint** (int32_t *milliohms)

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

milliohms – The resistance in milli-ohms (1 == 1e-3Ohms) 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 bEnable)

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

bEnable – enable or disable kelvin sensing.

Returns

Returns common entity return values

aErr **getKelvinSensingEnable** (uint8_t *bEnable)

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

bEnable – 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 RCServo Class

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 class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the servo entity being initialized.

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 **getEnable** (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”. ie. 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

3.3.19 Relay Class

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 class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the digital entity being initialized.

aErr **setEnabled** (const uint8_t bEnable)

Set the enable/disable state.

Parameters

bEnable – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns common entity return values

aErr **getEnabled** (uint8_t *bEnabled)

Get the state.

Parameters

bEnabled – 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

3.3.20 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.

~SignalClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module.
- **index** – The index.

aErr **setEnabled** (const uint8_t enable)

Enable/Disable the signal output.

Parameters

enable – True to enable, false to disable

Returns

Returns common entity return values

aErr **getEnabled** (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 – 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 – 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

3.3.21 Store Class

```
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.

~StoreClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module.
- **index** – The index.

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 *pData, const uint16_t length)

Load the slot.

Parameters

- **slot** – The slot number.
- **pData** – The data.
- **length** – The data length.

Returns

Returns common entity return values

aErr **unloadSlot** (const uint8_t slot, const size_t dataLength, uint8_t *pData, size_t *unloadedLength)

Unload the slot data.

Parameters

- **pData** – Byte array that the unloaded data will be placed into.
- **dataLength** – - The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than dataLength.
- **slot** – The slot number.

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

3.3.22 System Class

```
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. The most common brainstem example uses the system entity to blink the User LED.

Public Functions

SystemClass (void)

Constructor.

virtual **~SystemClass** (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the aSystem class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – Not used; always 0.

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 bOn)

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

bOn – true: turn the LED on, false: turn LED off.

Returns

Returns common entity return values

aErr **getLED** (uint8_t *bOn)

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

bOn – true: LED on, false: LED off.

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.

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 32bit 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.

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 persist 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 persist 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 bOn)`

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

bOn – 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)

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 bufLength, size_t *unloadLength)

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
- **bufLength** – Length of the buffer to be filled
- **unloadLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr `setInputPowerBehaviorConfig` (uint32_t *buffer, const size_t bufLength)

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.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

aErr `getName` (uint8_t *buffer, const size_t bufLength, size_t *unloadLength)

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
- **bufLength** – Length of the buffer to be filled
- **unloadLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr **setName** (uint8_t *buffer, const size_t bufLength)

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.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

aErr **resetEntityToFactoryDefaults** (void)

Resets the *SystemClass* Entity to it factory default configuration.

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

3.3.23 Temperature Class

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.

~TemperatureClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity.

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

aErr **resetEntityToFactoryDefaults** (void)

Resets the *TemperatureClass* Entity to it factory default configuration.

3.3.24 Timer Class

class **TimerClass** : public Acroname::BrainStem::EntityClass

TimerClass: The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

Public Functions

TimerClass (void)

Constructor.

~TimerClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the timer entity.

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 time. 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

Returns

aErrNone - Action completed successfully.

3.3.25 UART Class

class **UARTClass** : public Acroname::BrainStem::EntityClass

UART Class: 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.

~UARTClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity, i.e. aMUX_UART or aMUX_USB.

aErr **setEnabled** (const uint8_t bEnabled)

Enable the UART channel.

Parameters

bEnabled – true: enabled, false: disabled.

Returns

Returns common entity return values

aErr **getEnabled** (uint8_t *bEnabled)

Get the enabled state of the uart.

Parameters

bEnabled – true: enabled, false: disabled.

Returns

Returns common entity return values

aErr **setBaudRate** (const uint32_t rate)

Set the UART baud rate.

Parameters

rate – baud rate.

Returns

Returns common entity return values

aErr **getBaudRate** (uint32_t *rate)

Get the UART baud rate.

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

3.3.26 USB Class

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.

~USBClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity, i.e. the port

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, Super-Speed (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 and usbUpstreamModePort1

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 at [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 implement a common bit mapping for port mode at [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 bEnable)

Set Enable/Disable on the CC1 line.

Parameters

- **channel** – - USB channel.
- **bEnable** –
 - 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** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

aErr **setCC2Enable** (const uint8_t channel, const uint8_t bEnable)

Set Enable/Disable on the CC2 line.

Parameters

- **channel** – - USB channel.
- **bEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

aErr **getCC2Enable** (const uint8_t channel, uint8_t *pEnable)

Get Enable/Disable on the CC1 line.

Parameters

- **channel** – - USB channel.
- **pEnable** –
 - 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 bEnable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

Parameters

- **channel** – The USB sub channel.
- **bEnable** –
 - 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 bEnable)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters

- **channel** – The USB sub channel.
- **bEnable** –
 - 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

3.3.27 USBSystem Class

class **USBSystemClass** : public Acroname::BrainStem::EntityClass

USBSystem Class: The USBSystem class provides high level control of the lower level Port Class.

Subclassed by *aUSBHub3c::HubClass*

Public Functions

USBSystemClass (void)

Constructor.

~USBSystemClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity, i.e. the port

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 bufLength, 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
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr `setModeList` (uint32_t *buffer, const size_t bufLength)

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.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

aErr **getStateList** (uint32_t *buffer, const size_t bufLength, 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
- **bufLength** – 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 bufLength, 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
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr **setPowerBehaviorConfig** (uint32_t *buffer, const size_t bufLength)

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.
- **bufLength** – 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 bufLength, 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
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr `setDataRoleBehaviorConfig` (uint32_t *buffer, const size_t bufLength)

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.
- **bufLength** – 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 `resetEntityToFactoryDefaults` (void)

Resets the *USBSystemClass* Entity to it factory default configuration.

3.4 C API Reference

This section of the *Acroname BrainStem Reference* covers the lower level C programming interface. 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.*

Most users will want to begin with the C++ API or Python libraries, but for some specialized applications, or applications that must be ANSI C only, the C API provides access a little closer to the metal than the other APIs. To use the C api effectivley you will need to understand a little more about the BrainStem protocol, and particularly about UEI's and their significance.

- *Appendix: Brainstem Universal Entity Interface*
 - *Appendix: The BrainStem Communication Protocol*
-

3.4.1 Modules

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`

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.

group **aErrors**

Unified list of Error codes for BrainStem module Iteration.

aError.h provides a unified list of error codes. These error codes apply accross all API's. Library functions will return one of these error codes when appropriate.

group aFile

Platform Independent File Access Interface.

[aFile.h](#) provides a platform independent interface for opening, reading, and writing files.

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.

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.

group aPacket

BrainStem Packet.

[aPacket.h](#) Provides an interface for creating and destroying BrainStem Protocol packets.

group aProtocoldefs

BrainStem Protocol Definitions.

[aProtocoldefs.h](#) Provides protocol and BrainStem specific defines for entities, communication, and protocol specifics.

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.

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.

group aVersion

Library version interface.

[aVersion.h](#) Provides version information for the BrainStem2 library.

group aUEI

UEI Utilities.

[aUEI.h](#) Provides structs and utilities for working with UEIs.

3.4.2 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.

3.4.3 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 **TCP/IP**

- TCP/IP link type.

enumerator **SERIAL**

- Serial 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)
```

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.

Returns

Returns the number of modules found.

```
linkSpec *aDiscovery_FindModule (const linkType type, uint32_t serialNum)
```

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.

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)
```

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

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*.

3.4.4 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 to 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.

3.4.5 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.

3.4.6 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 BrainStem link reference.

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 TCPIP address

Returns

aLinkRef identifier if successful or 0 otherwise.

aLinkRef **aLink_CreateTCPIP** (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

- **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 successful completion of the Destroy call.

Returns

aStreamRef Return value will always be NULL. The return value has been left for backwards compatibility.

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 outgoing link FIFO.

Put a packet into the outgoing 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.

3.4.7 aMutex.h

group **aMutex**

Platform Independent Synchronization Primitive.

aMutex.h Provides a platform independent synchronization mechanism. The link interface and the packed 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 acquired 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.

Relinquish 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.

3.4.8 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.

3.4.9 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`

System Entity

group `cmdSYSTEM_Defines`

System entity defines.

Defines

cmdSYSTEM 3

3 - System entity 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

systemIPModeStatic 1

systemIPModeDefault 0

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 31
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
36 -

systemResetDeviceToFactoryDefaults 37
37 -

systemLinkInterface 38
38 - Setting the link interface for control

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)

systemReserved 39

39 - Reserved Option Code for Acroname Internal Use Only

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 option code.

systemErrors_OutputPowerProtection_Bit 1

1 - Output Power Protection bit for operational Errors option code.

systemNumberOfOptions 45

45 - Number of Options for System, always last entry

Slot Entity

group **cmdSLOT_Defines**

System entity defines.

Defines

cmdSLOT 4

4 - Slot Command Code.

group **cmdSLOT_Command_Options**

Defines

slotCapacity 1

1 - Slot Capacity option code.

slotSize 2

2 - Slot size option code

slotOpenRead 3

3 - Slot Open Read option code.

slotOpenWrite 4

4 - Slot Open Write option code.

slotSeek 5

5 - Slot Seek option code.

slotRead 6

6 - Slot Read option code.

slotWrite 7

7 - Slot Write option code.

slotClose 8

8 - Slot Close option code.

bitSlotError 0x80

0x80 - Bit Slot error code.

App Entity

group **cmdAPP_Defines**

App Entity 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.

Mux Entity

group **cmdMUX_Defines**

Mux Entity defines.

Defines

cmdMUX 6

6 - Mux command code.

group **cmdMUX_Command_Options**

Defines

muxEnable 1

1 - Channel enable option code.

muxChannel 2

2 - Select the active channel on the mux.

muxVoltage 3

3 - Get voltage measurement for the channel.

muxConfig 4

4 - Get voltage measurement for the channel.

muxConfig_default 0

muxConfig_splitMode 1

muxConfig_channelpriority 2

muxSplit 5

5 - Get voltage measurement for the channel.

Pointer Entity

group **cmdPOINTER_Defines**

Pointer entity defines.

Defines

cmdPOINTER 7

7 - Pointer command code.

group **cmdPOINTER_Command_Options**

Defines

pointerOffset 1

1 - Pointer offset option code.

pointerMode 2

2 - Pointer mode option code.

pointerModeStatic 0

0 - Static pointer mode for pointer mode option code.

pointerModeIncrement 1

1 - Increment pointer mode for pointer mode option code.

DefaultPointerMode 0

pointerModeStatic - Default pointer mode for pointer mode option code.

pointerTransferStore 3

3 - Set Transfer store option code.

pointerChar 4

4 - Char pointer option code.

pointerShort 5

5 - Short pointer option code.

pointerInt 6

6 - Int pointer option code.

pointerTransferToStore 7

7 - Transfer to Store option code.

pointerTransferFromStore 8

8 - Transfer From store option code.

Debug command

cmdDEBUG 23

Debug command.

Analog Entity

group **cmdANALOG_Defines**

Analog Entity 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 for configuration option code.

analogConfigurationOutput 1

1 - Output configuration for configuration option code.

analogConfigurationHiZ 2

2 - HiZ configuration for configuration option code.

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 for Bulk capture Sample Rate option code.

analog_Hz_Maximum 200000

200000 - maximum hertz sample rate for Bulk capture Sample Rate option code.

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 for Bulk Capture state option code.

bulkCapturePending 1

1 - Pending state for Bulk Capture state option code.

bulkCaptureFinished 2

2 - Finished state for Bulk Capture state option code.

bulkCaptureError 3

3 - Error state for Bulk Capture state option code.

analogRange 8

8 - Analog Range option code.

analogRange_P0V064N0V064 0

0 - +/- 64mV range for Analog Range option code.

analogRange_P0V64N0V64 1

1 - +/- 640mV range for Analog Range option code.

analogRange_P0V128N0V128 2

2 - +/- 128mV range for Analog Range option code.

analogRange_P1V28N1V28 3

3 - +/- 1.28V range for Analog Range option code.

analogRange_P1V28N0V0 4

4 - 0-1.28V range for Analog Range option code.

analogRange_P0V256N0V256 5

5 - +/- 256mV range for Analog Range option code.

analogRange_P2V56N2V56 6

6 - +/- 2.56V range for Analog Range option code.

analogRange_P2V56N0V0 7

7 - 0-2.56V range for Analog Range option code.

analogRange_P0V512N0V512 8

8 - +/- 512mV range for Analog Range option code.

analogRange_P5V12N5V12 9

9 - +/- 5.12V range for Analog Range option code.

analogRange_P5V12N0V0 10

10 - 0-5.12V range for Analog Range option code.

analogRange_P1V024N1V024 11

11 - +/- 1.024V range for Analog Range option code.

analogRange_P10V24N10V24 12

12 - +/- 10.24V range for Analog Range option code.

analogRange_P10V24N0V0 13

13 - 0-10.24V range for Analog Range option code.

analogRange_P2V048N0V0 14

14 - 0-2.048V range for Analog Range option code.

analogRange_P4V096N0V0 15

15 - 0-4.096V range for Analog Range option code.

analogEnable 9

9 - Analog Enable option code.

Digital Entity

group **cmdDIGITAL_Defines**

Digital entity defines.

Defines

cmdDIGITAL 31

31 - Digital command code.

group **cmdDIGITAL_Command_Options**

Defines

digitalConfiguration 1

1 - Digital configuration option code.

digitalConfigurationInput 0x00

0 - Input Digital configuration for configuration option code.

digitalConfigurationOutput 0x01

1 - Output Digital configuration for configuration option code.

digitalConfigurationRCServoInput 0x02

2 - RC Servo Input Digital configuration for configuration option code.

digitalConfigurationRCServoOutput 0x03

3 - RC Servo Output Digital configuration for configuration option code.

digitalConfigurationHiZ 0x04

4 - Hi Z the digital pin.

digitalConfigurationInputPullUp 0x00

0 - Input digital configuration with pull-up.

digitalConfigurationInputNoPull 0x04

4 - Input digital configuration with no pull-up/pull-down.

digitalConfigurationInputPullDown 0x05

5 - Input digital configuration with pull-down.

digitalConfigurationSignalOutput 0x06

6 - Signal output configuration

digitalConfigurationSignalInput 0x07

7 - Signal input configuration

digitalConfigurationSignalCounterInput 0x08

8 - Signal input conter configuration

digitalState 2

9 - State option code.

digitalStateAll 3

Rail Entity

group **cmdRAIL_Defines**

Rail entity 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 option code.

kelvinSensingOn_Value 1

1 - Kelvin Sensing on mode for Kelvin Sensing mode option code.

railKelvinSensingState 8

8 - Kelving Sensing state option code.

railOperationalMode 9

9 - Operational mode option code. railOperationalMode is a bit masked field with two multi bit fields.

railOperationalMode_HardwareConfiguration_Offset 0

0-3 - Operational Mode hardware configuration offset region (bits[0:3]).

railOperationalModeAuto_Value 0

0 - Auto operational mode for operational mode option code.

railOperationalModeLinear_Value 1

1 - Linear mode for operational mode option code.

railOperationalModeSwitcher_Value 2

2 - Switcher mode for operational mode option code.

railOperationalModeSwitcherLinear_Value 3

3 - Switcher Linear mode for operational mode option code.

railOperationalMode_Mode_Offset 4

4-7 - Operational Mode offset region (bits[4:7]).

railOperationalModeConstantCurrent_Value 0

0 - Constant Current mode for operational mode option code.

railOperationalModeConstantVoltage_Value 1

1 - Constant Voltage mode for operational mode option code.

railOperationalModeConstantPower_Value 2

2 - Constant Power mode for operational mode option code.

railOperationalModeConstantResistance_Value 3

3 - Constant Resistance mode for operational mode option code.

railOperationalModeFactoryReserved_Value 0xF

15 - Factory Reserved Operating Mode.

DefaultOperationalRailMode_Value 0

0 - Default operational mode for operational mode option code.

railOperationalState 10

10 - Operational state option code. The railOperationalState is a bit masked field that has single bit and multi-bit entries.

railOperationalState_Initializing_Bit 0

0 - Initializing bit for operational state option code.

railOperationalState_Enabled_Bit 1

1 - Enabled bit for operational state option code.

railOperationalState_Fault_Bit 2

2 - Fault bit for operational state option code.

railOperationalState_HardwareConfiguration_Offset 8

3-7 These bits are unused **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

23 - This bit is Unused 24-31 - Operating Mode region (bits[24:31]) for operational state.

railOperationalStateConstantCurrent_Value 0

0 - Constant Current mode for operational state option code.

railOperationalStateConstantVoltage_Value 1

1 - Constant Voltage mode for operational state option code.

railOperationalStateConstantPower_Value 2

2 - Constant Power mode for operational state option codes.

railOperationalStateConstantResistance_Value 3

3 - Constant Resistance mode for operational state option code.

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.

railFactoryReserved 62

63 - Factory Reserved Code.

railFactoryReserved2 63

63 - Factory Reserved Code.

Temperature Entity

group **cmdTEMPERATURE_Defines**

Temperature entity defines.

Defines

cmdTEMPERATURE 33

33 - Temperature command code.

group **cmdTEMPERATURE_Command_Options**

Defines

temperatureMicroCelsius 1

1 - Temperature option code.

temperatureMinimumMicroCelsius 2

2 - Min temperature option code.

temperatureMaximumMicroCelsius 3

3 - Max temperature option code.

temperatureResetEntityToFactoryDefaults 4

4 Reset temperature entity option code

temperatureNumberOfOptions 5

2 - Number of Options for temperature, always last entry

Capacity Command

group **cmdCAPACITY_Defines**

Capacity command.

Defines

cmdCAPACITY 73

73 - Capacity command code.

group **cmdCAPACITY_Command_Options**

Defines

capacityUEI 1

1 - UEI command option.

capacitySubClassSize 3

3 - SubClass size command option.

capacityClassQuantity 4

4 - Class Quantity command option.

capacitySubClassQuantity 5

5 - SubClass Quantity command option.

capacityEntityGroup 6

6 - Entity Group command option.

capacityBuild 255

7 - Build command option.

Store Entity

group **cmdSTORE_Defines**

Store entity 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 - Write Slot option code.

storeReadSlot 5

5 - Read Slot option code.

storeCloseSlot 6

6 - Close Slot option code.

storeLock 7

7 - Lock Slot option code.

storeNumberOfOptions 8

8 - Number of Options for cmdStore, always last entry

Timer Entity

group cmdTIMER_Defines

Timer Entity Defines.

Defines

cmdTIMER 79

79 - Timer command code.

group cmdTIMER_Command_Options

Defines

timerExpiration 1

1 - Timer expiration option code.

timerMode 2

2 - Timer Mode option code.

timerModeSingle 0

0 - Single mode for timer mode option code.

timerModeRepeat 1

1 - Repeat mode for timer mode option code.

DefaultTimerMode 0

timerModeSingle - Default mode for timer mode option code.

Clock Entity

group cmdCLOCK_Defines

Clock entity 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.

USB Entity

group **cmdUSB_Defines**

USB entity 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

13 - SystemTemperature option code.

usbUpstreamModeAuto 2

2 - UpstreamMode Auto for upstream mode option code.

usbUpstreamModePort0 0

0 - UpstreamMode Port 0 for upstream mode option code.

usbUpstreamModePort1 1

1 - UpstreamMode Port 1 for upstream mode option code.

usbUpstreamModeNone 255

255 - UpstreamMode None to turn off all upstream connections.

usbUpstreamModeDefault 2

1 - UpstreamMode default for upstream mode option code.

usbUpstreamState 15

15 - UpstreamState option code.

usbUpstreamStateNone 2

2 - UpstreamMode Auto for upstream mode option code.

usbUpstreamStatePort0 0

0 - UpstreamMode Port 0 for upstream mode option code.

usbUpstreamStatePort1 1

1 - UpstreamMode Port 1 for upstream mode option code.

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.

usbDownstreamBoostMode 19

19 - Set/Get downstream boost mode.

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) The portMode bits follow and numbered according to their bit position. if they are set i.e. a 1 in the bit position the corresponding setting is enabled.

usbPortMode_sdp 0

0 - Standard Downstream port (0.5A max)

usbPortMode_cdp 1

1 - Charging Downstream port (5A max)

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
4 - USB2 dataline B side enabled

usbPortMode_VBusEnable 6
5 - USB VBUS enabled

usbPortMode_SuperSpeed1Enable 7
6 - USB SS Speed dataline side A enabled

usbPortMode_SuperSpeed2Enable 8
7 - USB SS Speed dataline side B enabled

usbPortMode_USB2BoostEnable 9
8 - USB2 Boost Mode Enabled

usbPortMode_USB3BoostEnable 10
9 - USB3 Boost Mode Enabled

usbPortMode_AutoConnectEnable 11
10 - Auto-connect Mode Enabled

usbPortMode_CC1Enable 12
11 - CC1 Enabled

usbPortMode_CC2Enable 13
12 - CC2 Enabled

usbPortMode_SBUEnable 14
13 - 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).

usbCC2Enable 28

28 - CC2 Disable option code (USB Type C).

usbSBUEnable 29

29 - SBU1/2 enable option code (USB Type C).

usbCC1Current 30

30 - CC1 get current option code (USB Type C).

usbCC2Current 31

31 - CC2 get current option code (USB Type C).

usbCC1Voltage 32

32 - CC1 get voltage option code (USB Type C).

usbCC2Voltage 33

33 - CC2 get voltage option code (USB Type C).

usbPortState 34

34 - TODO

usbPortError 35

35 - TODO

usbCableFlip 36

36 - TODO

usbAltMode 37

37 - USB Alt Mode configuration.

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

usbSBU1Voltage 38

38 - SBU1 get voltage option code (USB Type C).

usbSBU2Voltage 39

39 - SBU2 get voltage option code (USB Type C).

Upgrade command

cmdUPGRADE 95

Upgrade command.

Last command

cmdLAST 95

Last command.

3.4.10 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 cotents 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 occured 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.

3.4.11 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 epoch 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.

3.4.12 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.

3.4.13 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 9

Minor revision level of library.

Minor revisions should largely be compatible, however new features may be added with a minor revision change.

aVERSION_PATCH 29

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.

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 sotware 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.

3.4.14 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 *deviceList`, `uint32_t deviceListLength`, `uint32_t *devicesFound`)

Gets downstream device USB information for all Acroname hubs.

Parameters

- **deviceList** – Pointer to the start of a list/array to be used by the function.
- **deviceListLength** – Size of the list/array in `DeviceNode_t`'s, not bytes.)
- **devicesFound** – The number of `DeviceNode_t`'s that were populated.

Returns

`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.

3.5 .NET API Reference

Welcome to the BrainStem .NET API reference documentation. This documentation covers the .NET Acroname BrainStem library. 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.](#)

The Getting started guide is particularly useful for learning how to use the application tools we provide to communicate with your hardware.

3.5.1 Classes

class **AnalogClass**

The [AnalogClass](#) is the 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.

class **AppClass**

The [AppClass](#) is 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

class **ClockClass**

[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.

class **DigitalClass**

The [DigitalClass](#) is the 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.

class **EqualizerClass**

[EqualizerClass](#). Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

class **I2CClass**

The [I2CClass](#) is the interface the I2C busses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entitie's bus.

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.

class MuxClass

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 (multiplexor) 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).

class PointerClass

PointerClass. Access 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 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 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.

class RailClass

RailClass. Provides power rail functionality on certain modules. This entity is only available 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.

class RCServoClass

The *RCServoClass* is the 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.

class RelayClass

The *RelayClass* is the 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.

class SignalClass

SignalClass is the 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.

class StoreClass

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.

class SystemClass

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. The most common brainstem example uses the system entity to blink the User LED.

class TemperatureClass

TemperatureClass. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

class TimerClass

TimerClass. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

class UARTClass

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.

class USBCClass

USBCClass. 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.

3.5.2 Errors

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.

3.5.3 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 - Cancellation 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.

3.5.4 Analog Class

class AnalogClass

The *AnalogClass* is the 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 (Acroname::BrainStem::AnalogClass &analog)

Constructor.

~AnalogClass ()

Destructor.

!AnalogClass ()

Finalizer.

aErr **getValue** (unsigned short %value)

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.

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 **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 **getVoltage** (int %microvolts)

Get the scaled micro volt value with refrence 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 **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 **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 **setEnable** (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 **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 **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 **getConfiguration** (unsigned char %configuration)

Get the analog configuration.

Parameters

configuration – - Current configuration of the analog entity.

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.

Return values

aErrConfiguration – - Entity does not support this configuration.

Returns

EntityReturnValues “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 `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 `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 `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: (BRAINSTEM_RAM_SLOT_SIZE / 2) = (3FFF / 2) = 1FFF = 8191

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 unsigned char array with each sample stored in two consecutive bytes, LSB first.

Returns

Returns common entity return values. When the bulk capture is complete `getBulkCaptureState()` will return either `bulkCaptureFinished` or `bulkCaptureError`.

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

3.5.5 App Class

class **AppClass**

The *AppClass* is 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

Public Functions

AppClass (Acroname::BrainStem::AppClass &app)

Constructor.

~AppClass ()

Destructor.

!AppClass ()

Finalizer.

aErr **execute** (const unsigned int appParam)

Execute the app reflex on the module. Don'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.

Returns

aErr::aErrNone - success.

Returns

aErr::aErrTimeout - The request timed out waiting to start execution.

Returns

aErr::aErrConnection - No active link connection.

Returns

aErr::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. Wait 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.

Returns

aErr::aErrNone - success.

Returns

aErr::aErrTimeout - The request timed out waiting for a response.

Returns

aErr::aErrConnection - No active link connection.

Returns

aErr::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. Wait 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.

Returns

aErr::aErrNone - success.

Returns

aErr::aErrTimeout - The request timed out waiting for a response.

Returns

aErr::aErrConnection - No active link connection.

Returns

aErr::aErrNotFound - the app reflex was not found or not enabled on the module.

3.5.6 Clock Class

class **ClockClass**

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 (Acroname::BrainStem::ClockClass &clock)

Constructor.

~ClockClass ()

Destructor.

!ClockClass ()

Finalizer.

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)

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 **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 %min)

Get the two digit minute value (0-59).

Parameters

min – The two digit minute portion of the real-time clock value.

Returns

Returns common entity return values

aErr **setMinute** (const unsigned char min)

Set the two digit minute value (0-59).

Parameters

min – The two digit minute portion of the real-time clock value.

Returns

Returns common entity return values

aErr **getSecond** (unsigned char %sec)

Get the two digit second value (0-59).

Parameters

sec – The two digit second portion of the real-time clock value.

Returns

Returns common entity return values

aErr **setSecond** (const unsigned char sec)

Set the two digit second value (0-59).

Parameters

sec – The two digit second portion of the real-time clock value.

Returns

Returns common entity return values

3.5.7 Digital Class

class **DigitalClass**

The *DigitalClass* is the 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 (Acroname::BrainStem::DigitalClass &digital)

Constructor.

~DigitalClass ()

Destructor.

!DigitalClass ()

Finalizer.

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.

Parameters

state – 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.

Returns

Returns common entity return values

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** -

- Digital Input: digitalConfigurationInput = 0
- Digital Output: digitalConfigurationOutput = 1
- RCServo Input: FdigitalConfigurationRCServoInput = 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

Returns

aErr::aErrConfiguration - Entity does not support this configuration.

aErr **getConfiguration** (unsigned char %configuration)

Get the digital configuration.

Parameters

configuration - - Current configuration of the digital entity.

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

3.5.8 Equalizer Class

class **EqualizerClass**

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 (Acroname::BrainStem::*EqualizerClass* &equalizer)

Constructor.

~EqualizerClass (void)

Destructor.

!EqualizerClass (void)

Finalizer.

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.

3.5.9 I2C Class

class **I2CClass**

The *I2CClass* is the interface the I2C busses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entitie's bus.

Public Functions

I2CClass (Acroname::BrainStem::I2CClass &i2c)

Constructor.

~I2CClass ()

Destructor.

!I2CClass ()

Finalizer.

aErr read (const unsigned char address, const unsigned char length, unsigned char %result)

Read from a device on this I2C bus.

Parameters

- **address** -- The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** -- The length of the data to read in bytes.
- **result** -- 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 length, const unsigned char %data)

Write to a device on this I2C bus.

Parameters

- **address** -- The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** -- The length of the data to read in bytes.
- **data** -- 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 bool bEnable)

Set bus pullup state. This call only works with stems that have software controlled pullups. Check the datasheet for more information. This parameter is saved when system.save is called.

Parameters

bEnable -- true enables pullups 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

3.5.10 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 (Acroname::BrainStem::Module &module)

Constructor.

~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.

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.

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

Public Static Functions

static BrainStem2CLI::m_linkSpec **findFirstModule** (BrainStem2CLI::m_linkType type)

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** (BrainStem2CLI::m_linkType type,
unsigned int serialNum)

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** (BrainStem2CLI::m_linkType type,
List<m_linkSpec> ^specList)

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.

3.5.11 Mux Class

class **MuxClass**

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 (multiplexor) 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).

Public Functions

MuxClass (Acroname::BrainStem::MuxClass &rcservo)

Constructor.

~MuxClass ()

Destructor.

!MuxClass ()

Finalizer.

aErr getEnable (unsigned char %bEnabled)

Get the mux enable/disable status

Parameters

bEnabled – true: mux is enabled, false: the mux is disabled.

Returns

Returns common entity return values

aErr setEnable (const unsigned char bEnable)

Enable the mux.

Parameters

bEnable – 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 muxConfig_splitMode.

Returns

Returns common entity return values

aErr **getSplitMode** (int %splitMode)

Get the current mux mode.

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)

Set the current mux mode.

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

3.5.12 Pointer Class

class **PointerClass**

PointerClass. Access 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 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 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 and incrementing pointer.

Public Functions

PointerClass (Acroname::BrainStem::PointerClass &pointer)

Constructor.

~PointerClass ()

Destructor.

!PointerClass ()

Finalizer.

aErr **getOffset** (unsigned short %offset)

Get the offset of the pointer

Parameters

offset – The value of the offset.

Returns

All possible standard UEI return values.

aErr **setOffset** (unsigned short offset)

Set the offset of the pointer

Parameters

offset – The value of the offset.

Returns

All possible standard UEI 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

All possible standard UEI return values.

aErr **setMode** (unsigned char mode)

Set the mode of the pointer

Parameters

mode – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Returns

All possible standard UEI return values.

aErr **getTransferStore** (unsigned char %handle)

Get the handle to the store.

Parameters

handle – The handle of the store.

Returns

All possible standard UEI return handles.

aErr **setTransferStore** (unsigned char handle)

Set the handle to the store.

Parameters

handle – The handle of the store.

Returns

All possible standard UEI return handles.

aErr **initiateTransferToStore** (unsigned char length)

Transfer data to the store.

Parameters

length – The length of the data transfer.

Returns

All possible standard UEI return values.

aErr **initiateTransferFromStore** (unsigned char length)

Transfer data from the store.

Parameters

length – The length of the data transfer.

Returns

All possible standard UEI return values.

aErr **getChar** (unsigned char %value)

Get a char (1 unsigned char) value from the pointer at this object's index, where elements are 1 unsigned char long.

Parameters

value – The value of a single character (1 unsigned char) stored in the pointer.

Returns

All possible standard UEI return values.

aErr **setChar** (const unsigned char value)

Set a char (1 unsigned char) value to the pointer at this object's element index, where elements are 1 unsigned char long.

Parameters

value – The single char (1 unsigned char) value to be stored in the pointer.

Returns

All possible standard UEI return values.

aErr **getShort** (unsigned short %value)

Get a short (2 unsigned char) value from the pointer at this objects index, where elements are 2 bytes long

Parameters

value – The value of a single short (2 unsigned char) stored in the pointer.

Returns

All possible standard UEI 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 unsigned char) value to be set in the pointer.

Returns

All possible standard UEI 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 unsigned char) stored in the pointer.

Returns

All possible standard UEI 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 unsigned char) value to be stored in the pointer.

Returns

All possible standard UEI return values.

3.5.13 Port Class

class **PortClass**

Port Class 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 (Acroname::BrainStem::PortClass &port)

Constructor.

~PortClass ()

Destructor.

!PortClass ()

Finalizer.

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 setDataEnabled.

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 (SS) 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.

aErr **getDataSpeed** (unsigned char %speed)

Gets the speed of the enumerated device.

Parameters

speed – Bit mapped value representing the devices speed. See product datasheet 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 product datasheet 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 product datasheet 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 bufLength, 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
- **bufLength** – Length of the buffer to be filled

- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr **setName** (unsigned char %buffer, const unsigned int bufLength)

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.
- **bufLength** – Length of the buffer to be transferred.

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 **resetVbusAccumulatedPower** (void)

Reset the Vbus Accumulated Power

Returns

Returns common entity return values

aErr **getVconnAccumulatedPower** (int %milliwatthours)

Gets the Vconn Accumulated Power

Parameters

milliwatthours – The accumuled power on Vconn in milliwatt-hours.

Returns

Returns common entity return values

aErr **resetVconnAccumulatedPower** (void)

Reset the Vconn Accumulated Power

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 **resetEntityToFactoryDefaults** (void)

Resets the *PortClass* Entity to it factory default configuration.

Returns

Returns common entity return values

3.5.14 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 Acroname hub.

aErr **lastError**
Provides access to error code of update on construction.

3.5.15 Power Delivery Class

class **PowerDeliveryClass**

Power Delivery Class. 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 (Acroname::BrainStem::PowerDeliveryClass &pd)

Constructor.

~PowerDeliveryClass (void)

Destructor.

!PowerDeliveryClass ()

Finalizer.

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 bufLength, 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.
- **bufLength** – 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. 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 char partner, 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. 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
- **rdo** – Request Data Object to be set.

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 **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)

Returns

The returned error represents the success of the request being sent to the partner only. The success of the request being serviced by the remote partner can be obtained through [PowerDeliveryClass::requestStatus\(\)](#) 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 **resetEntityToFactoryDefaults** (void)

Resets the [PowerDeliveryClass](#) Entity to it factory default configuration.

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

3.5.16 Rail Class

class **RailClass**

RailClass. Provides power rail functionality on certain modules. This entity is only available on certain modules. The *RailClass* can be used to control power to downstream devices, I has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

Public Functions

RailClass (Acroname::BrainStem::*RailClass* &pointer)

Constructor.

~RailClass ()

Destructor.

!RailClass ()

Finalizer.

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 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) 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 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 %bEnable)

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

bEnable – 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 bEnable)

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

bEnable – 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 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) to be supply 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 %milliwatts)

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

milliwatts – The power in milli-watts (1 == 1e-3W) 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 milliwatts)

Set the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Parameters

milliwatts – The power in milli-watts (1 == 1e-3W) to be supplied by the rail.

Returns

Returns common entity return values

aErr getPowerSetpoint (int %milliwatts)

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

milliwatts – The power in milli-watts (1 == 1e-3W) 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 to determine if this is a measured or stored value.

Returns

Returns common entity return values

aErr setPowerLimit (const int milliwatts)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

milliwatts – The power in milli-watts (1 == 1e-3W).

Returns

Returns common entity return values

aErr getPowerLimit (int %milliwatts)

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

milliwatts – The power in milli-watts (1 == 1e-3W).

Returns

Returns common entity return values

aErr getResistance (int %milliohms)

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

milliohms – The resistance in milli-ohms (1 == 1e-3Ohms) 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 to determine if this is a measured or stored value.

Returns

Returns common entity return values

aErr setResistanceSetpoint (const int milliohms)

Set the rail setpoint resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Parameters

milliohms – The power in milli-ohms (1 == 1e-3Ohms) to be drawn by the rail.

Returns

Returns common entity return values

aErr `getResistanceSetpoint` (int %milliohms)

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

milliohms – The resistance in milli-ohms (1 == 1e-3Ohms) 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 bEnable)

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

bEnable – enable or disable kelvin sensing.

Returns

Returns common entity return values

aErr `getKelvinSensingEnable` (unsigned char %bEnable)

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

bEnable – 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.

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

3.5.17 RCServo Class

class **RCServoClass**

The *RCServoClass* is the 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 (Acroname::BrainStem::RCServoClass &rcservo)

Constructor.

~RCServoClass ()

Destructor.

!RCServoClass ()

Finalizer.

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 **getEnable** (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 1 ms 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”. ie. 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

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

3.5.18 Relay Class

class **RelayClass**

The *RelayClass* is the 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 (Acroname::BrainStem::RelayClass &relay)

Constructor.

~RelayClass ()

Destructor.

!RelayClass ()

Finalizer.

aErr **setEnabled** (const unsigned char bEnable)

Set the enable/disable state.

Parameters

bEnable – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns common entity return values

aErr **getEnable** (unsigned char %bEnabled)

Get the state.

Parameters

bEnabled – 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 refrence 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 refrence voltages.

Returns

Returns common entity return values

3.5.19 Signal Class

See the *Signal Entity* for generic information.

class **SignalClass**

SignalClass is the 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 (Acroname::BrainStem::SignalClass &signal)

Constructor.

~SignalClass ()

Destructor.

!SignalClass ()

Finalizer.

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 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 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

3.5.20 Store Class**class StoreClass**

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** (Acroname::BrainStem::StoreClass &store)

Constructor.

~StoreClass ()

Destructor.

!StoreClass ()

Finalizer.

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 %pData, const unsigned short length)

Load the slot.

Parameters

- **slot** – The slot number.
- **pData** – The data.
- **length** – The data length.

Returns

Returns common entity return values

aErr **unloadSlot** (const unsigned char slot, const unsigned int dataLength, unsigned char %pData, unsigned int %unloadedLength)

Unload the slot data.

Parameters

- **pData** – Byte array that the unloaded data will be placed into.
- **dataLength** – - The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.

- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than **dataLength**.
- **slot** – The slot number.

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.

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

Parameters

- **slot** – The slot number.
- **size** – The slot size.

Returns

Returns common entity return values

3.5.21 System Class

class **SystemClass**

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. The most common brainstem example uses the system entity to blink the User LED.

Public Functions

SystemClass (Acroname::BrainStem::SystemClass &system)

Constructor.

~SystemClass ()

Destructor.

!SystemClass ()

Finalizer.

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 bOn)

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

bOn – true: turn the LED on, false: turn LED off.

Returns

Returns common entity return values

aErr **getLED** (unsigned char %bOn)

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

bOn – true: LED on, false: LED off.

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.

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 32bit 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.

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** (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 in micro-C

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 in micro-C

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 bOn)

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

bOn – Enable or disable of the route to me function 1 = enable.

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

3.5.22 Temperature Class

class **TemperatureClass**

TemperatureClass. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

Public Functions

TemperatureClass (Acroname::BrainStem::TemperatureClass &temperature)

Constructor.

~TemperatureClass ()

Destructor.

!TemperatureClass ()

Finalizer.

aErr **getValue** (int %microcelsius)

Get the temperature.

Parameters

microcelsius – The temperature in micro-Celsius (1 == 1e-6C).

Returns

Returns common entity return values

aErr **getValueMin** (int %minTemperature)

Get the module's minimum temperature in micro-C

Parameters

minTemperature – The module's minimum system temperature in micro-C

Returns

Returns common entity return values

aErr **getValueMax** (int %maxTemperature)

Get the module's maximum temperature in micro-C

Parameters

maxTemperature – The module's maximum system temperature in micro-C

Returns

Returns common entity return values

3.5.23 Timer Class

class **TimerClass**

TimerClass. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

Public Functions

TimerClass (Acroname::BrainStem:: *TimerClass* &timer)

Constructor.

~TimerClass ()

Destructor.

!TimerClass ()

Finalizer.

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 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

Returns

aErr::aErrNone - Action completed successfully.

3.5.24 UART Class

class **UARTClass**

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 (Acroname::BrainStem::UARTClass &uart)

Constructor.

~UARTClass ()

Destructor.

!UARTClass ()

Finalizer.

aErr **setEnabled** (const unsigned char bEnabled)

Enable the UART channel.

Parameters

bEnabled – true: enabled, false: disabled.

Returns

Returns common entity return values

aErr **getEnable** (unsigned char %bEnabled)

Get the UART channel state.

Parameters

bEnabled – true: enabled, false: disabled.

Returns

Returns common entity return values

aErr **setBaudRate** (const unsigned int rate)

Set the UART baud rate.

Parameters

rate – baud rate.

Returns

Returns common entity return values

aErr **getBaudRate** (unsigned int %rate)

Get the UART baud rate.

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 – Serial protocol.

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

3.5.25 USB Class

class **USBClass**

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 (Acroname::BrainStem::USBClass &USB)

Constructor.

~USBClass ()

Destructor.

!USBClass ()

Finalizer.

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, Super-Speed (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 %state)

Get a bit mapped representation of the hub mode; see the product datasheet for state mapping. Usually represents the hub's downstream ports data and power line enable/disable state.

Parameters

state – The USB hub state.

Returns

Returns common entity return values

aErr **setHubMode** (const unsigned int state)

Set a bit mapped hub state; see the product datasheet for state mapping. Usually represents the hub's downstream ports data and power line enable/disable state.

Parameters

state – The USB hub state.

Returns

Returns common entity return values

aErr **clearPortErrorStatus** (const unsigned char channel)

Clear the error status for the given channel.

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 and usbUpstreamModePort1

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 interport enumeration delay in milliseconds. This setting should be saved with a stem.system.save() call.

Parameters

ms_delay – 100ms delay increment.

Returns

Returns common entity return values

aErr **getEnumerationDelay** (unsigned int %ms_delay)

Get the interport enumeration delay.

Parameters

ms_delay – 100ms delay increment.

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. This reflects the limit setting currently. in effect.

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 datasheet for a complete list of capabilities. There is a unified bit mapping for port mode at usbPortMode

Parameters

- **channel** – USB downstream channel to set the mode on.
- **mode** – The port mode setting as packet bit mask.

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 datasheet for a complete list of capabilities. There is a unified bit mapping for port mode at 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 bEnable)

Set Enable/Disable on the CC1 line.

Parameters

- **channel** – - USB channel.
- **bEnable** –
 - 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** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

aErr **setCC2Enable** (const unsigned char channel, const unsigned char bEnable)

Set Enable/Disable on the CC2 line.

Parameters

- **channel** – - USB channel.
- **bEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

aErr **getCC2Enable** (const unsigned char channel, unsigned char %pEnable)

Get Enable/Disable on the CC1 line.

Parameters

- **channel** – - USB channel.
- **pEnable** –
 - 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 bEnable)

Enable/Disable the only the SBU1/2 based on the configuration of the usbPortMode settings.

Parameters

- **channel** – The USB sub channel.
- **bEnable** –
 - 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 bEnable)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters

- **channel** – The USB sub channel.
- **bEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

aErr **getCableFlip** (const unsigned char channel, unsigned char %pEnable)

Get Cable 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

3.5.26 USBSystem Class

class **USBSystemClass**

USBSystem Class The USBSystem class provides high level control of the lower level Port Class.

Public Functions

USBSystemClass (Acroname::BrainStem::USBSystemClass &usb)

Constructor.

~USBSystemClass ()

Destructor.

!USBSystemClass ()

Finalizer.

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. This setting should be saved with a `stem.system.save()` call. 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 *Port-Class::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 *Port-Class::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 *Port-Class::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 bufLength, 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
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr **setModeList** (unsigned int %buffer, const unsigned int bufLength)

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.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

aErr **getStateList** (unsigned int %buffer, const unsigned int bufLength, 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
- **bufLength** – 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 bufLength, 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
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr **setPowerBehaviorConfig** (unsigned int %buffer, const unsigned int bufLength)

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.
- **bufLength** – 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 bufLength, 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
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

aErr **setDataRoleBehaviorConfig** (unsigned int %buffer, const unsigned int bufLength)

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.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

aErr **resetEntityToFactoryDefaults** (void)

Resets the *USBSYSTEMCLASS* Entity to it factory default configuration.

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
- **mode** – Mode to be set.

Returns

Returns common entity return values Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Returns

Returns common entity return values

3.6 LabVIEW API Reference

Welcome to the BrainStem LabVIEW API reference documentation. This documentation covers the LabVIEW Acroname BrainStem library. 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.*](#)

The Getting started guide is particularly useful for learning how to use the application tools we provide to communicate with your hardware.

3.6.1 Entities

group **AnalogEntity**

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.

group **AppEntity**

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.

group **ClockEntity**

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.

group **DigitalEntity**

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.

group **EqualizerEntity**

EqualizerClass. Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

group **I2CEntity**

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.

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.

group MuxEntity

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 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).

group PointerEntity

PointerClass. Access 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 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 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 and incrementing pointer.

group RailEntity

RailClass. Provides power rail functionality on certain modules. This entity is only available 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.

group RCServoEntity

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.

group RelayEntity

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.

group SignalEntity

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.

group StoreEntity

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.

group SystemEntity

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. The most common brainstem example uses the system entity to blink the User LED.

group TemperatureEntity

TemperatureClass. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

group TimerEntity

TimerClass. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

group UARTEntity

UART Class. 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.

group USBEntity

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.

3.6.2 Analog Entity

group AnalogEntity

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.

void **analog_getValue** (unsigned int *id, struct Result *result, const int index)

Get the raw ADC output value in bits.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **analog_getVoltage** (unsigned int *id, struct Result *result, const int index)

Get the scaled micro volt value with reference to ground.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **analog_getRange** (unsigned int *id, struct Result *result, const int index)

Get the analog input range.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **analog_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the analog output enable status.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **analog_setValue** (unsigned int *id, struct Result *result, const int index, const int 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

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.

Note: Voltage range is dependent on the specific DAC channel range.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns common entity return values

void **analog_setRange** (unsigned int *id, struct Result *result, const int index, const unsigned char range)
Set the analog input range.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **range** – 8 bit value corresponding to a discrete range option

Returns

Returns common entity return values

void **analog_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)
Set the analog output enable state.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – set 1 to enable or 0 to disable.

Returns

Returns common entity return values

void **analog_setConfiguration** (unsigned int *id, struct Result *result, const int index, const unsigned char configuration)

Set the analog configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **configuration** -- bitAnalogConfigurationOutput configures the analog entity as an output.

Return values

aErrConfiguration -- Entity does not support this configuration.

Returns

EntityReturnValues “common entity” return values

void **analog_getConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the analog configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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

Returns

Returns common entity return values

void **analog_getBulkCaptureSampleRate** (unsigned int *id, struct Result *result, const int index)

Get the current sample rate setting for this analog when bulk capturing.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **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

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values. When the bulk capture is complete getBulkCaptureState() will return either bulkCaptureFinished or bulkCaptureError.

void **analog_getBulkCaptureState** (unsigned int *id, struct Result *result, const int index)

Get the current bulk capture state for this analog.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.3 App Entity

group **AppEntity**

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.

void **app_execute** (unsigned int *id, struct Result *result, const int index, const unsigned int appParam)

Execute the app reflex on the module. Don't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **appParam** – The app parameter handed to the reflex.

Returns

::aErrNone - success.

Returns

::aErrTimeout - The request timed out waiting to start execution.

Returns

::aErrConnection - No active link connection.

Returns

::aErrNotFound - the app reflex was not found or not enabled on the module.

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. Wait for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **returnValue** – 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.

Returns

::aErrNone - success.

Returns

::aErrTimeout - The request timed out waiting for a response.

Returns

::aErrConnection - No active link connection.

Returns

::aErrNotFound - the app reflex was not found or not enabled on the module.

3.6.4 Clock Entity

group **ClockEntity**

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.

void **clock_getYear** (unsigned int *id, struct Result *result, const int index)

Get the four digit year value (0-4095).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **clock_setYear** (unsigned int *id, struct Result *result, const int index, const int year)

Set the four digit year value (0-4095).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **year** – Set the year portion of the real-time clock value.

Returns

Returns common entity return values

void **clock_getMonth** (unsigned int *id, struct Result *result, const int index)

Get the two digit month value (1-12).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **clock_setMonth** (unsigned int *id, struct Result *result, const int index, const unsigned char month)
Set the two digit month value (1-12).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **month** – The two digit month portion of the real-time clock value.

Returns

Returns common entity return values

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **day** – The two digit day portion of the real-time clock value.

Returns

Returns common entity return values

void **clock_getHour** (unsigned int *id, struct Result *result, const int index)
Get the two digit hour value (0-23).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **clock_setHour** (unsigned int *id, struct Result *result, const int index, const unsigned char hour)
Set the two digit hour value (0-23).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **hour** – The two digit hour portion of the real-time clock value.

Returns

Returns common entity return values

void **clock_getMinute** (unsigned int *id, struct Result *result, const int index)

Get the two digit minute value (0-59).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **clock_setMinute** (unsigned int *id, struct Result *result, const int index, const unsigned char min)

Set the two digit minute value (0-59).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **min** – The two digit minute portion of the real-time clock value.

Returns

Returns common entity return values

void **clock_getSecond** (unsigned int *id, struct Result *result, const int index)

Get the two digit second value (0-59).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **clock_setSecond** (unsigned int *id, struct Result *result, const int index, const unsigned char sec)

Set the two digit second value (0-59).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **sec** – The two digit second portion of the real-time clock value.

Returns

Returns common entity return values

3.6.5 Digital Entity

group **DigitalEntity**

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.

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. Note: Some configurations are only supported on specific pins.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **configuration** –
 - 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

Returns

::aErrConfiguration - Entity does not support this configuration.

void **digital_getConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the digital configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **digital_setState** (unsigned int *id, struct Result *result, const int index, const unsigned char state)
Set the logical state.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **state** – The state to be set. 0 is logic low, 1 is logic high.

Returns

Returns common entity return values

void **digital_getState** (unsigned int *id, struct Result *result, const int index)
Get the state.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.6 Equalizer Entity

group **EqualizerEntity**

EqualizerClass. 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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The equalizer receiver channel.
- **config** – Configuration to be applied to the receiver.

Returns

Returns common entity return values.

void **equalizer_getReceiverConfig** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Gets the receiver configuration for a given channel.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **config** – Configuration of the receiver.

Returns

Returns common entity return values.

void **equalizer_setTransmitterConfig** (unsigned int *id, struct Result *result, const int index, const unsigned char config)

Sets the transmitter configuration

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **config** – Configuration to be applied to the transmitter.

Returns

Returns common entity return values.

void **equalizer_getTransmitterConfig** (unsigned int *id, struct Result *result, const int index)

Gets the transmitter configuration

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values.

3.6.7 I2C Entity

group **I2CEntity**

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.

void **i2c_read** (unsigned int *id, struct Result *result, const int index, const int address, const int bufferLength, unsigned char *buffer)

Read from a device on this I2C bus.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **address** – - The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** – - The length of the data to read in bytes.
- **result** – - 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

void **i2c_write** (unsigned int *id, struct Result *result, const int index, const int address, const int bufferLength, unsigned char *buffer)

Write to a device on this I2C bus.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **address** – - The I2C address (7bit <XXXX-XXX0>) of the device to write.
- **length** – - The length of the data to write in bytes.
- **data** – - The data to send to the device, This array should be no larger than aBRAINSTEM_MAXPACKETBYTES - 5

Returns

Returns common entity return values

void **i2c_setPullup** (unsigned int *id, struct Result *result, const int index, const bool bEnable)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – - true enables pull-ups false disables them.

Returns

Returns common entity return values

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **speed** – - The speed setting value.

Returns

Returns common entity return values

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.8 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 int sn = 0)

Creates a generic BrainStem/stem object to be used in the program

Parameters

- **id** – This value will be assigned by the underlying library if a serial number is not provided and will be replaced by the devices serial number once a connection is made.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **sn** – Serial number of the device you want to create. Not required.

void **module_disconnectAndDestoryStem** (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** – The id assigned by the create stem vi.
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_discoverAndConnect** (unsigned int *id, struct Result *result, int transport = (int)USB)

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** – The id assigned by the create stem vi.
- **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 deviceListItem *stemList, int *stemListLength, int transport = (int)USB)

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** – The id assigned by the create stem vi.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **stemList** – List of device specifiers for each of the discovered devices
- **stemListLength** – Indicates how long the list is.
- **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** – The id assigned by the create stem vi.
- **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** – The id assigned by the create stem vi.
- **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** – The id assigned by the create stem vi.
- **id_linkStem** – The link stem’s id assigned by the create stem vi. (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** – The id assigned by the create stem vi.
- **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** – The id assigned by the create stem vi.
- **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** – The id assigned by the create stem vi.

- **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** – The id assigned by the create stem vi.
- **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

3.6.9 Mux Entity

group MuxEntity

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).

void **mux_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the mux enable/disable status

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **mux_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char bEnable)

Enable the mux.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

- **index** – The index of the entity in question.
- **bEnable** – true: enables the mux for the selected channel.

Returns

Returns common entity return values

void **mux_getChannel** (unsigned int *id, struct Result *result, const int index)

Get the current selected mux channel.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **mux_setChannel** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Set the current mux channel.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – mux channel to select.

Returns

Returns common entity return values

void **mux_getChannelVoltage** (unsigned int *id, struct Result *result, const int index, const unsigned char channel)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

Returns

Returns common entity return values

void **mux_getConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the configuration of the mux.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **mux_setConfiguration** (unsigned int *id, struct Result *result, const int index, const int config)

Set the configuration of the mux.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **config** – integer representing the mux configuration either muxConfig_default, or muxConfig_splitMode.

Returns

Returns common entity return values

void **mux_getSplitMode** (unsigned int *id, struct Result *result, const int index)

Get the current split mode mux configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **mux_setSplitMode** (unsigned int *id, struct Result *result, const int index, const int splitMode)

Sets the mux's split mode configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

3.6.10 Pointer Entity

group **PointerEntity**

PointerClass. Access 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 and incrementing pointer.

void **pointer_getOffset** (unsigned int *id, struct Result *result, const int index)

Get the offset of the pointer

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

All possible standard UEI return values.

void **pointer_setOffset** (unsigned int *id, struct Result *result, const int index, int offset)

Set the offset of the pointer

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **offset** – The value of the offset.

Returns

All possible standard UEI return values.

void **pointer_getMode** (unsigned int *id, struct Result *result, const int index)

Get the mode of the pointer

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

All possible standard UEI return values.

void **pointer_setMode** (unsigned int *id, struct Result *result, const int index, unsigned char mode)

Set the mode of the pointer

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

Returns

All possible standard UEI return values.

void **pointer_getTransferStore** (unsigned int *id, struct Result *result, const int index)

Get the handle to the store.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

All possible standard UEI return handles.

void **pointer_setTransferStore** (unsigned int *id, struct Result *result, const int index, unsigned char handle)

Set the handle to the store.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **handle** – The handle of the store.

Returns

All possible standard UEI return handles.

void **pointer_initiateTransferToStore** (unsigned int *id, struct Result *result, const int index, unsigned char length)

Transfer data to the store.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **length** – The length of the data transfer.

Returns

All possible standard UEI return values.

void **pointer_initiateTransferFromStore** (unsigned int *id, struct Result *result, const int index, unsigned char length)

Transfer data from the store.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **length** – The length of the data transfer.

Returns

All possible standard UEI return values.

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

All possible standard UEI return values.

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – The single char (1 byte) value to be stored in the pointer.

Returns

All possible standard UEI return values.

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

All possible standard UEI return values.

void **pointer_setShort** (unsigned int *id, struct Result *result, const int index, const int value)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – The single short (2 byte) value to be set in the pointer.

Returns

All possible standard UEI return values.

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

All possible standard UEI return values.

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – The single int (4 byte) value to be stored in the pointer.

Returns

All possible standard UEI return values.

3.6.11 Port Entity

group PortEntity

Port Class 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

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getVbusCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the Vbus Current

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getVconnVoltage** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Voltage

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getVconnCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Current

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getPowerMode** (unsigned int *id, struct Result *result, const int index)

Gets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerMode** – The power mode to be set.

Returns

Returns common entity return values

void **port_getEnabled** (unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_setEnabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the entire port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Fully enable port; 0 = Fully disable port.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_setDataHS1Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Hight Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_setDataHS2Enabled** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Hight Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

Returns

Returns common entity return values

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 get-DataSSEnabled.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_setDataSS1Enabled** (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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable power; 0 = Disable disable.

Returns

Returns common entity return values

void **port_getDataRole** (unsigned int *id, struct Result *result, const int index)

Gets the Port Data Role.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn lines; 0 = Disable Vconn lines.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn1 lines; 0 = Disable Vconn1 lines.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn2 lines; 0 = Disable Vconn2 lines.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC lines; 0 = Disable CC lines.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC1 lines; 0 = Disable CC1 lines.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC2 lines; 0 = Disable CC2 lines.

Returns

Returns common entity return values

void **port_getVoltageSetpoint** (unsigned int *id, struct Result *result, const int index)

Gets the current voltage setpoint value for the port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – the voltage setpoint of the port in uV.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getMode** (unsigned int *id, struct Result *result, const int index)

Gets current mode of the port

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_setMode** (unsigned int *id, struct Result *result, const int index, const unsigned int mode)

Sets the mode of the port

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – Port mode to be set. See product datasheet for details.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getCurrentLimit** (unsigned int *id, struct Result *result, const int index)

Gets the current limit of the port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_setCurrentLimit** (unsigned int *id, struct Result *result, const int index, const unsigned int limit)

Sets the current limit of the port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **limit** – Current limit to be applied in microAmps (uA).

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getPowerLimit** (unsigned int *id, struct Result *result, const int index)

Gets the user defined power limit for the port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **limit** – Power limit to be applied in milli-watts (mW).

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

void **port_getName** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const int bufferSize)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

void **port_setName** (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const int bufferSize)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

void **port_getVbusAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the Vbus Accumulated Power

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_resetVbusAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Resets the Vbus Accumulated Power to zero.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_getVconnAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Accumulated Power

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_resetVconnAccumulatedPower** (unsigned int *id, struct Result *result, const int index)

Resets the Vconn Accumulated Power to zero.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **port_resetEntityToFactoryDefaults** (unsigned int *id, struct Result *result, const int index)

Resets the PortClass Entity to it factory default configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.12 PowerDelivery Entity

group **PowerDeliveryEntity**

Power Delivery Class. 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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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
- **numRules** – Variable to be filled with the number of PDOs.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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** – Variable to be filled with the requested power rule.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

void **powerdelivery_getPowerDataObjectList** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const int bufferLength)

Gets all Power Data Objects (PDOs). Equivalent to calling PowerDeliveryClass::getPowerDataObject() on all partners, power roles, and index's.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.
- **bufLength** – 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

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **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

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **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

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. 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

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **rdo** – Variable to be filled with the current RDO. Zero indicates the RDO is not active.

Returns

Returns common entity return values

void **powerdelivery_setRequestDataObject** (unsigned int *id, struct Result *result, const int index, const unsigned char partner, 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. 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **partner** – Indicates which side of the PD connection is in question.
 - Local = 0 = powerdeliveryPartnerLocal
- **rdo** – Request Data Object to be set.

Returns

Returns common entity return values

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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)

Returns

Returns common entity return values

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Value to be applied.
 - Disabled = 0 = powerdeliveryPowerRoleDisabled
 - Source = 1 = powerdeliveryPowerRoleSource
 - Sink = 2 = powerdeliveryPowerRoleSink

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **powerdelivery_getCableOrientation** (unsigned int *id, struct Result *result, const int index)

Gets the current orientation being used for PD communication

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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)

Returns

The returned error represents the success of the request being sent to the partner only. The success of the request being serviced by the remote partner can be obtained through `PowerDeliveryClass::requestStatus()` Returns common entity return values

void **powerdelivery_requestStatus** (unsigned int *id, struct Result *result, const int index)

Gets the status of the last request command sent.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **powerdelivery_getOverride** (unsigned int *id, struct Result *result, const int index)

Gets the current enabled overrides

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **powerdelivery_setOverride** (unsigned int *id, struct Result *result, const int index, const unsigned int overrides)

Sets the current enabled overrides

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **overrides** – Overrides to be set in a bit mapped representation.

Returns

Returns common entity return values

void **powerdelivery_resetEntityToFactoryDefaults** (unsigned int *id, struct Result *result, const int index)

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – Variable to be filled with the current mode.
 - Disabled (0)
 - Enabled (1)
 - Auto (2) default

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **configuration** – An enumerated value referring to the configuration to be set
 - Allowable values are 0 - 4

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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)

Returns

Returns common entity return values

void **powerdelivery_packDataObjectAttributes** (unsigned int *id, struct Result *result, const int index, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **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

aErrNone on success; aErrParam with bad input.

void **powerdelivery_unpackDataObjectAttributes** (unsigned int *id, struct Result *result, const int index, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **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

aErrNone on success; aErrParam with bad input.

3.6.13 Rail Entity

group **RailEntity**

RailClass. Provides power rail functionality on certain modules. This entity is only available on certain modules. The RailClass can be used to control power to downstream devices, I 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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microamps** – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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)

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microamps** – The current in micro-amps (1 == 1e-6A).

Returns

Returns common entity return values

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)

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **rail_getTemperature** (unsigned int *id, struct Result *result, const int index)

Get the rail temperature.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **rail_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char bEnable)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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)

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns common entity return values

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)

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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)

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V).

Returns

Returns common entity return values

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)

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **rail_setPowerSetpoint** (unsigned int *id, struct Result *result, const int index, const int milliwatts)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **milliwatts** – The power in milli-watts (1 == 1e-3W) to be supplied by the rail.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **rail_setPowerLimit** (unsigned int *id, struct Result *result, const int index, const int milliwatts)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

- **index** – The index of the entity in question.
- **milliwatts** – The power in milli-watts (mW).

Returns

Returns common entity return values

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)

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **rail_setResistanceSetpoint** (unsigned int *id, struct Result *result, const int index, const int milliohms)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **milliohms** – The resistance in milli-ohms (1 == 1e-3Ohms) to be drawn by the rail.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **rail_setKelvinSensingEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char bEnable)

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – enable or disable kelvin sensing.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – The operational mode to employ.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.14 RCServo Entity

group **RCServoEntity**

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.

void **rcservo_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enable the servo channel

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – The state to be set. 0 is disabled, 1 is enabled.

Returns

Returns common entity return values

void **rcservo_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the enable status of the servo channel.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **rcservo_setPosition** (unsigned int *id, struct Result *result, const int index, const unsigned char position)

Set the position of the servo channel

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **position** – The position to be set. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns

Returns common entity return values

void **rcservo_getPosition** (unsigned int *id, struct Result *result, const int index)

Get the position of the servo channel

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **reverse** – Reverses the value set by “setPosition”. ie. 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

void **rcservo_getReverse** (unsigned int *id, struct Result *result, const int index)

Get the reverse status of the servo channel

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.15 Relay Entity

group RelayEntity

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.

void **relay_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char bEnable)

Set the enable/disable state.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns common entity return values

void **relay_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the state.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **relay_getVoltage** (unsigned int *id, struct Result *result, const int index)

Get the scaled micro volt value with reference to ground.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.16 Signal Entity

group **SignalEntity**

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.

void **signal_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enable/Disable the signal output.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – True to enable, false to disable

Returns

Returns common entity return values

void **signal_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the Enable/Disable of the signal.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **invert** – to invert, false for normal mode.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

void **signal_getT3Time** (unsigned int *id, struct Result *result, const int index)

Get the signal period or T3 in nanoseconds.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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.

Returns

Returns common entity return values

void **signal_getT2Time** (unsigned int *id, struct Result *result, const int index)

Get the signal active period or T2 in nanoseconds.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.17 Store Entity

group StoreEntity

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.

void **store_getSlotState** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Get slot state.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **state** – true: enabled, false: disabled.

Returns

Returns common entity return values

void **store_loadSlot** (unsigned int *id, struct Result *result, const int index, const int slot, unsigned char *buffer, const int bufferLength)

Load the slot.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **pData** – The data.
- **length** – The data length.

Returns

Returns common entity return values

void **store_unloadSlot** (unsigned int *id, struct Result *result, const int index, const int slot, unsigned char *buffer, const int bufferLength)

Unload the slot data.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **pData** – Byte array that the unloaded data will be placed into.
- **dataLength** – - The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than dataLength.
- **slot** – The slot number.

Returns

Returns common entity return values

void **store_slotEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)
Enable slot.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.

Returns

Returns common entity return values

void **store_slotDisable** (unsigned int *id, struct Result *result, const int index, const unsigned char slot)
Disable slot.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **capacity** – The slot capacity.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **size** – The slot size.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **lock** – Variable to be filed with the locked state.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number
- **lock** – state to be set.

Returns

Returns common entity return values

3.6.18 System Entity

group **SystemEntity**

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. The most common brainstem example uses the system entity to blink the User LED.

void **system_getModule** (unsigned int *id, struct Result *result)

Get the current address the module uses on the BrainStem network.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getModuleBaseAddress** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setRouter** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **address** – The router address to be used.

Returns

Returns common entity return values

void **system_getRouter** (unsigned int *id, struct Result *result)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setHBInterval** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **interval** – The desired heartbeat delay.

Returns

Returns common entity return values

void **system_getHBInterval** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setLED** (unsigned int *id, struct Result *result, const unsigned char bOn)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **bOn** – true: turn the LED on, false: turn LED off.

Returns

Returns common entity return values

void **system_getLED** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setBootSlot** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **slot** – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

Returns

Returns common entity return values

void **system_getBootSlot** (unsigned int *id, struct Result *result)

Get the store slot which is mapped when the module boots.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getVersion** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

void **system_getModel** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getHardwareVersion** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getSerialNumber** (unsigned int *id, struct Result *result)

Get the module's serial number. The serial number is a unique 32bit integer which is usually communicated in hexadecimal format.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_save** (unsigned int *id, struct Result *result)

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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_reset** (unsigned int *id, struct Result *result)

Reset the system.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_logEvents** (unsigned int *id, struct Result *result)

Saves system log events to a slot defined by the module (usually ram slot 0).

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getUptime** (unsigned int *id, struct Result *result)

Get the module's accumulated uptime in minutes

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getTemperature** (unsigned int *id, struct Result *result)

Get the module's current temperature in micro-C

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getMinimumTemperature** (unsigned int *id, struct Result *result)

Get the module's minimum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getMaximumTemperature** (unsigned int *id, struct Result *result)

Get the module's maximum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getInputVoltage** (unsigned int *id, struct Result *result)

Get the module's input voltage.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getInputCurrent** (unsigned int *id, struct Result *result)

Get the module's input current.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getModuleHardwareOffset** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setModuleSoftwareOffset** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **address** – The address for the module. Value must be even from 0-254.

Returns

Returns common entity return values

void **system_getModuleSoftwareOffset** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getRouterAddressSetting** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_routeToMe** (unsigned int *id, struct Result *result, const unsigned char bOn)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **bOn** – Enable or disable of the route to me function 1 = enable.

Returns

Returns common entity return values

void **system_getPowerLimit** (unsigned int *id, struct Result *result)

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

void **system_getPowerLimitMax** (unsigned int *id, struct Result *result)

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setPowerLimitMax** (unsigned int *id, struct Result *result, const unsigned int power)

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **power** – Limit in milli-Watts (mW) to be set.

Returns

Returns common entity return values

void **system_getPowerLimitState** (unsigned int *id, struct Result *result)

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit().

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getUnregulatedVoltage** (unsigned int *id, struct Result *result)

Gets the voltage present at the unregulated port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getUnregulatedCurrent** (unsigned int *id, struct Result *result)

Gets the current passing through the unregulated port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getInputPowerSource** (unsigned int *id, struct Result *result)

Provides the source of the current power source in use.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getInputPowerBehavior** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setInputPowerBehavior** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **behavior** – An enumerated representation of behavior to be set.

Returns

Returns common entity return values

void **system_getInputPowerBehaviorConfig** (unsigned int *id, struct Result *result, unsigned int *buffer, const int bufferSize)

Gets the input power behavior configuration. Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

void **system_setInputPowerBehaviorConfig** (unsigned int *id, struct Result *result, const unsigned int bufLength)

Sets the input power behavior configuration. Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

void **system_getName** (unsigned int *id, struct Result *result, unsigned char *buffer, const int bufferSize)

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

void **system_setName** (unsigned int *id, struct Result *result, unsigned char *buffer, const int bufferSize)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

void **system_resetEntityToFactoryDefaults** (unsigned int *id, struct Result *result)

Resets the SystemClass Entity to it factory default configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_resetDeviceToFactoryDefaults** (unsigned int *id, struct Result *result)

Resets the device to it factory default configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_getLinkInterface** (unsigned int *id, struct Result *result)

Gets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **system_setLinkInterface** (unsigned int *id, struct Result *result, const unsigned char linkInterface)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **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

3.6.19 Temperature Entity

group **TemperatureEntity**

TemperatureClass. 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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **temperature_resetEntityToFactoryDefaults** (unsigned int *id, struct Result *result, const int index)

3.6.20 Timer Entity

group **TimerEntity**

TimerClass. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **usecDuration** – The duration before timer expiration in microseconds.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **timer_setMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Set the mode of the timer which is either single or repeat mode.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – The mode of the timer. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Returns

Returns common entity return values

Returns

::aErrNone - Action completed successfully.

3.6.21 UART Entity

group **UARTEntity**

UART Class. 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.

void **uart_setEnable** (unsigned int *id, struct Result *result, const int index, const unsigned char bEnable)

Enable the UART channel.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – False or 0 = Disabled, True or 1 = Enabled

Returns

Returns common entity return values

void **uart_getEnable** (unsigned int *id, struct Result *result, const int index)

Get the UART channel state.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

3.6.22 USB Entity

group **USBEntity**

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.

void **usb_setPortEnable** (unsigned int *id, struct Result *result, const unsigned char channel)

Enable both power and data lines for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setPortDisable** (unsigned int *id, struct Result *result, const unsigned char channel)

Disable both power and data lines for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setDataEnable** (unsigned int *id, struct Result *result, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setDataDisable** (unsigned int *id, struct Result *result, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setHiSpeedDataEnable** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setHiSpeedDataDisable** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setSuperSpeedDataEnable** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setSuperSpeedDataDisable** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setPowerEnable** (unsigned int *id, struct Result *result, const unsigned char channel)

Enable only the power line for a port without changing the state of the data lines.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_setPowerDisable** (unsigned int *id, struct Result *result, const unsigned char channel)

Disable only the power line for a port without changing the state of the data lines.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

Returns

Returns common entity return values

void **usb_getPortCurrent** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the current through the power line for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns common entity return values

void **usb_getPortVoltage** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the voltage on the power line for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in microvolts (1 == 1e-6V).

Returns

Returns common entity return values

void **usb_getHubMode** (unsigned int *id, struct Result *result)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **usb_setHubMode** (unsigned int *id, struct Result *result, const unsigned int mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **mode** – The USB hub mode.

Returns

Returns common entity return values

void **usb_clearPortErrorStatus** (unsigned int *id, struct Result *result, const unsigned char channel)

Clear the error status for the given port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The port to clear error status for.

Returns

Returns common entity return values

void **usb_getUpstreamMode** (unsigned int *id, struct Result *result)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **usb_setUpstreamMode** (unsigned int *id, struct Result *result, const unsigned char mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0 and usbUpstreamModePort1

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **mode** – The Upstream port mode.

Returns

Returns common entity return values

void **usb_getUpstreamState** (unsigned int *id, struct Result *result)

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

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **usb_setEnumerationDelay** (unsigned int *id, struct Result *result, const unsigned int ms_delay)

Set the inter-port enumeration delay in milliseconds.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **ms_delay** – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

Returns common entity return values

void **usb_getEnumerationDelay** (unsigned int *id, struct Result *result)

Get the inter-port enumeration delay in milliseconds.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **usb_setPortCurrentLimit** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

Returns

Returns common entity return values

void **usb_getPortCurrentLimit** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the current limit for the port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** – The current limit setting.

Returns

Returns common entity return values

void **usb_setPortMode** (unsigned int *id, struct Result *result, 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 datasheet for a complete list of capabilities. Some devices use a common bit mapping for port mode at usbPortMode

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – USB downstream channel to set the mode on.
- **mode** – The port mode setting as packet bit mask.

Returns

Returns common entity return values

void **usb_getPortMode** (unsigned int *id, struct Result *result, 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 datasheet for a complete list of capabilities. Some devices implement a common bit mapping for port mode at usbPortMode

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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

void **usb_getPortState** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the current State for the Port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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

void **usb_getPortError** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the current error for the Port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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

void **usb_setUpstreamBoostMode** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **setting** – Upstream boost setting 0, 1, 2, or 3.

Returns

Returns common entity return values

void **usb_setDownstreamBoostMode** (unsigned int *id, struct Result *result, 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

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **setting** – Downstream boost setting 0, 1, 2, or 3.

Returns

Returns common entity return values

void **usb_getUpstreamBoostMode** (unsigned int *id, struct Result *result)

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **usb_getDownstreamBoostMode** (unsigned int *id, struct Result *result)

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

Returns

Returns common entity return values

void **usb_getDownstreamDataSpeed** (unsigned int *id, struct Result *result, 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

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **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

void **usb_setConnectMode** (unsigned int *id, struct Result *result, const unsigned char channel, const unsigned char mode)

Sets the connect mode of the switch.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel.
- **mode** – The connect mode
 - usbManualConnect = 0
 - usbAutoConnect = 1

Returns

Returns common entity return values

void **usb_getConnectMode** (unsigned int *id, struct Result *result, const unsigned char channel)

Gets the connect mode of the switch.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **mode** – The current connect mode

Returns

Returns common entity return values

void **usb_setCC1Enable** (unsigned int *id, struct Result *result, const unsigned char channel, const unsigned char bEnable)

Set Enable/Disable on the CC1 line.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – - USB channel.
- **bEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

void **usb_getCC1Enable** (unsigned int *id, struct Result *result, const unsigned char channel)

Get Enable/Disable on the CC1 line.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

void **usb_setCC2Enable** (unsigned int *id, struct Result *result, const unsigned char channel, const unsigned char bEnable)

Set Enable/Disable on the CC2 line.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – - USB channel.
- **bEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

void **usb_getCC2Enable** (unsigned int *id, struct Result *result, const unsigned char channel)

Get Enable/Disable on the CC1 line.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

void **usb_getCC1Current** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the current through the CC1 for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns common entity return values

void **usb_getCC2Current** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the current through the CC2 for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns

Returns common entity return values

void **usb_getCC1Voltage** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the voltage of CC1 for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns common entity return values

void **usb_getCC2Voltage** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the voltage of CC2 for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns common entity return values

void **usb_setSBUEnable** (unsigned int *id, struct Result *result, const unsigned char channel, const unsigned char bEnable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel.
- **bEnable** –
 - Disabled: 0
 - Enabled: 1

Returns

Returns common entity return values

void **usb_getSBUEnable** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the Enable/Disable status of the SBU

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** – The enable/disable status of the SBU

Returns

Returns common entity return values

void **usb_setCableFlip** (unsigned int *id, struct Result *result, const unsigned char channel, const unsigned char bEnable)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel.
- **bEnable** –

- Disabled: 0
- Enabled: 1

Returns

Returns common entity return values

void **usb_getCableFlip** (unsigned int *id, struct Result *result, const unsigned char channel)

Get Cable flip setting.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** – The enable/disable status of cable flip.

Returns

Returns common entity return values

void **usb_setAltModeConfig** (unsigned int *id, struct Result *result, const unsigned char channel, const unsigned int configuration)

Set USB Alt Mode Configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

Returns

Returns common entity return values

void **usb_getAltModeConfig** (unsigned int *id, struct Result *result, const unsigned char channel)

Get USB Alt Mode Configuration.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **configuration** – The USB configuration for the given channel.

Returns

Returns common entity return values

void **usb_getSBU1Voltage** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the voltage of SBU1 for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns common entity return values

void **usb_getSBU2Voltage** (unsigned int *id, struct Result *result, const unsigned char channel)

Get the voltage of SBU2 for a port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns common entity return values

3.6.23 USBSystem Entity

group **USBSystemEntity**

USBSystem Class The USBSystem class provides high level control of the lower level Port Class.

void **usbsystem_getUpstream** (unsigned int *id, struct Result *result, const int index)

Gets the upstream port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

void **usbsystem_setUpstream** (unsigned int *id, struct Result *result, const int index, const unsigned char port)

Sets the upstream port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **port** – The upstream port to set.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **msDelay** – The delay in milliseconds to be applied between port enables

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enabledList** – Bit packed representation of the enabled status for all ports to be applied.

Returns

Returns common entity return values

void **usbsystem_getModeList** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const int bufferLength)

Gets the current mode of all ports with a single call. Equivalent to calling PortClass:getMode() on each port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

void **usbsystem_setModeList** (unsigned int *id, struct Result *result, const int index, const unsigned int bufLength)

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

void **usbsystem_getStateList** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const int bufferLength)

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filled

- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **behavior** – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns common entity return values

void **usbsystem_getPowerBehaviorConfig** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const int bufferLength)

Gets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

void **usbsystem_setPowerBehaviorConfig** (unsigned int *id, struct Result *result, const int index, const unsigned int bufLength)

Sets the current power behavior configuration. Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

Returns

Returns common entity return values

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.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **behavior** – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns common entity return values

void **usbsystem_getDataRoleBehaviorConfig** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const int bufferLength)

Gets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine priority host priority.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns common entity return values

void **usbsystem_setDataRoleBehaviorConfig** (unsigned int *id, struct Result *result, const int index, const unsigned int bufLength)

Sets the current data role behavior configuration. Certain data role behaviors use a list of ports to determine host priority.

Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bufLength** – Length of the buffer to be transferred.

Returns

Returns common entity return values

Warning: doxygenfunction: Cannot find function “usbsystem_getSelectorMode” in doxygen xml output for project “BrainStem2_LabVIEW” from directory: doxml/labVIEW/xml

Warning: doxygenfunction: Cannot find function “usbsystem_setSelectorMode” in doxygen xml output for project “BrainStem2_LabVIEW” from directory: doxml/labVIEW/xml

void **usbsystem_resetEntityToFactoryDefaults** (unsigned int *id, struct Result *result, const int index)

3.7 Reflex Language Reference

re-flex: 'rēˌfleks/

(*noun: reflex; plural noun: reflexes*) An action that is performed as a response to a stimulus and without conscious thought.

3.7.1 Introduction

The Reflex language is a high level, C-like language designed to support rapid development of embedded applications that bridge the gap between hardware and software. The reflex language leverages the BrainStem command protocol to make it easy for developers to turn a network of devices into a functional system.

The following sections in the BrainStem Language documentation will dive into the Reflex language.

3.7.2 Working with Reflex files

Writing reflexes requires a few tools. You'll need access to your favorite text editor, and the programs provided in the BrainStem support download. These tools include:

- **arc** - The reflex language compiler
- **Python** - The BrainStem python library can now load and unload reflex files.

The BrainStem support download can be obtained from the [download](#) page on the [acroname](#) website.

A Note about Directories

Acroname prefers to distribute a set of directories you can place anywhere on your system rather than rely on platform specific installation routines, and default program locations.

The BrainStem Support download includes the following directories:

- **acroname** - Top level folder
 - **bin** - Executable apps and libraries
 - **development** - Examples and API libraries (Python and C++)
 - **alInclude** - Header files for reflexes

Arc, the Acroname Reflex Compiler

Arc can be run from the command line. It will compile reflex source files from within the **bin** folder or from a relative file path (absolute paths will not work). The reflex files will be compiled into map files and placed into the local directory. Map files, which are binary byte code files are executed by the Reflex Virtual Machine on BrainStem modules.

```
$>./arc -h
```

This will print information about using the arc compiler. Arc has a couple of useful flags useful for working with reflex file output, but first let us look at the basic command line for compiling reflex files.

```
$>./arc ../development/reflex_examples/Blink_LED.reflex
```

This will read in the Blink_LED.reflex source file located in the **development/reflex_examples** folder, and produce the compiled result at **bin/Blink_LED.map**.

```
$>./arc -p ../development/reflex_examples/Blink_LED.reflex
```

This will read in the Blink_LED.reflex source file in the **development/reflex_examples** directory, and output the resulting tokenized output of the lexer portion of the compilation process.

```
$>./arc -a ../development/reflex_examples/Blink_LED.reflex
```

This will read in the Blink_LED.reflex source file located in the **development/reflex_examples** folder, and will output the AST for the reflex.

```
$>./arc -d Blink_LED.map
```

This will read in the compiled Blink_LED.map binary located in the **bin** folder, and will generate a .dsm human readable disassembly file in the **bin** directory. See [Appendix II: Reflex map file Disassembly](#) for more information about understanding the format of the disassembly.

Loading reflex .map files with ReflexLoader

ReflexLoader is our new Command Line Interface (CLI) tool for loading .map files into any BrainStem device. It can be found in the BrainStem development kit under the **bin** folder.

```
$>./ReflexLoader -H
```

This will print the usage information for ReflexLoader. If you are having trouble getting something to work this is one of the best places to look for more information. One thing that is important to make note of is that specifying a device '-d' is required. Additionally, that means that you must also have a device (SN), store and slot specified. Failure to do so will cause the program to return an error. With that said a command is also required if you were wanting the program to actually do something; however, failure to include one will not cause an error.

```
$>./ReflexLoader -L -i /location/of/file.map -d 0x40F5849A INTERNAL 0
```

This command will load '-L' the input file '-i' /location/of/file.map to the device '-d' into the INTERNAL store at slot 0.

Unloading a slot is just as easy but instead of using the '-L' and '-i' you would use '-U' and '-o' to specify where you would like the file to go. ReflexLoader will not create any file paths that do not exist so you must give a valid path.

```
$>./ReflexLoader -E -d 0x40F5849A RAM 0
```

This command will enable '-E' the reflex that is loaded into the RAM store at slot 0 in the specified device 0x40F5849A. Keep in mind that although you can load a reflex into RAM it will not survive a power cycle as it is volatile memory.

To disable this slot just swap the '-E' for '-D'

```
$>./ReflexLoader -B -d 0x40F5849A INTERNAL 0
```

This last command will make a slot bootable. Once this has been set and a power cycle occurs the BrainStem device will automatically enable the reflex at the designated slot. In order to disable a boot slot you will need to replace the slot parameter '0' in this case with 255.

Loading reflex .map files with Python

The BrainStem python library can be used to load compiled map files onto BrainStem modules. To load .map files with python you will need to follow the installation instructions of the [getting started guide](#) in the Python section of this reference.

To load a reflex map file into a slot on your brainstem module. First compile the reflex with arc. The resulting map file can be found in your aObject directory. Then start the python interpreter, and import the brainstem package. Instantiate your module and connect. Once you're connected, the following code block will show you how to load, enable and disable the .map file.

```
>>> fh = open('path/to/map/mymap.map', 'rb')
>>> mapfile = fh.read()
>>> fh.close()
>>> stem.store[0].loadSlot(0, mapfile, len(mapfile)) #loads map into store 0 slot 0
0
>>> stem.store[0].slotEnable(0)
0
>>> stem.store[0].slotDisable(0)
```

The above code simply opens the map file for reading in binary mode, reads the contents of the file into the mapfile variable, loads that data into store 0 slot 0 and enables and disables the slot.

Boot Reflexes

You can also set your stem to enable the map file on boot. To do this you will need to load the map file into a slot on the internal store of your module (the internal store is index 0). Then set the boot slot on the module by accessing the setBootSlot procedure of the system module

```
>>> stem.system.setBootSlot(0)
0
>>> stem.system.setBootSlot(255) # will disable the slot from being enabled on boot.
```

If you do not have python installed on your system, you can also use the aConsole utility to load .map files.

3.7.3 A Basic “Hello World” Example.

```
1  #include <a40PinModule.reflex>
2  #define ON 1
3  #define OFF 0
4
5  a40PinModule stem;
6
7  reflex mapEnable()
8  {
9      stem.system.setLED(ON);
10 }
11
12 reflex mapDisable()
13 {
14     stem.system.setLED(OFF);
15 }
```

Wait Where's the “Hello World?”

We don't actually print “hello world” in this example. Since we're working with a BrainStem, the equivalent action is to turn on and off the User LED. This is exactly what the reflex above accomplishes. Next we will break this example down line by line to illuminate some of the interesting bits. Finally we will look at a slightly more complex example which outlines one of the ways the BrainStem architecture is different (and better) than other embedded languages you may be familiar with.

Includes

Line one includes the BrainStem module definition file we will be using in the example. There is generally one of these for each type of BrainStem module you may encounter. See [Appendix I](#) to have a look at one of these files. The BrainStem module definition file tells the reflex compiler what the BrainStem is capable of doing.

Defines

We follow a C-like convention for the majority of our language constructs, and the `#define` preprocessor directive is one of the preprocessor directives we support. Lines 2 and 3 define some human readable names for the UserLED states.

The Module declaration

```
a40PinModule stem;
```

The module declaration creates a named “instance” of a BrainStem. Instance is in quotes because there is no real concept of an object in the reflex language, but in essence the declaration provides us with a way to refer to a particular BrainStem. Each module in a BrainStem network has a unique module address, by default `a40PinModule stem;` without an argument refers to the module on which the reflex file is loaded, while a module declaration with a module address, like `a40PinModule stem2(8);` refers to the module with address 8 and is not necessarily the module on which the reflex is loaded.

Keywords and builtins

We try to minimize the number of keywords, and builtin bits that you as a user have to learn, but there are a couple of keywords in the reflex language that are used all the time. `reflex` defines a top level routine that is attached to an event or entity in the BrainStem system. There are 4 built in reflexes that can be defined by the user to perform behaviors on certain system events. The first two are represented in this example `mapEnable` and `mapDisable` are startup and teardown reflexes that are called whenever a reflex file is enabled and disabled. The second set of built in routines is `linkUp` and `linkDown`, and are called when a connection to a host is established or disconnected.

A reflex declaration looks like the following snippet. It starts with the keyword `reflex`, then declares the entity for which the reflex is executed, is enclosed with `{` and `}`. We will see an example of an entity reflex declaration in the second code example.

```
reflex 'entity'()
{
    ...
}
```

There is no `main` routine in the reflex world. When you enable a map, you are essentially attaching behaviors to certain entities or events which occur in the system. Enable does not necessarily execute code, code execution is a side effect of defining a behavior to the `mapEnable` reflex. In the case of the example, this is what we want.

There is no `main`, only Zuul!

It is worth reiterating this point because it is one of the main issues new users encounter with the Reflex system. Most programming languages have a single entry point, typically called “main”. There is no such concept when writing reflexes. Reflexes are meant to respond to changing conditions within the BrainStem system. As such, relying on `mapEnable` as a main-like routine is a discouraged practice, and can have negative consequences for the speed and performance of your reflex. So, no `main` ... got it. Moving on.

The Rest

The rest of the example is relatively straight forward. The `mapEnable` routine as defined on lines 7–10 turns on the user LED when the mapfile is enabled, and the `mapDisable` routine on lines 12–15 turns the LED off when the mapfile is disabled. The final interesting bit is the command sent to the BrainStem to enable or disable the user LED.

```
stem.system.setLED(ON);
```

This command is fairly typical of many BrainStem commands. It consists of three parts, the module where we will send the command, in this case the current module, the BrainStem entity `system` to which the `setLED` command belongs, and the argument `ON` which sets the state of the user LED.

Next we'll take a look at a more complex example, where we blink the LED multiple times, which highlights one of the main strengths of the BrainStem Virtual Machine: a BrainStem is a multiprocess machine, designed to do more than one thing at a time.

Note: A note about includes. We follow the convention that an include file in `<>` brackets comes from the `ainclude` directory, and is usually an acronym supplied file, like the Module definitions. An include that is surrounded by quotes `" "` is searched by path from the `aUser` directory. So, `"mylib/myReflex.reflex"` would look in the **mylib** folder within the **aUser** directory.

3.7.4 Blink My LED Example

This second reflex code example highlights some of the syntactical differences in the reflex language from that of basic Ansi C, and introduces you to a couple of the fundamental reflex concepts that make it different from many other embedded languages. In this example we will see the preferred method for writing a timed loop, and we will also introduce the ScratchPad, and its use as shared variable.

Here is the entire reflex file. Enabling the reflex will start the user LED blinking at a half second interval until we disable it.

```
1 // file: flashmyled.reflex
2
3 #include <a40PinModule.reflex>
4 a40PinModule stem;
5
6 // 1/2 second delay
```

(continues on next page)

(continued from previous page)

```

7  #define DELAY 500000
8  #define ON 1
9  #define OFF 0
10
11 pad[0:0] unsigned char state;
12
13 reflex mapEnable()
14 {
15     state = ON;
16     stem.system.setLED(state);
17     stem.timer[0].setMode(timerModeRepeat);
18     stem.timer[0].setExpiration(DELAY);
19
20 } // end of mapEnable
21
22 reflex timer[0].expiration()
23 {
24     if (state == ON) {
25         state = OFF;
26     } else {
27         state = ON;
28     }
29     stem.system.setLED(state);
30
31 } // end of timer reflex
32
33 reflex mapDisable()
34 {
35     stem.timer[0].setExpiration(0);
36     stem.timer[0].setMode(timerModeSingle);
37     stem.system.setLED(OFF);
38
39 }

```

The Timer Entity

Timer is a BrainStem entity class that every BrainStem module includes. It allows users to schedule events to be executed at some time in the future. Its range is approximately 1 microsecond to 4200 seconds, and its resolution is in microseconds. There is more information about the timer entity in the BrainStem Entities reference section, however there are two commands that we will use in this example.

`timer setExpiration` is the method of scheduling a timer event to occur in the future, it takes one argument which is a 32bit integer representing the number of microseconds in the future that the timer should execute.

`timer setMode` is the second command we use in the example. Timers have two modes, single (which is the default) and repeat. Single timers execute once, and, as you guessed, repeat timers execute on the interval defined in `setExpiration`. For a timer in repeat mode, setting the expiration to 0 will stop execution of the timer event.

```

stem.timer[0].setMode(timerModeRepeat);
stem.timer[0].setExpiration(DELAY);

```

Lines 17 and 18 in the code listing set up a repeat timer with an interval of 500000 microseconds or 1/2 second. You will have noticed the array like syntax of the command, BrainStem entities sometimes come in groups. There are generally 4 or 8 timers in a BrainStem module, and we address each timer by its index just

as if we had set up an array of timers.

The Timer Reflex

```
reflex timer[0].expiration()
{
    if (state == ON) {
        state = OFF;
    } else {
        state = ON;
    }
    stem.system.setLED(state);
}
```

This is the first time we have seen an “entity” reflex declaration. This code defines the behavior that should occur when the timer expires and the timer triggers.

```
reflex timer[0].expiration() { ...
```

Notice that we declare the timer index we want to attach this behavior to in the declaration. Timers don’t receive any arguments, but other Entity types like analogs and digitals do.

In this case we are setting the User LED either on or off depending on the current state. This leads naturally to the question of how to share state between reflex routines.

The ScratchPad

Individual reflex routines are essentially separate processes within the BrainStem system. They have their own execution space, and variable scope. The only way to share information between two reflex routines is to use the shared data space we call the ScratchPad.

```
pad[0:0] unsigned char state;
```

Line 11 in the code listing declares a single byte in the pad for use as our shared LED state variable. We define it as an unsigned char, and we declare its extents with the array-like syntax `[0:0]`. If we were declaring a short or an int their extents might be something like `[0:1]` and `[0:3]` respectively. When declaring multiple pad variables, the extents must be mutually exclusive. In other words, variables can not overlap. Declaring one variable with the extent `[0:3]` and another with the extent `[2:5]` would not make any sense. Generally pads are around 300 bytes, but this is module specific, so check your data sheet.

Now that we’ve declared our shared variable, it can be used in any of our reflexes.

This doesn’t seem correct. If it is not thread safe there is no guarantee that a read won’t happen in the middle of a write: In general the pad is not safe from multiple concurrent modifications, so the idiomatic pattern is to have one producer which modifies a pad variable, and one or more consumers which read it.

The Rest

The rest of the example should be familiar to you if you followed us from the hello world example. The `mapEnable` routine (lines 13–20) sets up our state, turns on the userLED, and sets the timer to expire at 1/2 second intervals. The `timer[0].expiration` routine (lines 22–31) toggles the userLED from off to on or vice versa. Finally the `mapDisable` routine (lines 33–39) shuts down the timer, and turns off the userLED.

Now that we have traced through two basic examples, the remainder of the reflex reference describes the reflex language in-depth. For more information about BrainStem entities and capabilities see the Entities section of this reference. The entity concept is useful for those wishing to write host code in C, C++, or Python.

Note: There are few native datatypes in the Reflex language, essentially these are all numeric values. They are `char`, `short` and `int`, and they all come in `signed` and `unsigned` flavors. Types are signed by default unless specified with `unsigned` keyword.

3.7.5 Built in reflex origins

BrainStem devices have a number of built in reflex origins that can be mapped to provide reflex functionality when certain system events occur. Each reflex routine in the reflex file is declared similarly. Built in origins do not have arguments passed in. for Example:

```
reflex mapEnable() {
    // ... Reflex statements
}
```

mapEnable

This reflex is triggered when the reflex map file is enabled.

transportUp

On a reset of a module this reflex is triggered when its transport becomes ready (USB is enumerated, TCPIP has acquired an address).

linkUp

A BrainStem link to the host has been created.

linkDown

The BrainStem link has been disconnected.

transportDown

The transport is currently down (USB no upstream connection, TCPIP has lost IP address, or has been disconnected)

mapDisable

Actions to be executed before the map file is disabled.

Map Enable

The map enable reflex is the primary entry point for reflex code when a map file is enabled on a module slot. The reflex machine within the module first adds each of the reflexes defined in the map file to their respective origins, and then executes the code inside the mapEnable reflex.

Example:

```
reflex mapEnable() {  
    // ...  
}
```

Transport Up

Transport Up is called when the Module detects that its primary transport to the host is configured and ready for communication. On USB modules this is when the host has successfully enumerated the Module. In TCP/IP based modules, this event occurs when the stem detects that it has a valid IP address.

```
reflex transportUp() {  
    // ...  
}
```

Link Up

Link up is called when the module detects that an active BrainStem link has been established.

```
reflex linkUp() {  
    // ...  
}
```

Link Down

Link down is called when the module detects that an active BrainStem link has been disconnected.

```
reflex linkDown() {  
    // ...  
}
```

Transport Down

Transport Down is called when the Module detects that its primary transport to the host has been disconnected.

```
reflex transportDown() {  
    // ...  
}
```

Map Disable

Map Disable is called when the map file is disabled. The reflex machine within the module first executes this reflex origin, and then removes all allocated reflex origins for the map.

Example:

```
reflex mapDisable() {
    // ...
}
```

3.7.6 Keywords in the Reflex Language

The Reflex language has a couple of keywords with special meaning. The word `reflex` is used to declare reflexes. The `pad` keyword is used in the declaration of pad variables. Other than these two keywords, many of the language keywords share semantics with the same keyword in the ANSI C language.

Keywords		
<code>reflex</code>	<code>pad</code>	<code>asm</code>
<code>if</code>	<code>else</code>	<code>for</code>
<code>while</code>	<code>do</code>	<code>switch</code>
<code>case</code>	<code>return</code>	<code>char</code>
<code>int</code>	<code>short</code>	<code>unsigned</code>
<code>signed</code>	<code>continue</code>	<code>break</code>

Keyword EBNF

```
keyword ::= 'reflex' | 'pad' | 'asm' | 'if' | 'else' | 'for' | 'while' | 'do'
          | 'switch' | 'case' | 'return' | 'char' | 'int' | 'short'
          | 'unsigned' | 'signed' | 'continue' | 'break' ;
```

3.7.7 Operators and Precedence

The Reflex language shares many of its operators with other C-like languages. The following table presents the different operators, and organizes them by precedence. Operators higher in the table take precedence over operators at lower rows in the table.

Level	Operator	Description	Associativity
1	++	Pre Increment	Left-to-right
	--	Pre Decrement	
	()	Function call	
	[]	Index Subscript	
	.	command selection	

continues on next page

Table 5 – continued from previous page

Level	Operator	Description	Associativity
2	++ -- + - ! ~ (T)	Post Increment Post Decrement Unary plus Unary minus Logical NOT Bitwise NOT Type cast	Right-to-left
3	* / %	Multiplication Divide Modulo	Left-to-right
4	+ -	Addition Subtraction	Left-to-right
5	<< >>	Bitwise Left Shift Bitwise Right Shift	Left-to-right
6	< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left-to-right
7	== !=	Equal to NOT Equal to	Left-to-right
8	&	Bitwise AND	Left-to-right
9	^	Bitwise XOR	Left-to-right
10		Bitwise OR	Left-to-right
11	&&	Logical AND	Left-to-right
12		Logical OR	Left-to-right

continues on next page

Table 5 – continued from previous page

Level	Operator	Description	Associativity
13	? :	Ternary Operator	Right-to-left
14	=	Assignment	Right-to-left
	+=	Assignment by sum	
	-=	Assignment by difference	
	*=	Assignment by product	
	/=	Assignment by quotient	
	%=	Assignment by modulo	
	<<=	Assignment by left shift	
	>>=	Assignment by right shift	

3.7.8 Types, Identifiers and Numbers

Reflex limits types to integer types and booleans. There is no floating point type included in the language, and new types cannot be declared. There are a couple of cases where certain constructs behave like types. For instance module definitions and declarations behave in some ways like types, but are not treated as such in the grammar or in the language definition. Identifiers follow the familiar C-like rules, Identifiers must start with a character which is not a digit, terminal character or '+' and identifiers cannot shadow keywords.

Legal Identifiers

```
__hello
value
v1234
$value
@value
aHappyVariable
a$Happy@Variable
another_happy_variable
```

Illegal Character

```
1badvar
+mybadvar
;anotherbadvar
```

Integer Literals

Negative integer literals can be written by prefixing the integer with a `-` sign. Signed values are represented by two's complement values as in other C-like languages.

Integer literals can also be represented as hexadecimal values to write a hex value, prefix the literal with `0x`. So 127 can be represented as `0x7F` and -128 can be represented as `-0xF0`

Signed vs Unsigned

A given type is considered a signed value unless it is prefixed with the `unsigned` keyword. The language does include the `signed` keyword and a value may be fully and explicitly declared as such, but in general use of `signed` is not necessary.

Type	Size in Bytes	Min	Max	unsigned Max
char (byte)	1	-128	127	255
short	2	-32768	32767	65535
int	4	-2147483648	2147483647	4294967295

Identifiers, and Type Declarations EBNF

Identifier

```
letter      ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J'
              | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T'
              | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z' | 'a' | 'b' | 'c' | 'd'
              | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n'
              | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x'
              | 'y' | 'z' ;
nonterminal ::= '_' | '$' | '@' ;
ID           ::= letter | nonterminal , { letter | digit | nonterminal } ;
```

Integer Literal

```
digit-not-zero ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'
                  | '9' ;
digit           ::= '0' | digit-not-zero ;
hex-digit       ::= digit | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'a' | 'b' | 'c'
                  | 'd' | 'e' | 'f' ;
INT             ::= '0'
                  | digit-not-zero , { digit }
                  | '-' digit-not-zero , { digit }
                  | '0x' hex-digit , { hex-digit }
                  | '-0x' hex-digit , { hex-digit } ;
```


Type Specifier

```

TYPE          ::= 'char' | 'short' | 'int' ;
type_specifier ::= TYPE | 'unsigned' , TYPE | 'signed' , TYPE ;

```

3.7.9 The Reflex Preprocessor

Reflex source files are preprocessed prior to compilation. There are a few useful preprocessor directives which are available within a reflex source file. These directives are borrowed from the C language;

Preprocessor keywords
define
ifdef/endif
include

Examples

```

#define VAL 1234
#define DEBUG

#ifdef DEBUG
...
#endif

#include "myroutines.reflex"
#include <a40PinModule.reflex>

```

3.7.10 Variable Declaration

Reflex supports two types of global variable, and it supports a single scope of variables within reflex and routine declarations. Because reflexes are limited to a single variable scope. A declared variable is visible anywhere within the routine. Redefining a reflex variable will cause a compile time error. In addition, variables must be declared at the beginning of the routine, before any other type of statement.

All “Global” variables are declared as specification on Scratchpad elements. Or as module declarations See the Scratchpad section of the reference for more information about the usage of the scratchpad. Module declarations are used to declare any BrainStem modules on the BrainStem bus that will be accessed by the reflexes and routines in the reflex map.

Module Delcarations

A module declaration allows a reflex or routine to interface with a specific BrainStem module on the BrainStem network. These module declarations must come before any reflex routine definitions.

```
a40PinModule stem;  
// or  
a40PinModule stem(address);
```

The first example above declares the module that the reflex is running on, the address in the first example is implicit, and will be the module address of the current module. The second form allows an explicit module address to be given. This is particularly usefull for communicating with other modules on the BrainStem BUS.

Module Declaration EBNF

```
module_declaration ::= model_def , ID , ";"  
model_def          ::= "Model definitions are included in the reflex"  
                    " map via #include directives"  
                    " for example; #include <a40PinModule.reflex>"
```

Pad Variables

The Scratchpad allows reflex routines, and the host to share information and state. the Reflex language provides a way for a user to declare a typed variable to be stored at an offset within the Scratchpad.

```
pad[0:3] unsigned int value;
```

The keyword `pad` starts the declaration, the range declaration `[0:3]`, defines the indices of the pad that will be allocated for the variable. The type specification `unsigned int` follows the pad declaration and finally the variable name is given.

This fully specifies a pad variable. Pad variables cannot overlap in offset within the pad. An int declaration of `pad[0:3]` and a second of `pad[2:5]` would fail at compile time.

Pad Declaration EBNF

```
pad_declaration ::= 'pad' , "[" , INT , ":" , INT , "]" ,  
                  type_specifier , ID , ";" ;
```

Routine Variables

Variables can be declared within a routine or reflex block. They must be declared at the beginning of the block before any other expression or statement. Routine variable declarations should be familiar developers familiar with C-like languages. Variables may be initialized when they are declared but initialization at declaration is not required.

```
char value;
// or
char value = 12;
```

Routine Variable Declaration EBNF

```
variable_declaration ::= type_specifier , ID , [ "=" , INT ] , ";" ;
```

3.7.11 Statements

Statements in Reflex include control structures, assignment statements, and routine and reflex execution statements. A reflex or routine is made of a series of statements. In addition, a *variable_declaration* is a special type of statement which must precede other types of statements within a statement list.

Control Statements

The largest class of statements are the control statements. These include branching statements like If-else and switch statements and looping control statements with include for and while loops. The reflex language syntax for control statements follows the familiar C-Like pattern. However the lack of sub scoping and the fact that variables are defined before other statments means that care must be given when introducing control statments which use variables.

There is only one variable scope within the reflex language and that is the routine/reflex scope. compound statements that are part of a loop or if statement do not introduce new scope, and declared variables are visible and available throughout the routine.

Control statments include:

Control Statements	
If/Else	Branching
Switch	Branching
While	Looping
For	Looping
Do while	Looping

If/Else

If/Else statements follow the C syntax. For example;

```
if ( a == b ) {  
    ...  
} else {  
    ...  
}
```

Switch

Switch statements follow the C syntax. For example;

```
switch ( a ) {  
    case 1:  
        ...  
        break;  
  
    case 2:  
        ...  
        break;  
  
    default:  
        ...  
        break;  
}
```

While

While statements follow the C syntax. For example;

```
while ( a != b ) {  
    ...  
}
```

For

For statements follow the C syntax. However the iteration variable must be declared with The rest of the variable declarations at the beginning of the reflex or routine compound statement. For example;

```
reflex mapEnable() {  
    int i;  
    for ( i = k; i > 0; i-- ) {  
        ...  
    }  
}
```

Do While

Do while statements follow the C syntax. For example;

```
do {
    ...
} while (a != b);
```

3.7.12 Reflex and Routine Definition

There are two types of top level definitions in the reflex language, reflex definitions and routine definitions. A reflex file consists of 1 or more reflex definitions, and zero or more routine definitions. The major syntactic difference between a routine definition and a reflex definition is that a reflex definition must be preceded by the keyword `reflex`, the fact that reflex definitions often include an entity index specification [`index`] as part of the reflex name declaration and reflexes do not return values.

Semantically the two types are very different. Routines are very similar to functions in other C-like languages. They exist to perform a function and are called within other code blocks. Reflexes on the other hand are tied to events which occur in the BrainStem module, such as an analog reading, timer expiration, or in the case of map enable and disable, the enabling of the compiled reflex file itself. In the reflex nomenclature we say that a reflex definition is tied to a reflex 'origin', multiple reflexes can be attached to a reflex origin through the enabling of multiple reflex files, and each reflex definition that is 'mapped' to an origin will execute when the condition triggering the origin occurs. In this way reflexes form a reactive system of behaviors on each BrainStem module.

Note: Reflexes cannot return values themselves. These are behaviors performed as the result of some system event, and as such have no place, like `main`, to return a value to. If a reflex needs to retain state at the end of its execution, it should place such state information into the Scratchpad.

Reflex Definition

```
reflex system.timer[0].expiration(void)
{
    ...
}
```

Routine Definition Example

```
short linearizeIRData(short data) {
    ...
    return linearizedData;
}
```

Routine definitions are syntactically much like function definitions in C. They declare a return type, and take zero or more parameters.

Reflex Definition EBNF

```
reflex_definition ::= 'reflex' , entity_specifier , parameter_list ,  
                      compound_statement ;  
parameter_list   ::= '(' [ parameter , { ',' parameter } ] ')' ;  
parameter        ::= type_specifier , ID ;  
compound_statement ::= '{' statement_list '}' ;  
statement_list   ::= { variable_declaration } , { statement } ;  
statement        ::= routine_call | assignment_statement | control_statement ;
```

Routine Definition EBNF

```
routine_definition ::= type_specifier , ID , parameter_list ,  
                      compound_statement ;
```

Entity Specifier EBNF

The entity specifier fully describes a brainstem UEI. See the [UEI appendix](#) for more information. UEI's consist of an entity class, the entity index, and an option/command.

```
entity_specifier ::= entity_class , [ "[" , index , "]" ] , "." , class_option ;  
index            ::= '0' | digit-not-zero , digit ;  
entity_class     ::= "Module specific list of possible entities"  
class_option     ::= "Entity specific list of possible options for the entity"
```

3.7.13 Appendix

Appendix I: Example Reflex Module Definition file.

A reflex module definition file is used by the arc compiler to determine which entities and capabilities the BrainStem device supports. These files exist in the alnclude folder, and are included in a Reflex file that will use the module.

Example Include Directive

```
#include <a40PinModule.reflex>
```

Reflex Module Definition syntax

Module definition files are similar to C style headers, and support preprocessor directives like `#include` and `#define`.

The module definition begins with some includes for constants, and a module declaration `module a40PinModule`. Each line of the declaration block defines an individual entity or capability.

```
<cmdType> [Index] { <cmdOption>, type, <ueiRequestTypes(GET|SET)> }
```

```
#ifndef __a40PinModule_reflex__
#define __a40PinModule_reflex__

#include "aProtocoldefs.h"
#include "a40PinModuleDefs.h"

module a40PinModule
{
cmdANALOG[0]  { analogConfiguration, 0, ueiOPTION_GET }
cmdANALOG[0]  { analogVoltage,      0, ueiOPTION_GET }
cmdANALOG[1]  { analogConfiguration, 0, ueiOPTION_GET }
cmdANALOG[1]  { analogVoltage,      0, ueiOPTION_GET }
cmdANALOG[2]  { analogConfiguration, 0, ueiOPTION_GET }
cmdANALOG[2]  { analogVoltage,      0, ueiOPTION_GET }
cmdANALOG[3]  { analogConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdANALOG[3]  { analogVoltage,      0, ueiOPTION_SET | ueiOPTION_GET }

cmdAPP[0]     { appExecute,          0, ueiOPTION_SET }
cmdAPP[0]     { appReturn,           0, ueiOPTION_SET }
cmdAPP[1]     { appExecute,          0, ueiOPTION_SET }
cmdAPP[1]     { appReturn,           0, ueiOPTION_SET }
cmdAPP[2]     { appExecute,          0, ueiOPTION_SET }
cmdAPP[2]     { appReturn,           0, ueiOPTION_SET }
cmdAPP[3]     { appExecute,          0, ueiOPTION_SET }
cmdAPP[3]     { appReturn,           0, ueiOPTION_SET }

cmdCLOCK[0]   { clockYear,           0, ueiOPTION_SET | ueiOPTION_GET }
cmdCLOCK[0]   { clockMonth,          0, ueiOPTION_SET | ueiOPTION_GET }
cmdCLOCK[0]   { clockDay,            0, ueiOPTION_SET | ueiOPTION_GET }
cmdCLOCK[0]   { clockHour,           0, ueiOPTION_SET | ueiOPTION_GET }
cmdCLOCK[0]   { clockMinute,         0, ueiOPTION_SET | ueiOPTION_GET }
cmdCLOCK[0]   { clockSecond,         0, ueiOPTION_SET | ueiOPTION_GET }
```

(continues on next page)

(continued from previous page)

```

cmdDIGITAL[0] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[0] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[1] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[1] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[2] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[2] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[3] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[3] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[4] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[4] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[5] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[5] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[6] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[6] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[7] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[7] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[8] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[8] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[9] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[9] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[10] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[10] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[11] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[11] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[12] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[12] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[13] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[13] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[14] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdDIGITAL[14] { digitalState, 0, ueiOPTION_SET | ueiOPTION_GET }

cmdPOINTER[0] { pointerOffset, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[0] { pointerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[0] { pointerTransferStore, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[0] { pointerChar, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[0] { pointerShort, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[0] { pointerInt, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[0] { pointerTransferToStore, 0, ueiOPTION_SET }
cmdPOINTER[0] { pointerTransferFromStore, 0, ueiOPTION_SET }

cmdPOINTER[1] { pointerOffset, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[1] { pointerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[1] { pointerTransferStore, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[1] { pointerChar, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[1] { pointerShort, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[1] { pointerInt, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[1] { pointerTransferToStore, 0, ueiOPTION_SET }
cmdPOINTER[1] { pointerTransferFromStore, 0, ueiOPTION_SET }

cmdPOINTER[2] { pointerOffset, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[2] { pointerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[2] { pointerTransferStore, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[2] { pointerChar, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[2] { pointerShort, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[2] { pointerInt, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[2] { pointerTransferToStore, 0, ueiOPTION_SET }

```

(continues on next page)

(continued from previous page)

```

cmdPOINTER[2]  { pointerTransferFromStore, 0, ueiOPTION_SET }

cmdPOINTER[3]  { pointerOffset, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[3]  { pointerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[3]  { pointerTransferStore, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[3]  { pointerChar, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[3]  { pointerShort, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[3]  { pointerInt, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdPOINTER[3]  { pointerTransferToStore, 0, ueiOPTION_SET }
cmdPOINTER[3]  { pointerTransferFromStore, 0, ueiOPTION_SET }

cmdSTORE[0]    { storeSlotEnable, 0, ueiOPTION_SET }
cmdSTORE[0]    { storeSlotDisable, 0, ueiOPTION_SET }
cmdSTORE[0]    { storeSlotState, 0, ueiOPTION_GET }
cmdSTORE[0]    { storeWriteSlot, 0, ueiOPTION_GET }
cmdSTORE[0]    { storeReadSlot, 0, ueiOPTION_GET }
cmdSTORE[0]    { storeCloseSlot, 0, ueiOPTION_SET }

cmdSTORE[1]    { storeSlotEnable, 0, ueiOPTION_SET }
cmdSTORE[1]    { storeSlotDisable, 0, ueiOPTION_SET }
cmdSTORE[1]    { storeSlotState, 0, ueiOPTION_GET }
cmdSTORE[1]    { storeWriteSlot, 0, ueiOPTION_GET }
cmdSTORE[1]    { storeReadSlot, 0, ueiOPTION_GET }
cmdSTORE[1]    { storeCloseSlot, 0, ueiOPTION_SET }

cmdSTORE[2]    { storeSlotEnable, 0, ueiOPTION_SET }
cmdSTORE[2]    { storeSlotDisable, 0, ueiOPTION_SET }
cmdSTORE[2]    { storeSlotState, 0, ueiOPTION_GET }
cmdSTORE[2]    { storeWriteSlot, 0, ueiOPTION_GET }
cmdSTORE[2]    { storeReadSlot, 0, ueiOPTION_GET }
cmdSTORE[2]    { storeCloseSlot, 0, ueiOPTION_SET }

cmdSYSTEM[0]   { systemModule, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdSYSTEM[0]   { systemRouter, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdSYSTEM[0]   { systemHBInterval, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdSYSTEM[0]   { systemLED, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdSYSTEM[0]   { systemSleep, 0, ueiOPTION_SET }
cmdSYSTEM[0]   { systemBootSlot, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdSYSTEM[0]   { systemVersion, 0, ueiOPTION_GET }
cmdSYSTEM[0]   { systemModel, 0, ueiOPTION_GET }
cmdSYSTEM[0]   { systemSerialNumber, 0, ueiOPTION_GET }
cmdSYSTEM[0]   { systemSave, 0, ueiOPTION_SET }
cmdSYSTEM[0]   { systemReset, 0, ueiOPTION_SET }
cmdSYSTEM[0]   { systemInputVoltage, 0, ueiOPTION_GET }

cmdTIMER[0]    { timerExpiration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[0]    { timerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[1]    { timerExpiration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[1]    { timerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[2]    { timerExpiration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[2]    { timerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[3]    { timerExpiration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[3]    { timerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[4]    { timerExpiration, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[4]    { timerMode, 0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[5]    { timerExpiration, 0, ueiOPTION_SET | ueiOPTION_GET }

```

(continues on next page)

(continued from previous page)

```
cmdTIMER[5]    { timerMode,          0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[6]    { timerExpiration,    0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[6]    { timerMode,          0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[7]    { timerExpiration,    0, ueiOPTION_SET | ueiOPTION_GET }
cmdTIMER[7]    { timerMode,          0, ueiOPTION_SET | ueiOPTION_GET }
}

#endif // __a40PinModule_reflex__
```

Appendix II: Reflex map file Disassembly.

The arc compile can provide a disassembly of a .map bytecode file for easier debugging of opcodes, and instructions. An example disassembly file is given below, with explanation.

Reflex file flashmyled.reflex

The reflex file that will be shown in the disassembly example.

```
#include <a40PinModule.reflex>
a40PinModule stem;

#define DELAY 500000
#define ON 1
#define OFF 0

pad[0:0] unsigned char state;

reflex mapEnable()
{
    state = ON;

    stem.system.setLED(state);
    stem.timer[0].setMode(1);
    stem.timer[0].setExpiration(DELAY);
}

reflex timer[0].expiration() {
    if (state == ON) {
        state = OFF;
    } else {
        state = ON;
    }

    stem.system.setLED(state);
}

reflex mapDisable() {
    stem.timer[0].setExpiration(0);
    stem.timer[0].setMode(0);
    stem.system.setLED(OFF);
}
```

Header Lines

The disassembly header provides information about the arc version.

```
Acroname Reflex Machine file
Version 1.0
-----
```

Reflex and Routine Jump Table

The next section of the disassembly is the jump table within the bytecode for the entry points to each declared reflex and routine. The routine symbol is followed by the byte location of the entry instruction of the routine, or reflex.

```
3 Reflex Maps
mapEnable() -> 0x0000
timer[0].expiration(void) -> 0x0024
mapDisable() -> 0x004A
-----
```

VM Instruction lines

Instruction lines are read left to right and then down on the right. The fields are: [byteOffset, VM operation, operation parameter, byte sequence]. The byte sequence is the raw bytecode represented vertically.

offset	VMop	param	bytes
0x0002	pushls	0x0000	0x04 0x00 0x00

Disassembly (.dsm) of the flashmyled reflex.

```
Acroname Reflex Machine file
Version 1.0
-----
3 Reflex Maps
mapEnable() -> 0x0000
timer[0].expiration(void) -> 0x0024
mapDisable() -> 0x004A
-----
0x0000 pushlc 0x01 0x03
0x0002 pushls 0x0000 0x04
0x0005 poppc 0x0000 0x3C
0x0006 pushls 0x0000 0x04
0x0009 pushpc 0x038460 0x74
0x000A popec 0x03 0x84
0x000E popn 0x01 0x02
```

(continues on next page)

(continued from previous page)

0x0010	pushlc	0x01	0x01
			0x03
			0x01
0x0012	popec	0x4F8260	0x74
			0x4F
			0x82
			0x60
0x0016	popn	0x01	0x02
			0x01
0x0018	pushli	0x0007A120	0x05
			0x20
			0xA1
			0x07
			0x00
0x001D	popei	0x4F8160	0x76
			0x4F
			0x81
			0x60
0x0021	popn	0x01	0x02
			0x01
0x0023	exit		0x01
0x0024	pushls	0x0000	0x04
			0x00
			0x00
0x0027	pushpc		0x39
0x0028	pushlc	0x01	0x03
			0x01
0x002A	eqc		0x54
0x002B	popn	0x01	0x02
			0x01
0x002D	brz	0x0039	0x12
			0x39
			0x00
0x0030	pushlc	0x00	0x03
			0x00
0x0032	pushls	0x0000	0x04
			0x00
			0x00
0x0035	poppc		0x3C
0x0036	br	0x003F	0x11
			0x3F
			0x00
0x0039	pushlc	0x01	0x03
			0x01
0x003B	pushls	0x0000	0x04
			0x00
			0x00
0x003E	poppc		0x3C
0x003F	pushls	0x0000	0x04
			0x00
			0x00
0x0042	pushpc		0x39
0x0043	popec	0x038460	0x74
			0x03
			0x84
			0x60

(continues on next page)

(continued from previous page)

0x0047	popn	0x01	0x02 0x01
0x0049	exit		0x01
0x004A	pushlc	0x00	0x03 0x00
0x004C	convci		0x1C
0x004D	popei	0x4F8160	0x76 0x4F 0x81 0x60
0x0051	popn	0x01	0x02 0x01
0x0053	pushlc	0x00	0x03 0x00
0x0055	popec	0x4F8260	0x74 0x4F 0x82 0x60
0x0059	popn	0x01	0x02 0x01
0x005B	pushlc	0x00	0x03 0x00
0x005D	popec	0x038460	0x74 0x03 0x84 0x60
0x0061	popn	0x01	0x02 0x01
0x0063	exit		0x01

4.1 BrainStem Platform

Acroname's BrainStem technology is a foundational tool for bridging the gap between hardware and software systems.

4.1.1 What is BrainStem?

What makes the BrainStem platform unique and exciting? Get an [overview](#).

4.1.2 Explore your module's capabilities With HubTool.

New Hardware? See what is included in the box, and how to quickly get connected to your new hardware. Find out about links and your hardware's I/O capabilities.

4.1.3 Are you up-to-date?

Is your hardware running the latest firmware? Learn how to take advantage of new features or make sure you don't get bitten by any nasty bugs. Also learn how to easily update your modules using aUpdater.

4.1.4 Hello World Reflex

Get started writing Reflex code for your BrainStem module. This tutorial will introduce you to the basics of the Reflex language and how to load and enable your reflex with aStemStoreTool.

4.1.5 Hello World C++

Now that you've seen the world of Reflex, Say hello to the C++ API. This tutorial will introduce you to the basics of writing a C++ application that communicates with your module to say hello.

4.1.6 Next Steps

BrainStem Terminology

Read up about why we call it what we call it.

Reflex Language Reference

Explore the reflex programming language, and the advantages of enabling basic behaviors on your BrainStem devices.

C++ API Reference

Create rich host based applications using the C++ BrainStem API.

Python API Reference

Create rich host based applications using the Python BrainStem API.

C API Reference

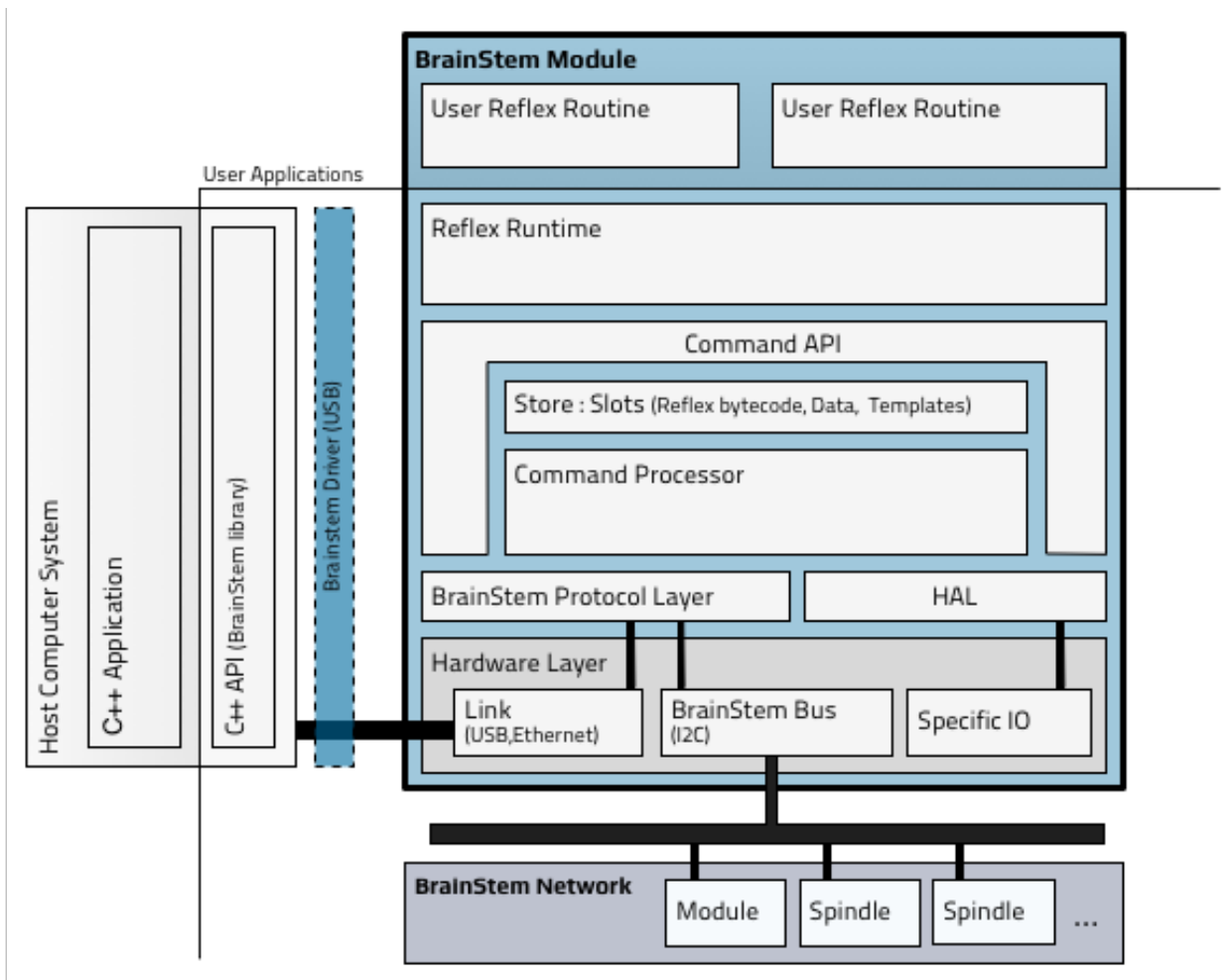
Dig into the lower level C API for more control, or to extend BrainStem to your favorite language.

4.2 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.

4.2.1 Embedded With Reflex

A high-level C-like programming language called *Reflex*, and the associated compiler, enable code to execute directly on-board the controller modules. The BrainStem Reflex system allows access to a soft real-time operating system. This type of system is ideal for those who have outgrown the "super-loop" type systems like Arduino.



However, the elegant simplicity and C-like familiarity of the Reflex language makes low level microcontroller programming approachable; no more digging through thousands of pages of datasheets or wiring up expensive JTAG debuggers. The Reflex language allows for simple code to execute complex functions when triggered by external events, without getting bogged down in low level interrupt calls and microcontroller specific limitations. By utilizing BrainStem reflexes, a system can be remotely deployed and react intelligently to its environment. The embedded reflex program can collect and store information for later retrieval on the BrainStem's internal persistent memory or an external micro-SD card.

4.2.2 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.

4.2.3 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.

4.2.4 Next Steps

The best place to get started with your new BrainStem hardware is by checking out the [Getting Started](#) section of our documentation.

4.3 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](#).

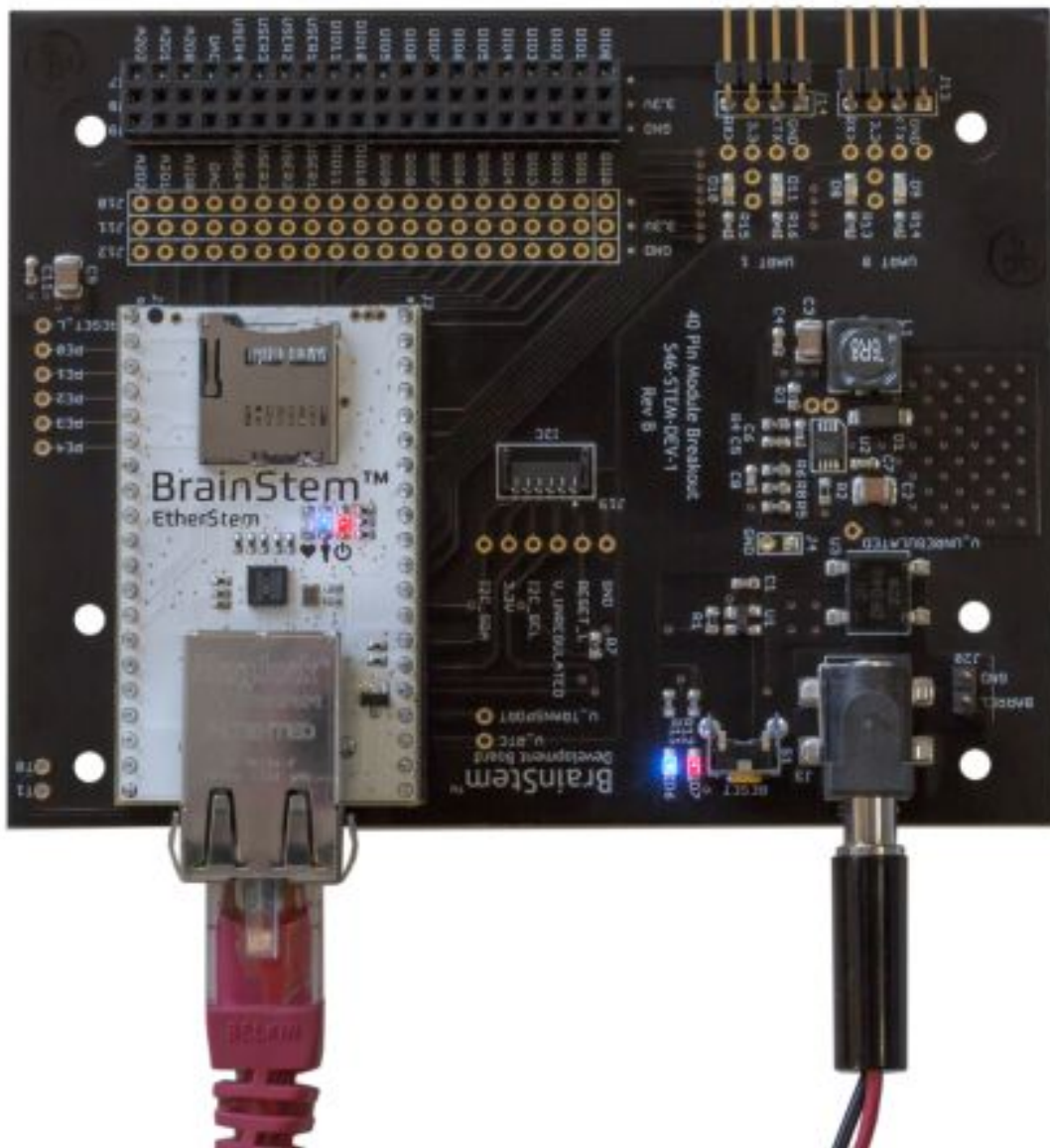
The latest version of the [BrainStem Development Kit or HubTool](#) can be downloaded from the [Acroname Download Page](#). Extract the package to the location of your choice when the download has finished.

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.

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.

Write and execute your first reflex.

Interested in Reflexes? See the [Reflex](#) section of this reference to get started.

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 Firmware Upgrade Precautions

Reliable power should be used when updating firmware and the update process should be allowed to complete without interruption. An interrupted firmware upgrade process will leave the BrainStem unusable and will require using the firmware recovery process.

4.4.2 Brainstem Firmware

Each BrainStem module uses an Acroname developed firmware which contains a boot loader, the BrainStem OS, the reflex virtual machine, access to the BrainStem protocol, and access to functionality specific to each module. Each module's firmware is unique to the module type and the specific module serial number.

The firmware is continuously maintained for bug fixes and feature enhancements. Updates to the firmware can be downloaded from Acroname using the tools detailed below. For large volume customers, offline firmware files can be made available under license.

4.4.3 Firmware Update Tools

BrainStem modules can be updated via their link interface (e.g. USB, EtherNet, etc) by using the command line interface (CLI) “Updater” tool. The Updater tool is packaged in each BrainStem development and HubTool package. The extensive built-in help can be accessed with the “-H” option. Example update processes are shown in the following sections.

For legacy updaters such as: “aUpdater” and “UpdateTool” please see [Legacy Firmware Management](#).

4.4.4 Using Updater 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 “Bin” folder located in the Brainstem Dev Kit download. Downloads for all of Acroname’s products can be found at [Download Center](#) under the Support tab.

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
```

```
Application:Updater
  Version:Version: 1.0 Dec 31 2015 11:00:34
  BrainStem 2.1.5
  Copyright (C) 1994-2015, Acroname Inc.

Updater [options]
  BrainStem Update and Maintenance Application

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].
  -f <bird file>             - Firmware file to be transferred to device.
  -r <router serial#>        - indirect connection through router device
  -s <serial port>          - name of serial port to use for recovery

  -l 'log message'           - message to place in device history log
  -a <mode>                  - set log file access mode [A, X].
                              Override the default append mode [A] with
                              overwrite mode [X].
  -k <filename>              - log messages written into given filename
                              Default:[APP.log]

Commands:
  -B                          - list the Builds available for given device.
```

(continues on next page)

(continued from previous page)

- 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.

- H or -h - Display extended usage information.

- I - View stored device information. Must either use DISCOVER command ["-D"] or specify a specific device ["-d" option].

- L - Log the session output into a log file. Use the "-k" option to override the default filename [APP.log].

- 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.

- R - Send RESET to device. Must specify a specific device ["-d" option].

- V - View stored history for device. Must either use DISCOVER command ["-D"] or specify a specific device ["-d" option].

- X <BIT_MASK> - Bitmask for Logging

DEBUG_Updater	0x00000001
DEBUG_History	0x00000002
DEBUG_Attributes	0x00000004
DEBUG_BirdFile	0x00000008
DEBUG_Network	0x00000010
DEBUG_ServerComm	0x00000020
DEBUG_Network	0x00000100
DEBUG_Network_SR	0x00000200
DEBUG_Network_FIRM	0x00000400
DEBUG_NXPSerial	0x00001000
DEBUG_NXPSerial_E	0x00002000
DEBUG_NXPSerial_SR	0x00004000
DEBUG_APP	0x00010000

- Y - change router address of an network device to given router. Requires: -d <mod:device> and -r <device> options.

Sample usage:

```
-D                   Discover all devices connected by either USB or TCP/IP.
```

(continues on next page)

(continued from previous page)

	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 0x40F5849A	Discover settings of the specific device (serial number 0x40F5849A)
-G -d 0x40F5849A	Get the latest firmware for the given device. Downloads the firmware file into the updater specific device directory.
-U -d 0x40F5849A	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 the device.
-G -d 0x40F5849A -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 0x40F5849A	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 -d 0x40F5849A -X 0x0010 →transfer.	Update the firmware as described above, but turns on the display of networking messages used in the transfer.
-U -t RECOVER -s /dev/serial	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 0xD272031D →network	Discover all devices connected to the I2C BrainStem of the given routing device 0xD272031D.
-G -U -d 0x2181F0EE -r 0xD272031D →0x2181F0EE	Get and Update to the latest firmware for device indirectly through routing device 0xD272031D.
Updater [Version: 1.0 Dec 31 2015 11:00:34] [BrainStem Release:2.1.5] [Copyright (C) 1994-2015, Acroname Inc.]	
NO COMMANDS GIVEN	
Completed processing: Updater [Version: 1.0 Dec 31 2015 11:00:34] [BrainStem Release:2.1.5] [Copyright (C) 1994-2015, Acroname Inc.]	

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.

4.4.5 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 both USB and TCP/IP devices connected to your machine and network.

```
$> ./Updater -D
```

Looking at the output below you can see that two devices were discovered. One USBStem and one EtherStem. In this example we will be using the USBStem 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.

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-D]
Updater [Version 0.2 Nov 24 2015 09:34:27] [Copyright (C) 1994-2015, Acroname Inc.]

Discovering Devices [USB]:
    Device    Module  Router  Model          Firmware Version
    3797E6F8  02        02      04 [USBStem    ]  2.1.5

Discovering Devices [TCPIP]:
    Device    Module  Router  Model          Firmware Version
    856C1C03  06        06      05 [EtherStem    ]  2.1.1 [10.128.38.159]

Completed processing: Updater [Version 0.2 Nov 24 2015 09:34:27] [Copyright (C)
1994-2015, Acroname Inc.]
```

Getting the latest firmware from Acroname's servers:

Using the serial number for the USBStem 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 0x3797E6F8
```

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-G]
    [-d]
    [0x3797E6F8]
```

(continues on next page)

(continued from previous page)

```
Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]
Current build for [3797E6F8][aUSBStem] ==> [130702287].
Get build for [3797E6F8][aUSBStem][130702287] ==> 72784 bytes [YmlyZAEBeJzsunk8lN/b].
GetBuild BIRD [/Users/Mitch/.acroname/updater/3797E6F8/130702287.bird] VERIFIED.
Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C)
1994-2015, Acroname Inc.]
```

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 0x3797E6F8
```

```
Home directory:[/Users/Mitch]
Application parameters:[Updater]
    [Updater]
    [-U]
    [-d]
    [0x3797E6F8]
Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]
Update using firmware file [/Users/Mitch/.acroname/updater/3797E6F8/130702287.bird].
Transferring loader to device
Transferring loader block 0, 8852 bytes
Transferred 1 blocks, 8852 total bytes
Transferring firmware to device [/Users/Mitch/.acroname/updater/3797E6F8/130702287.
↵bird]
Transferring firmware block 0, 8852 bytes
Transferring firmware block 1, 772 bytes
Transferring firmware block 2, 49668 bytes
Transferred 3 blocks, 59292 total bytes
Completed updating firmware on device [3797E6F8]
Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C)
1994-2015, Acroname Inc.]
```

At this point the firmware has been downloaded to the device and the device has reset and is running with the new firmware.

4.4.6 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 6FCBBAA4
```

Below you will see what was printed on my machine²⁷. 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.

```
2017:05:10:18:10:05 | Updater [Version: 1.1 May 1 2017 13:59:12]
[BrainStem Release:2.3.12] [Copyright (C) 1994-2016, Acroname Inc.]

2017:05:10:18:10:05 |

Build List for device[6FCBBAA4]:
2017:05:10:18:10:06 |          Build          Version
2017:05:10:18:10:06 |          219607159          2.2.0
2017:05:10:18:10:06 |          111486747          2.2.1
2017:05:10:18:10:06 |          231932966          2.2.2
2017:05:10:18:10:06 |          156729307          2.2.3
2017:05:10:18:10:06 |          25628300          2.2.4
2017:05:10:18:10:06 |          250406850          2.2.5
2017:05:10:18:10:06 |          226315939          2.2.6
2017:05:10:18:10:06 |          33051391          2.2.7
2017:05:10:18:10:06 |          130872794          2.2.8
2017:05:10:18:10:06 |          131823882          2.3.0
2017:05:10:18:10:06 |          119187462          2.3.1
2017:05:10:18:10:06 |          9473450          2.3.10
2017:05:10:18:10:06 |          4719070          2.3.11
2017:05:10:18:10:06 |          5329028          2.3.12
2017:05:10:18:10:06 |          130782904          2.3.2
2017:05:10:18:10:06 |          70185126          2.3.3
2017:05:10:18:10:06 |          72872664          2.3.4
2017:05:10:18:10:06 |          8157420          2.3.5
2017:05:10:18:10:06 |          10682084          2.3.6
2017:05:10:18:10:06 |          10941901          2.3.7
2017:05:10:18:10:06 |          3196158          2.3.9

2017:05:10:18:10:06 | Completed processing: Updater [Version: 1.1 May 1 2017
↪13:59:12]
[BrainStem Release:2.3.12] [Copyright (C) 1994-2016, Acroname Inc.]
```

²⁷ This output was spliced in to reflect the changes in Updaters output. Therefor, the build numbers will not align with the rest of the example.

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 0x3797E6F8 -b 99528558
```

You should see something similar to the below output.

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-U]
    [-d]
    [0x3797E6F8]
    [-b]
    [99528558]
Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]
Update using firmware file [/Users/Mitch/.acroname/updater/3797E6F8/99528558.bird].
Transferring loader to device
Transferring loader block 0, 8296 bytes
Transferred 1 blocks, 8296 total bytes
Transferring firmware to device [/Users/Mitch/.acroname/updater/3797E6F8/99528558.
↪bird]
Transferring firmware block 0, 8296 bytes
Transferring firmware block 1, 772 bytes
Transferring firmware block 2, 48776 bytes
Transferred 3 blocks, 57844 total bytes
Completed updating firmware on device [3797E6F8]
Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C)
1994-2015, Acroname Inc.]
```

Confirm that you have successfully restored the old firmware:

Just to be safe lets 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 2.1.4 firmware.

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-D]
Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]

Discovering Devices [USB]:
    Device    Module    Router    Model    Firmware Version
    3797E6F8   02         02        04 [USBStem    ]    2.1.4

Discovering Devices [TCPIP]:
```

(continues on next page)

(continued from previous page)

Device	Module	Router	Model	Firmware Version
856C1C03	06	06	05 [EtherStem]	2.1.1 [10.128.38.159]

Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]

4.4.7 Example: Recovering a BrainStem Module

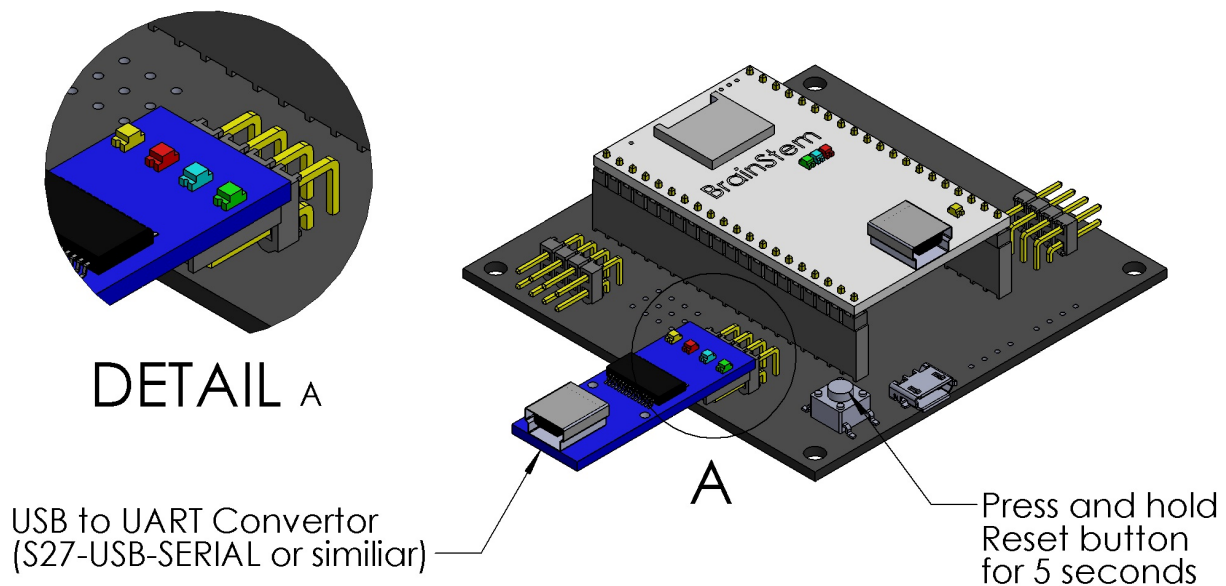
Lets take a look at how we can recover a BrainStem module after it has become unresponsive (aka: “Bricked”). This is also very helpful when dealing with old devices with a firmware version that might not be compatible with the new Updater utility.

In order to preform this recovery process you will need a [USB to Serial Module](#)²⁸. 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](#)²⁹.

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 programmed via the UART port.



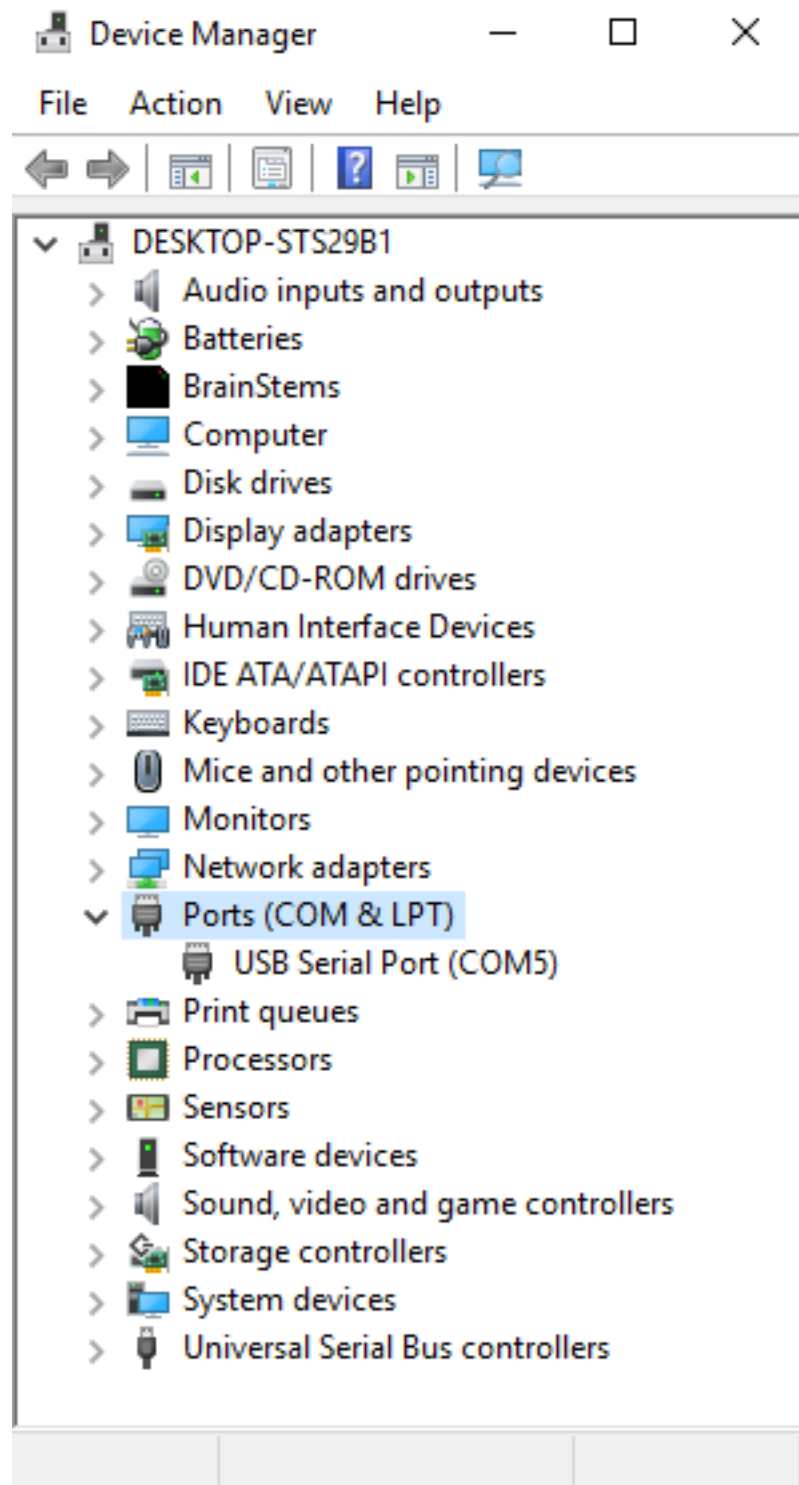
²⁸ <https://acroname.com/products/USB-SERIAL-CONVERTER?sku=S27-USB-SERIAL>

²⁹ <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.]

4.4.8 Example: 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*³⁰ where it will then be converted to the I2C transport and sent to a *MTM-PM1*³¹ module.

Before we begin lets make sure your device is connected and has power. Refer to the *Getting Started* page for additional information.

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.]
```

³⁰ <https://acroname.com/products/MTM-ETHERSTEM-ETHERNET-MICROCONTROLLER-MODULE?sku=S67-MTM-ETHERSTEM>

³¹ <https://acroname.com/products/ACRONAME-MTM-1-CHANNEL-POWER-MODULE?sku=S65-MTM-PM-1>

Brainstem Network Discovery

From the discovery we found a MTM-EtherStem with serial number D272031D. We will need this number in order to preform an 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 pervious 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.
```

(continues on next page)

(continued from previous page)

```

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.
(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.]

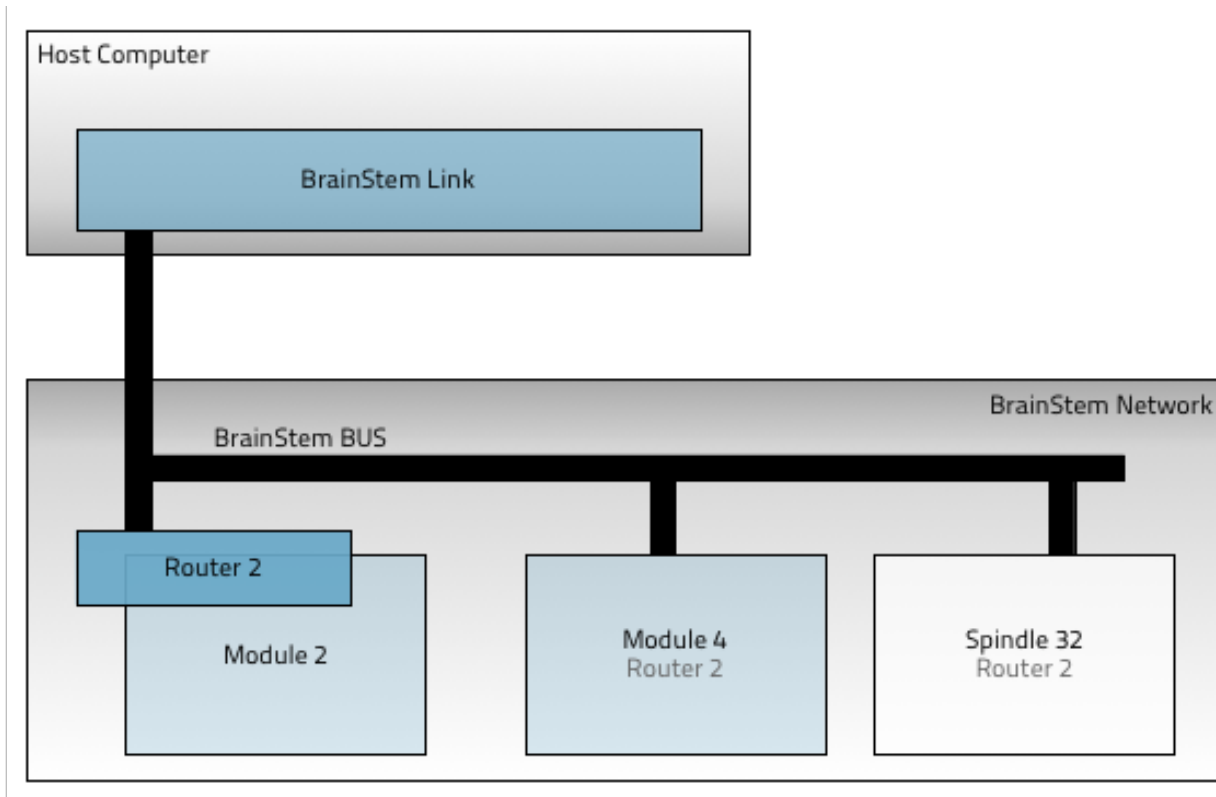
```

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.

4.5 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 I2C³² 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.

³² <http://i2c.info/i2c-bus-specification>

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.

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](#) 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 Reflex

The Reflex programming language is a C-like language that runs on BrainStem controllers. Its simple interface allows a user to quickly implement application specific functionality on a BrainStem module to provide process control, data acquisition, filtering, and other custom behavior on the hardware. Reflex provides a flexible event driven architecture of embedded code, and was born out of the hierarchical control model used in many robotic systems. Reflexes running on BrainStem are ideally suited to reactive first line behaviors just above the metal. Further information about the reflex Language can be found within in the [Reflex Language](#) section.

4.5.7 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.

4.5.8 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.

4.6.1 Mac OS X

On Mac OSX there are no installation procedures required.

4.6.2 Linux Ubuntu

Acroname is currently building and testing with Ubuntu 14.04LTS, on these 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.

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, and HubTool downloads that describes the process for installing the WinUSB Driver on Windows 7 OS.

4.7 Appendix

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.

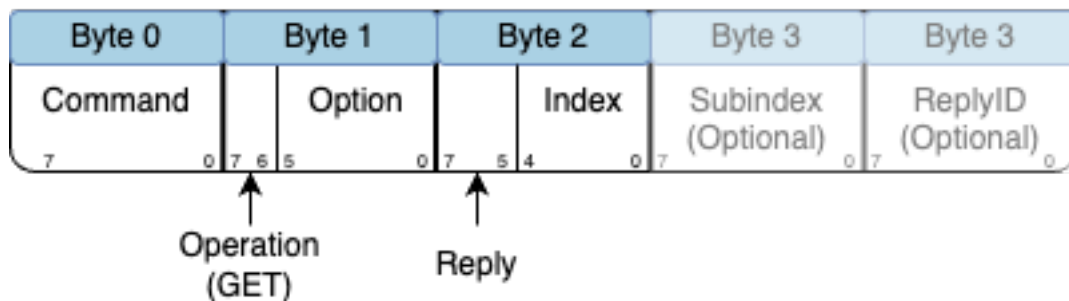


Fig. 1: Packed Byte GET UEI Structure

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.

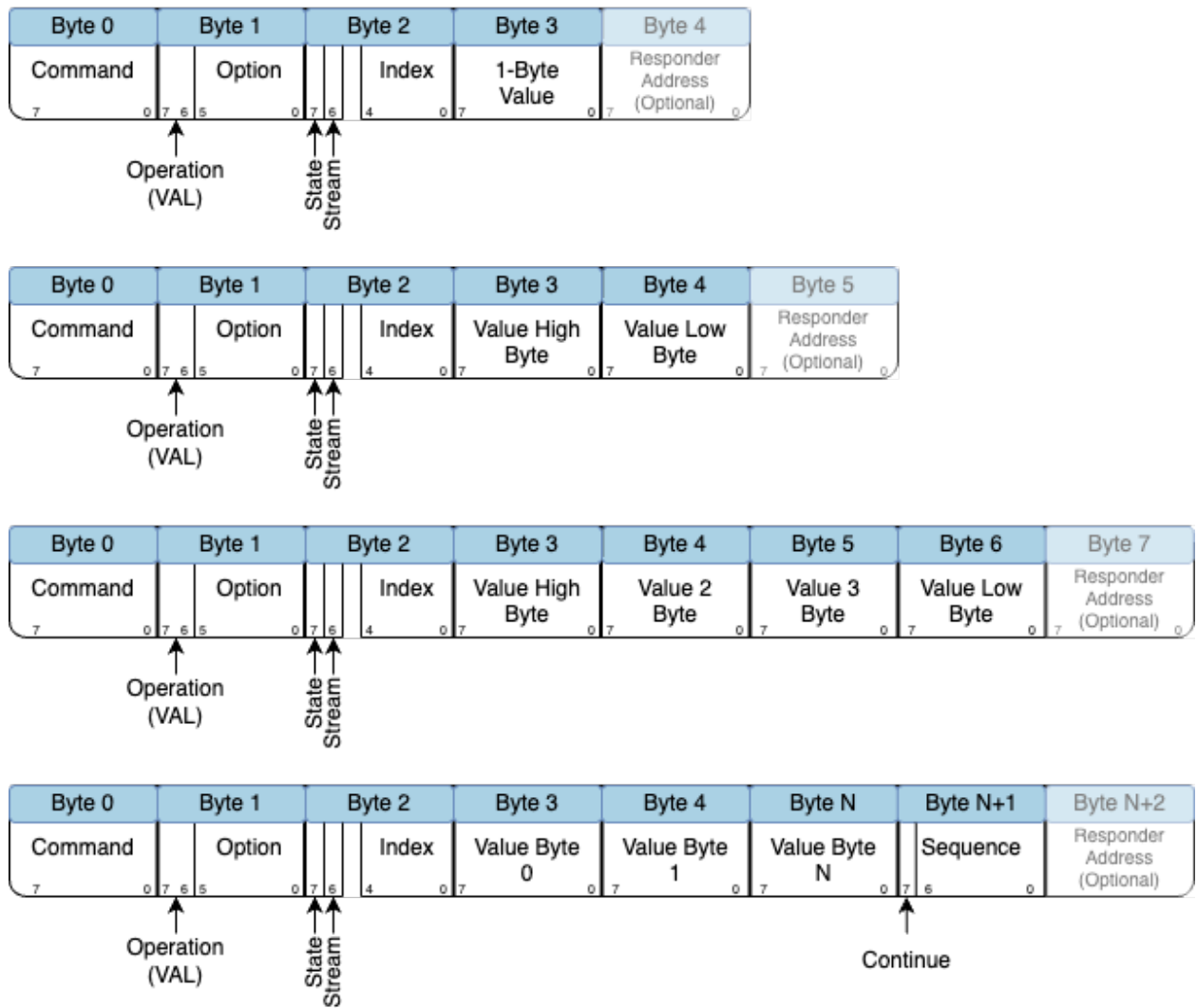


Fig. 2: Packed Byte VAL UEI Structure 1, 2, 4, and N Byte Values

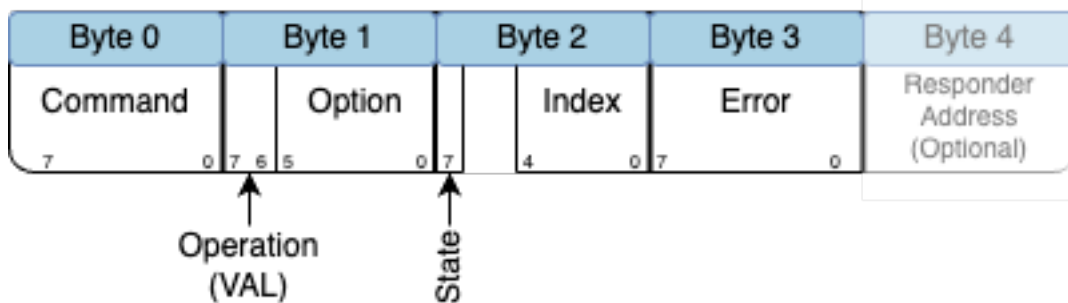


Fig. 3: Packed Byte VAL UEI Error Structure

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.

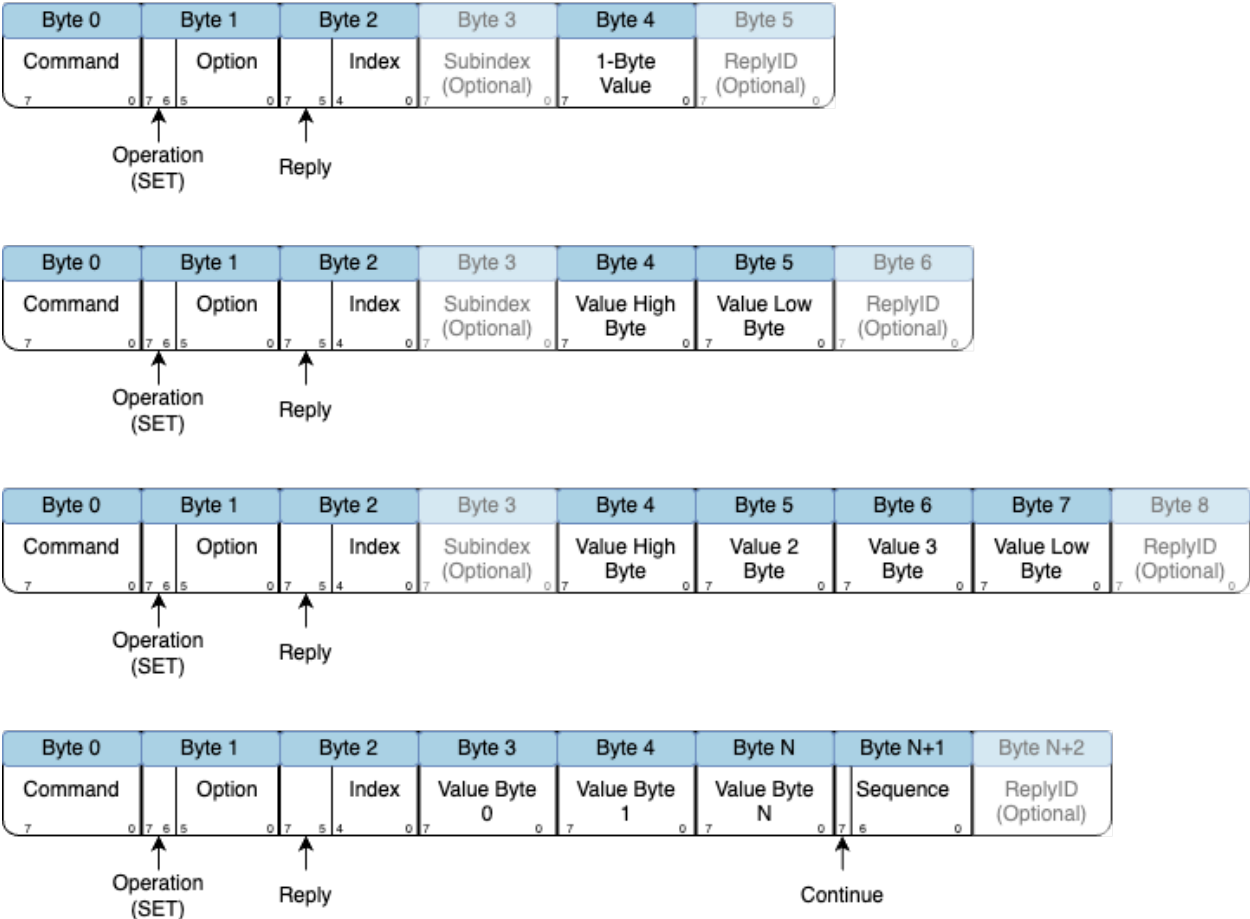


Fig. 4: Packed Byte SET UEI Structure 1, 2, 4, and N Byte Values

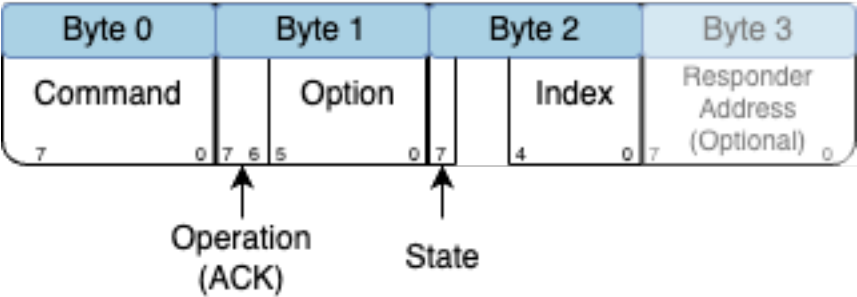


Fig. 5: Packed Byte ACK UEI Structure

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.

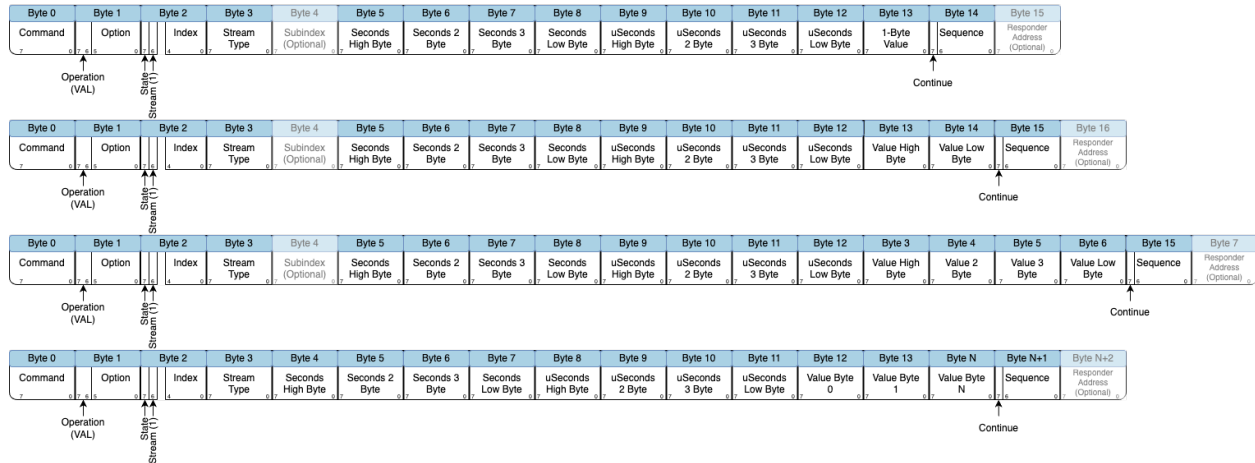


Fig. 7: Packed Byte VAL UEI Stream Structure 1, 2, 4, and N Byte Values

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.

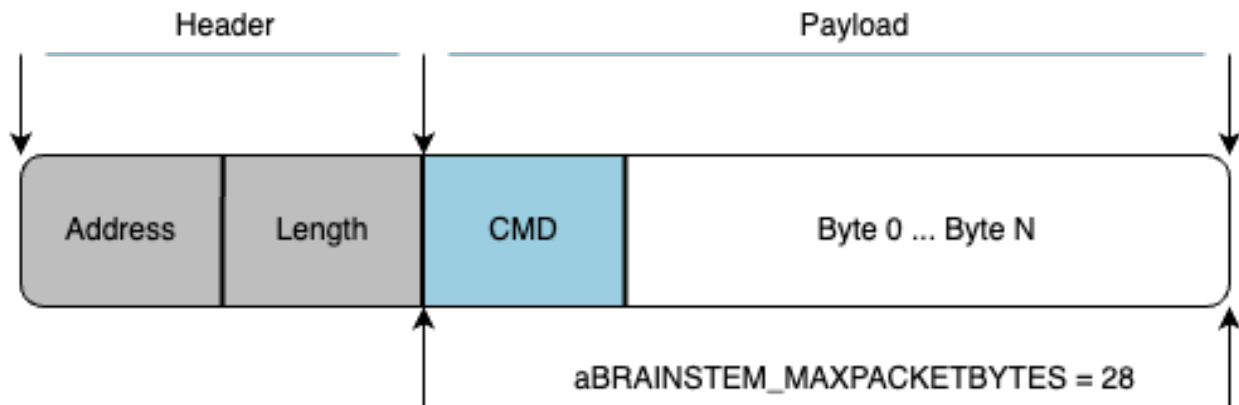


Fig. 8: Typical BrainStem Command Structure

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

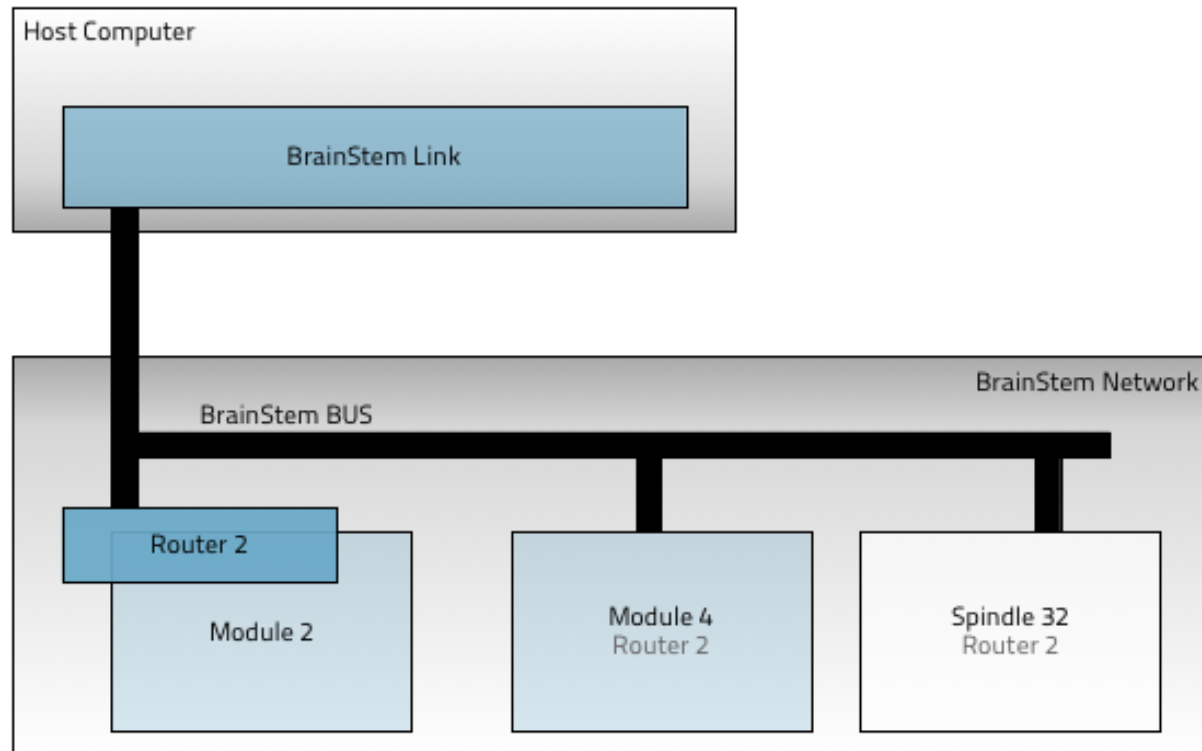
UEI's impose a request response behavior on top of the BrainStem protocol, but the protocol does not itself define a need for a response. Delivery of data is best effort. Some Link transports (TCP/IP) make guarantees about data delivery, this is not part of the BrainStem protocol. Please see specific 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 I2C³³ 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.



³³ <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	6
USBHub2x4 USBHub3+	6
MTM-IO-Serial	8
MTM-DAQ-2	10
MTM-Relay	12

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

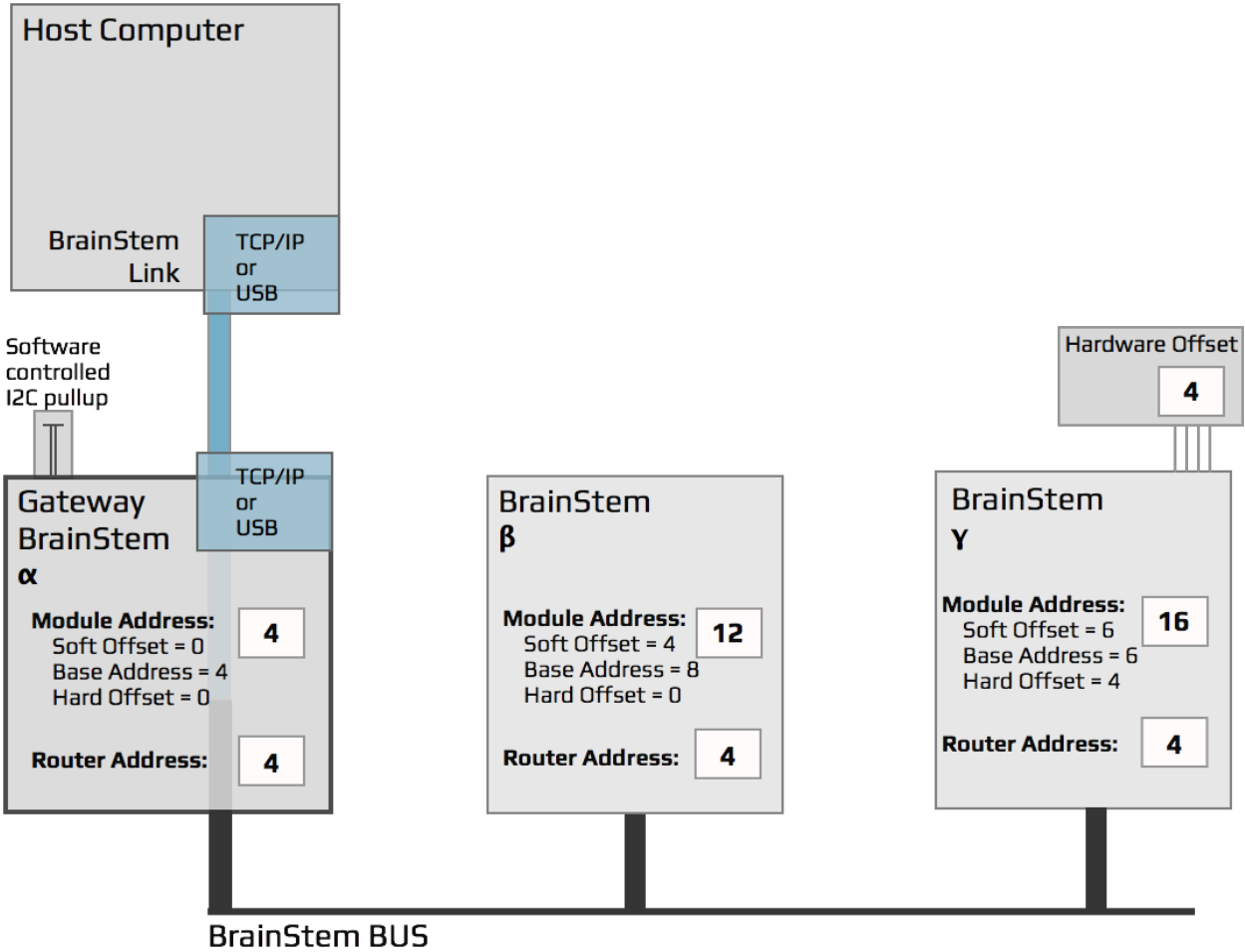
Router Addresses

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 the 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 have 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. This can be very handy if you would like to return to a previous firmware. See *“Example: Reverting to a Previous Version”* in the *“BrainStem Firmware Management”* section.

A

aBaudRate (*C++ enum*), 546
 aBaudRate::aBAUD_115200 (*C++ enumerator*), 546
 aBaudRate::aBAUD_19200 (*C++ enumerator*), 546
 aBaudRate::aBAUD_230400 (*C++ enumerator*), 546
 aBaudRate::aBAUD_2400 (*C++ enumerator*), 546
 aBaudRate::aBAUD_38400 (*C++ enumerator*), 546
 aBaudRate::aBAUD_4800 (*C++ enumerator*), 546
 aBaudRate::aBAUD_57600 (*C++ enumerator*), 546
 aBaudRate::aBAUD_9600 (*C++ enumerator*), 546
 aBRAINSTEM_MAXPACKETBYTES (*C macro*), 517
 Acroname::BrainStem2CLI::aErr (*C++ enum*), 566
 Acroname::BrainStem2CLI::aErr::aErrAsyncReturn (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrBusy (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrCancel (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrConfiguration (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrConnection (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrDuplicate (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrEOF (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrFileNameLength (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrIndexRange (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrInitialization (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrInvalidEntity (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrInvalidOption (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrIO (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrMedia (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrMemory (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrMode (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrNone (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrNotFound (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrNotReady (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrOverrun (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrPacket (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrParam (*C++ enumerator*), 566
 Acroname::BrainStem2CLI::aErr::aErrParse (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrPermission (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrRange (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrRead (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrResource (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrShortCommand (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrSize (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrStreamStale (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrTimeout (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrUnimplemented (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrUnknown (*C++ enumerator*), 568
 Acroname::BrainStem2CLI::aErr::aErrVersion (*C++ enumerator*), 567
 Acroname::BrainStem2CLI::aErr::aErrWrite (*C++ enumerator*), 566

Acroname::BrainStem2CLI::AnalogClass (C++ class), 568
Acroname::BrainStem2CLI::AnalogClass::~~AnalogClass (C++ function), 569
Acroname::BrainStem2CLI::AnalogClass::AnalogClass (C++ function), 569
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureNumberOfSamples (C++ function), 571
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureSampleRate (C++ function), 571
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureState (C++ function), 571
Acroname::BrainStem2CLI::AnalogClass::getConfiguration (C++ function), 570
Acroname::BrainStem2CLI::AnalogClass::getEnable (C++ function), 570
Acroname::BrainStem2CLI::AnalogClass::getRange (C++ function), 570
Acroname::BrainStem2CLI::AnalogClass::getValue (C++ function), 569
Acroname::BrainStem2CLI::AnalogClass::getVoltage (C++ function), 569
Acroname::BrainStem2CLI::AnalogClass::initiateBulkCapture (C++ function), 571
Acroname::BrainStem2CLI::AnalogClass::setBulkCaptureNumberOfSamples (C++ function), 571
Acroname::BrainStem2CLI::AnalogClass::setBulkCaptureSampleRate (C++ function), 571
Acroname::BrainStem2CLI::AnalogClass::setConfiguration (C++ function), 570
Acroname::BrainStem2CLI::AnalogClass::setEnable (C++ function), 570
Acroname::BrainStem2CLI::AnalogClass::setRange (C++ function), 570
Acroname::BrainStem2CLI::AnalogClass::setValue (C++ function), 569
Acroname::BrainStem2CLI::AnalogClass::setVoltage (C++ function), 569
Acroname::BrainStem2CLI::AppClass (C++ class), 572
Acroname::BrainStem2CLI::AppClass::~~AppClass (C++ function), 572
Acroname::BrainStem2CLI::AppClass::AppClass (C++ function), 572
Acroname::BrainStem2CLI::AppClass::execute (C++ function), 572
Acroname::BrainStem2CLI::ClockClass (C++ class), 573
Acroname::BrainStem2CLI::ClockClass::~~ClockClass (C++ function), 573
Acroname::BrainStem2CLI::ClockClass::ClockClass (C++ function), 573
Acroname::BrainStem2CLI::ClockClass::getDay (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::getHour (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::getMinute (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::getMonth (C++ function), 573
Acroname::BrainStem2CLI::ClockClass::getSecond (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::getYear (C++ function), 573
Acroname::BrainStem2CLI::ClockClass::setDay (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::setHour (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::setMinute (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::setMonth (C++ function), 574
Acroname::BrainStem2CLI::ClockClass::setSecond (C++ function), 575
Acroname::BrainStem2CLI::ClockClass::setYear (C++ function), 573
Acroname::BrainStem2CLI::DeviceNode (C++ class), 597
Acroname::BrainStem2CLI::DeviceNode::~~DeviceNode (C++ function), 597
Acroname::BrainStem2CLI::DeviceNode::DeviceNode (C++ function), 597
Acroname::BrainStem2CLI::DeviceNode::hubPort (C++ member), 597
Acroname::BrainStem2CLI::DeviceNode::hubSerialNumber (C++ member), 597
Acroname::BrainStem2CLI::DeviceNode::idProduct (C++ member), 597
Acroname::BrainStem2CLI::DeviceNode::idVendor (C++ member), 597
Acroname::BrainStem2CLI::DeviceNode::speed (C++ member), 597
Acroname::BrainStem2CLI::DigitalClass (C++ class), 575
Acroname::BrainStem2CLI::DigitalClass::~~DigitalClass (C++ function), 575
Acroname::BrainStem2CLI::DigitalClass::DigitalClass (C++ function), 575
Acroname::BrainStem2CLI::DigitalClass::getConfiguration (C++ function), 576
Acroname::BrainStem2CLI::DigitalClass::getState (C++ function), 575
Acroname::BrainStem2CLI::DigitalClass::getStateAll (C++ function), 576
Acroname::BrainStem2CLI::DigitalClass::setConfiguration (C++ function), 575
Acroname::BrainStem2CLI::DigitalClass::setState (C++ function), 575
Acroname::BrainStem2CLI::DigitalClass::setStateAll (C++ function), 576
Acroname::BrainStem2CLI::EqualizerClass (C++ class), 576
Acroname::BrainStem2CLI::EqualizerClass::~~EqualizerClass (C++ function), 577
Acroname::BrainStem2CLI::EqualizerClass::EqualizerClass (C++ function), 577
Acroname::BrainStem2CLI::EqualizerClass::getReceiverConfig (C++ function), 577
Acroname::BrainStem2CLI::EqualizerClass::getTransmitterConfig (C++ function), 577
Acroname::BrainStem2CLI::EqualizerClass::setReceiverConfig (C++ function), 577
Acroname::BrainStem2CLI::EqualizerClass::setTransmitterConfig (C++ function), 577
Acroname::BrainStem2CLI::I2CClass (C++ class), 577
Acroname::BrainStem2CLI::I2CClass::~~I2CClass (C++ function), 578
Acroname::BrainStem2CLI::I2CClass::getSpeed (C++ function), 578
Acroname::BrainStem2CLI::I2CClass::I2CClass (C++ function), 578
Acroname::BrainStem2CLI::I2CClass::read (C++ function), 578
Acroname::BrainStem2CLI::I2CClass::setPullup (C++ function), 578

Acroname::BrainStem2CLI::I2CClass::setSpeed (C++ function), 578
 Acroname::BrainStem2CLI::I2CClass::write (C++ function), 578
 Acroname::BrainStem2CLI::ModuleClass (C++ class), 579
 Acroname::BrainStem2CLI::ModuleClass::~~ModuleClass (C++ function), 579
 Acroname::BrainStem2CLI::ModuleClass::connect (C++ function), 580
 Acroname::BrainStem2CLI::ModuleClass::connectFromSpec (C++ function), 580
 Acroname::BrainStem2CLI::ModuleClass::connectThroughLinkModule (C++ function), 580
 Acroname::BrainStem2CLI::ModuleClass::disconnect (C++ function), 581
 Acroname::BrainStem2CLI::ModuleClass::discoverAndConnect (C++ function), 579
 Acroname::BrainStem2CLI::ModuleClass::findFirstModule (C++ function), 582
 Acroname::BrainStem2CLI::ModuleClass::findModule (C++ function), 582
 Acroname::BrainStem2CLI::ModuleClass::getModuleAddress (C++ function), 581
 Acroname::BrainStem2CLI::ModuleClass::isConnected (C++ function), 581
 Acroname::BrainStem2CLI::ModuleClass::ModuleClass (C++ function), 579
 Acroname::BrainStem2CLI::ModuleClass::reconnect (C++ function), 581
 Acroname::BrainStem2CLI::ModuleClass::sDiscover (C++ function), 582
 Acroname::BrainStem2CLI::ModuleClass::setModuleAddress (C++ function), 581
 Acroname::BrainStem2CLI::ModuleClass::setNetworkingMode (C++ function), 581
 Acroname::BrainStem2CLI::MuxClass (C++ class), 582
 Acroname::BrainStem2CLI::MuxClass::~~MuxClass (C++ function), 583
 Acroname::BrainStem2CLI::MuxClass::getChannel (C++ function), 583
 Acroname::BrainStem2CLI::MuxClass::getChannelVoltage (C++ function), 583
 Acroname::BrainStem2CLI::MuxClass::getConfiguration (C++ function), 584
 Acroname::BrainStem2CLI::MuxClass::getEnable (C++ function), 583
 Acroname::BrainStem2CLI::MuxClass::getSplitMode (C++ function), 584
 Acroname::BrainStem2CLI::MuxClass::MuxClass (C++ function), 583
 Acroname::BrainStem2CLI::MuxClass::setChannel (C++ function), 583
 Acroname::BrainStem2CLI::MuxClass::setConfiguration (C++ function), 584
 Acroname::BrainStem2CLI::MuxClass::setEnable (C++ function), 583
 Acroname::BrainStem2CLI::MuxClass::setSplitMode (C++ function), 584
 Acroname::BrainStem2CLI::PointerClass (C++ class), 584
 Acroname::BrainStem2CLI::PointerClass::~~PointerClass (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::getChar (C++ function), 586
 Acroname::BrainStem2CLI::PointerClass::getInt (C++ function), 586
 Acroname::BrainStem2CLI::PointerClass::getMode (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::getOffset (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::getShort (C++ function), 586
 Acroname::BrainStem2CLI::PointerClass::getTransferStore (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::initiateTransferFromStore (C++ function), 586
 Acroname::BrainStem2CLI::PointerClass::initiateTransferToStore (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::PointerClass (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::setChar (C++ function), 586
 Acroname::BrainStem2CLI::PointerClass::setInt (C++ function), 586
 Acroname::BrainStem2CLI::PointerClass::setMode (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::setOffset (C++ function), 585
 Acroname::BrainStem2CLI::PointerClass::setShort (C++ function), 586
 Acroname::BrainStem2CLI::PointerClass::setTransferStore (C++ function), 585
 Acroname::BrainStem2CLI::PORT_SPEED (C++ enum), 596
 Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_FULL (C++ enumerator), 597
 Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_HIGH (C++ enumerator), 597
 Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_LOW (C++ enumerator), 596
 Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_SUPER (C++ enumerator), 597
 Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_SUPER_PLUS (C++ enumerator), 597
 Acroname::BrainStem2CLI::PORT_SPEED::kPORT_SPEED_UNKNOWN (C++ enumerator), 596
 Acroname::BrainStem2CLI::PortClass (C++ class), 587
 Acroname::BrainStem2CLI::PortClass::~~PortClass (C++ function), 587
 Acroname::BrainStem2CLI::PortClass::getAllocatedPower (C++ function), 594
 Acroname::BrainStem2CLI::PortClass::getAvailablePower (C++ function), 593
 Acroname::BrainStem2CLI::PortClass::getCC1Enabled (C++ function), 591
 Acroname::BrainStem2CLI::PortClass::getCC2Enabled (C++ function), 592
 Acroname::BrainStem2CLI::PortClass::getCCEnabled (C++ function), 591
 Acroname::BrainStem2CLI::PortClass::getCurrentLimit (C++ function), 593
 Acroname::BrainStem2CLI::PortClass::getCurrentLimitMode (C++ function), 593
 Acroname::BrainStem2CLI::PortClass::getDataEnabled (C++ function), 588
 Acroname::BrainStem2CLI::PortClass::getDataHS1Enabled (C++ function), 589
 Acroname::BrainStem2CLI::PortClass::getDataHS2Enabled (C++ function), 589
 Acroname::BrainStem2CLI::PortClass::getDataHSEnabled (C++ function), 588
 Acroname::BrainStem2CLI::PortClass::getDataHSRoutingBehavior (C++ function), 595

Acroname::BrainStem2CLI::PortClass::getDataRole (C++ function), 590
Acroname::BrainStem2CLI::PortClass::getDataSpeed (C++ function), 592
Acroname::BrainStem2CLI::PortClass::getDataSS1Enabled (C++ function), 590
Acroname::BrainStem2CLI::PortClass::getDataSS2Enabled (C++ function), 590
Acroname::BrainStem2CLI::PortClass::getDataSSEnabled (C++ function), 589
Acroname::BrainStem2CLI::PortClass::getDataSSRoutingBehavior (C++ function), 595
Acroname::BrainStem2CLI::PortClass::getEnabled (C++ function), 588
Acroname::BrainStem2CLI::PortClass::getErrors (C++ function), 593
Acroname::BrainStem2CLI::PortClass::getHSBoost (C++ function), 596
Acroname::BrainStem2CLI::PortClass::getMode (C++ function), 592
Acroname::BrainStem2CLI::PortClass::getName (C++ function), 594
Acroname::BrainStem2CLI::PortClass::getPowerEnabled (C++ function), 590
Acroname::BrainStem2CLI::PortClass::getPowerLimit (C++ function), 594
Acroname::BrainStem2CLI::PortClass::getPowerLimitMode (C++ function), 594
Acroname::BrainStem2CLI::PortClass::getPowerMode (C++ function), 588
Acroname::BrainStem2CLI::PortClass::getState (C++ function), 592
Acroname::BrainStem2CLI::PortClass::getVbusAccumulatedPower (C++ function), 595
Acroname::BrainStem2CLI::PortClass::getVbusCurrent (C++ function), 587
Acroname::BrainStem2CLI::PortClass::getVbusVoltage (C++ function), 587
Acroname::BrainStem2CLI::PortClass::getVconn1Enabled (C++ function), 591
Acroname::BrainStem2CLI::PortClass::getVconn2Enabled (C++ function), 591
Acroname::BrainStem2CLI::PortClass::getVconnAccumulatedPower (C++ function), 596
Acroname::BrainStem2CLI::PortClass::getVconnCurrent (C++ function), 587
Acroname::BrainStem2CLI::PortClass::getVconnEnabled (C++ function), 590
Acroname::BrainStem2CLI::PortClass::getVconnVoltage (C++ function), 587
Acroname::BrainStem2CLI::PortClass::getVoltageSetpoint (C++ function), 592
Acroname::BrainStem2CLI::PortClass::PortClass (C++ function), 587
Acroname::BrainStem2CLI::PortClass::resetEntityToFactoryDefaults (C++ function), 596
Acroname::BrainStem2CLI::PortClass::resetVbusAccumulatedPower (C++ function), 596
Acroname::BrainStem2CLI::PortClass::resetVconnAccumulatedPower (C++ function), 596
Acroname::BrainStem2CLI::PortClass::setCC1Enabled (C++ function), 592
Acroname::BrainStem2CLI::PortClass::setCC2Enabled (C++ function), 592
Acroname::BrainStem2CLI::PortClass::setCCEnabled (C++ function), 591
Acroname::BrainStem2CLI::PortClass::setCurrentLimit (C++ function), 593
Acroname::BrainStem2CLI::PortClass::setCurrentLimitMode (C++ function), 593
Acroname::BrainStem2CLI::PortClass::setDataEnabled (C++ function), 588
Acroname::BrainStem2CLI::PortClass::setDataHS1Enabled (C++ function), 589
Acroname::BrainStem2CLI::PortClass::setDataHS2Enabled (C++ function), 589
Acroname::BrainStem2CLI::PortClass::setDataHSEnabled (C++ function), 589
Acroname::BrainStem2CLI::PortClass::setDataHSRoutingBehavior (C++ function), 595
Acroname::BrainStem2CLI::PortClass::setDataSS1Enabled (C++ function), 590
Acroname::BrainStem2CLI::PortClass::setDataSS2Enabled (C++ function), 590
Acroname::BrainStem2CLI::PortClass::setDataSSEnabled (C++ function), 589
Acroname::BrainStem2CLI::PortClass::setDataSSRoutingBehavior (C++ function), 595
Acroname::BrainStem2CLI::PortClass::setEnabled (C++ function), 588
Acroname::BrainStem2CLI::PortClass::setHSBoost (C++ function), 596
Acroname::BrainStem2CLI::PortClass::setMode (C++ function), 593
Acroname::BrainStem2CLI::PortClass::setName (C++ function), 595
Acroname::BrainStem2CLI::PortClass::setPowerEnabled (C++ function), 590
Acroname::BrainStem2CLI::PortClass::setPowerLimit (C++ function), 594
Acroname::BrainStem2CLI::PortClass::setPowerLimitMode (C++ function), 594
Acroname::BrainStem2CLI::PortClass::setPowerMode (C++ function), 588
Acroname::BrainStem2CLI::PortClass::setVconn1Enabled (C++ function), 591
Acroname::BrainStem2CLI::PortClass::setVconn2Enabled (C++ function), 591
Acroname::BrainStem2CLI::PortClass::setVconnEnabled (C++ function), 591
Acroname::BrainStem2CLI::PortClass::setVoltageSetpoint (C++ function), 592
Acroname::BrainStem2CLI::PortMapping (C++ class), 598
Acroname::BrainStem2CLI::PortMapping::~~PortMapping (C++ function), 598
Acroname::BrainStem2CLI::PortMapping::lastError (C++ member), 598
Acroname::BrainStem2CLI::PortMapping::PortMapping (C++ function), 598
Acroname::BrainStem2CLI::PortMapping::update (C++ function), 598
Acroname::BrainStem2CLI::PowerDeliveryClass (C++ class), 599
Acroname::BrainStem2CLI::PowerDeliveryClass::~~PowerDeliveryClass (C++ function), 599
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableCurrentMax (C++ function), 603
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableOrientation (C++ function), 603
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableSpeedMax (C++ function), 603
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableType (C++ function), 603
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableVoltageMax (C++ function), 602

Acroname::BrainStem2CLI::PowerDeliveryClass::getConnectionState (C++ function), 599
 Acroname::BrainStem2CLI::PowerDeliveryClass::getFastRoleSwapCurrent (C++ function), 605
 Acroname::BrainStem2CLI::PowerDeliveryClass::getFlagMode (C++ function), 604
 Acroname::BrainStem2CLI::PowerDeliveryClass::getNumberOfPowerDataObjects (C++ function), 599
 Acroname::BrainStem2CLI::PowerDeliveryClass::getOverride (C++ function), 604
 Acroname::BrainStem2CLI::PowerDeliveryClass::getPeakCurrentConfiguration (C++ function), 605
 Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObject (C++ function), 599
 Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectEnabled (C++ function), 600
 Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectEnabledList (C++ function), 601
 Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectList (C++ function), 600
 Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerRole (C++ function), 602
 Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerRolePreferred (C++ function), 602
 Acroname::BrainStem2CLI::PowerDeliveryClass::getRequestDataObject (C++ function), 601
 Acroname::BrainStem2CLI::PowerDeliveryClass::setFastRoleSwapCurrent (C++ function), 599
 Acroname::BrainStem2CLI::PowerDeliveryClass::request (C++ function), 603
 Acroname::BrainStem2CLI::PowerDeliveryClass::requestStatus (C++ function), 604
 Acroname::BrainStem2CLI::PowerDeliveryClass::resetEntityToFactoryDefaults (C++ function), 604
 Acroname::BrainStem2CLI::PowerDeliveryClass::resetPowerDataObjectToDefault (C++ function), 600
 Acroname::BrainStem2CLI::PowerDeliveryClass::setFastRoleSwapCurrent (C++ function), 605
 Acroname::BrainStem2CLI::PowerDeliveryClass::setFlagMode (C++ function), 604
 Acroname::BrainStem2CLI::PowerDeliveryClass::setOverride (C++ function), 604
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPeakCurrentConfiguration (C++ function), 605
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerDataObject (C++ function), 600
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerDataObjectEnabled (C++ function), 601
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerRole (C++ function), 602
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerRolePreferred (C++ function), 602
 Acroname::BrainStem2CLI::PowerDeliveryClass::setRequestDataObject (C++ function), 601
 Acroname::BrainStem2CLI::RailClass (C++ class), 606
 Acroname::BrainStem2CLI::RailClass::~RailClass (C++ function), 606
 Acroname::BrainStem2CLI::RailClass::clearFaults (C++ function), 611
 Acroname::BrainStem2CLI::RailClass::getCurrent (C++ function), 606
 Acroname::BrainStem2CLI::RailClass::getCurrentLimit (C++ function), 606
 Acroname::BrainStem2CLI::RailClass::getCurrentSetpoint (C++ function), 606
 Acroname::BrainStem2CLI::RailClass::getEnable (C++ function), 607
 Acroname::BrainStem2CLI::RailClass::getKelvinSensingEnable (C++ function), 610
 Acroname::BrainStem2CLI::RailClass::getKelvinSensingState (C++ function), 610
 Acroname::BrainStem2CLI::RailClass::getOperationalMode (C++ function), 610
 Acroname::BrainStem2CLI::RailClass::getOperationalState (C++ function), 610
 Acroname::BrainStem2CLI::RailClass::getPower (C++ function), 608
 Acroname::BrainStem2CLI::RailClass::getPowerLimit (C++ function), 609
 Acroname::BrainStem2CLI::RailClass::getPowerSetpoint (C++ function), 609
 Acroname::BrainStem2CLI::RailClass::getResistance (C++ function), 609
 Acroname::BrainStem2CLI::RailClass::getResistanceSetpoint (C++ function), 609
 Acroname::BrainStem2CLI::RailClass::getTemperature (C++ function), 607
 Acroname::BrainStem2CLI::RailClass::getVoltage (C++ function), 607
 Acroname::BrainStem2CLI::RailClass::getVoltageMaxLimit (C++ function), 608
 Acroname::BrainStem2CLI::RailClass::getVoltageMinLimit (C++ function), 608
 Acroname::BrainStem2CLI::RailClass::getVoltageSetpoint (C++ function), 607
 Acroname::BrainStem2CLI::RailClass::RailClass (C++ function), 606
 Acroname::BrainStem2CLI::RailClass::setCurrentLimit (C++ function), 606
 Acroname::BrainStem2CLI::RailClass::setCurrentSetpoint (C++ function), 606
 Acroname::BrainStem2CLI::RailClass::setEnable (C++ function), 607
 Acroname::BrainStem2CLI::RailClass::setKelvinSensingEnable (C++ function), 610
 Acroname::BrainStem2CLI::RailClass::setOperationalMode (C++ function), 610
 Acroname::BrainStem2CLI::RailClass::setPowerLimit (C++ function), 609
 Acroname::BrainStem2CLI::RailClass::setPowerSetpoint (C++ function), 608
 Acroname::BrainStem2CLI::RailClass::setResistanceSetpoint (C++ function), 609
 Acroname::BrainStem2CLI::RailClass::setVoltageMaxLimit (C++ function), 608
 Acroname::BrainStem2CLI::RailClass::setVoltageMinLimit (C++ function), 608
 Acroname::BrainStem2CLI::RailClass::setVoltageSetpoint (C++ function), 607
 Acroname::BrainStem2CLI::RCServoClass (C++ class), 611
 Acroname::BrainStem2CLI::RCServoClass::~RCServoClass (C++ function), 611
 Acroname::BrainStem2CLI::RCServoClass::getEnable (C++ function), 611
 Acroname::BrainStem2CLI::RCServoClass::getPosition (C++ function), 611
 Acroname::BrainStem2CLI::RCServoClass::getReverse (C++ function), 612
 Acroname::BrainStem2CLI::RCServoClass::RCServoClass (C++ function), 611
 Acroname::BrainStem2CLI::RCServoClass::setEnable (C++ function), 611
 Acroname::BrainStem2CLI::RCServoClass::setPosition (C++ function), 611

Acroname::BrainStem2CLI::RCServoClass::setReverse (C++ function), 612
Acroname::BrainStem2CLI::RelayClass (C++ class), 612
Acroname::BrainStem2CLI::RelayClass::~RelayClass (C++ function), 612
Acroname::BrainStem2CLI::RelayClass::getEnable (C++ function), 612
Acroname::BrainStem2CLI::RelayClass::getVoltage (C++ function), 613
Acroname::BrainStem2CLI::RelayClass::RelayClass (C++ function), 612
Acroname::BrainStem2CLI::RelayClass::setEnabled (C++ function), 612
Acroname::BrainStem2CLI::SignalClass (C++ class), 613
Acroname::BrainStem2CLI::SignalClass::~SignalClass (C++ function), 613
Acroname::BrainStem2CLI::SignalClass::getEnable (C++ function), 613
Acroname::BrainStem2CLI::SignalClass::getInvert (C++ function), 614
Acroname::BrainStem2CLI::SignalClass::getT2Time (C++ function), 614
Acroname::BrainStem2CLI::SignalClass::getT3Time (C++ function), 614
Acroname::BrainStem2CLI::SignalClass::setEnabled (C++ function), 613
Acroname::BrainStem2CLI::SignalClass::setInvert (C++ function), 614
Acroname::BrainStem2CLI::SignalClass::setT2Time (C++ function), 614
Acroname::BrainStem2CLI::SignalClass::setT3Time (C++ function), 614
Acroname::BrainStem2CLI::SignalClass::SignalClass (C++ function), 613
Acroname::BrainStem2CLI::StoreClass (C++ class), 615
Acroname::BrainStem2CLI::StoreClass::~StoreClass (C++ function), 615
Acroname::BrainStem2CLI::StoreClass::getSlotCapacity (C++ function), 616
Acroname::BrainStem2CLI::StoreClass::getSlotSize (C++ function), 616
Acroname::BrainStem2CLI::StoreClass::getSlotState (C++ function), 615
Acroname::BrainStem2CLI::StoreClass::loadSlot (C++ function), 615
Acroname::BrainStem2CLI::StoreClass::slotDisable (C++ function), 616
Acroname::BrainStem2CLI::StoreClass::slotEnable (C++ function), 616
Acroname::BrainStem2CLI::StoreClass::StoreClass (C++ function), 615
Acroname::BrainStem2CLI::StoreClass::unloadSlot (C++ function), 615
Acroname::BrainStem2CLI::SystemClass (C++ class), 616
Acroname::BrainStem2CLI::SystemClass::~SystemClass (C++ function), 617
Acroname::BrainStem2CLI::SystemClass::getBootSlot (C++ function), 618
Acroname::BrainStem2CLI::SystemClass::getErrors (C++ function), 621
Acroname::BrainStem2CLI::SystemClass::getHardwareVersion (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::getHBInterval (C++ function), 618
Acroname::BrainStem2CLI::SystemClass::getInputCurrent (C++ function), 620
Acroname::BrainStem2CLI::SystemClass::getInputVoltage (C++ function), 620
Acroname::BrainStem2CLI::SystemClass::getLED (C++ function), 618
Acroname::BrainStem2CLI::SystemClass::getMaximumTemperature (C++ function), 620
Acroname::BrainStem2CLI::SystemClass::getMinimumTemperature (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::getModel (C++ function), 618
Acroname::BrainStem2CLI::SystemClass::getModule (C++ function), 617
Acroname::BrainStem2CLI::SystemClass::getModuleBaseAddress (C++ function), 617
Acroname::BrainStem2CLI::SystemClass::getModuleHardwareOffset (C++ function), 620
Acroname::BrainStem2CLI::SystemClass::getModuleSoftwareOffset (C++ function), 620
Acroname::BrainStem2CLI::SystemClass::getRouter (C++ function), 617
Acroname::BrainStem2CLI::SystemClass::getRouterAddressSetting (C++ function), 620
Acroname::BrainStem2CLI::SystemClass::getSerialNumber (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::getTemperature (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::getUptime (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::getVersion (C++ function), 618
Acroname::BrainStem2CLI::SystemClass::logEvents (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::reset (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::routeToMe (C++ function), 621
Acroname::BrainStem2CLI::SystemClass::save (C++ function), 619
Acroname::BrainStem2CLI::SystemClass::setBootSlot (C++ function), 618
Acroname::BrainStem2CLI::SystemClass::setHBInterval (C++ function), 617
Acroname::BrainStem2CLI::SystemClass::setLED (C++ function), 618
Acroname::BrainStem2CLI::SystemClass::setModuleSoftwareOffset (C++ function), 620
Acroname::BrainStem2CLI::SystemClass::setRouter (C++ function), 617
Acroname::BrainStem2CLI::SystemClass::SystemClass (C++ function), 617
Acroname::BrainStem2CLI::TemperatureClass (C++ class), 621
Acroname::BrainStem2CLI::TemperatureClass::~TemperatureClass (C++ function), 621
Acroname::BrainStem2CLI::TemperatureClass::getValue (C++ function), 621
Acroname::BrainStem2CLI::TemperatureClass::getValueMax (C++ function), 622
Acroname::BrainStem2CLI::TemperatureClass::getValueMin (C++ function), 621
Acroname::BrainStem2CLI::TemperatureClass::TemperatureClass (C++ function), 621
Acroname::BrainStem2CLI::TimerClass (C++ class), 622
Acroname::BrainStem2CLI::TimerClass::~TimerClass (C++ function), 622

Acraname::BrainStem2CLI::TimerClass::getExpiration (*C++ function*), 622
 Acraname::BrainStem2CLI::TimerClass::getMode (*C++ function*), 622
 Acraname::BrainStem2CLI::TimerClass::setExpiration (*C++ function*), 622
 Acraname::BrainStem2CLI::TimerClass::setMode (*C++ function*), 623
 Acraname::BrainStem2CLI::TimerClass::TimerClass (*C++ function*), 622
 Acraname::BrainStem2CLI::UARTClass (*C++ class*), 623
 Acraname::BrainStem2CLI::UARTClass::~UARTClass (*C++ function*), 623
 Acraname::BrainStem2CLI::UARTClass::getEnable (*C++ function*), 623
 Acraname::BrainStem2CLI::UARTClass::setEnable (*C++ function*), 623
 Acraname::BrainStem2CLI::UARTClass::UARTClass (*C++ function*), 623
 Acraname::BrainStem2CLI::UARTClass::UARTClass::getBaudRate (*C++ function*), 624
 Acraname::BrainStem2CLI::UARTClass::UARTClass::getProtocol (*C++ function*), 624
 Acraname::BrainStem2CLI::UARTClass::UARTClass::setBaudRate (*C++ function*), 623
 Acraname::BrainStem2CLI::UARTClass::UARTClass::setProtocol (*C++ function*), 624
 Acraname::BrainStem2CLI::USBCClass (*C++ class*), 624
 Acraname::BrainStem2CLI::USBCClass::~USBCClass (*C++ function*), 624
 Acraname::BrainStem2CLI::USBCClass::clearPortErrorStatus (*C++ function*), 626
 Acraname::BrainStem2CLI::USBCClass::getAltModeConfig (*C++ function*), 632
 Acraname::BrainStem2CLI::USBCClass::getCableFlip (*C++ function*), 631
 Acraname::BrainStem2CLI::USBCClass::getCC1Current (*C++ function*), 630
 Acraname::BrainStem2CLI::USBCClass::getCC1Enable (*C++ function*), 630
 Acraname::BrainStem2CLI::USBCClass::getCC1Voltage (*C++ function*), 631
 Acraname::BrainStem2CLI::USBCClass::getCC2Current (*C++ function*), 630
 Acraname::BrainStem2CLI::USBCClass::getCC2Enable (*C++ function*), 630
 Acraname::BrainStem2CLI::USBCClass::getCC2Voltage (*C++ function*), 631
 Acraname::BrainStem2CLI::USBCClass::getConnectMode (*C++ function*), 629
 Acraname::BrainStem2CLI::USBCClass::getDownstreamBoostMode (*C++ function*), 629
 Acraname::BrainStem2CLI::USBCClass::getDownstreamDataSpeed (*C++ function*), 629
 Acraname::BrainStem2CLI::USBCClass::getEnumerationDelay (*C++ function*), 627
 Acraname::BrainStem2CLI::USBCClass::getHubMode (*C++ function*), 626
 Acraname::BrainStem2CLI::USBCClass::getPortCurrent (*C++ function*), 626
 Acraname::BrainStem2CLI::USBCClass::getPortCurrentLimit (*C++ function*), 627
 Acraname::BrainStem2CLI::USBCClass::getPortError (*C++ function*), 628
 Acraname::BrainStem2CLI::USBCClass::getPortMode (*C++ function*), 628
 Acraname::BrainStem2CLI::USBCClass::getPortState (*C++ function*), 628
 Acraname::BrainStem2CLI::USBCClass::getPortVoltage (*C++ function*), 626
 Acraname::BrainStem2CLI::USBCClass::getSBU1Voltage (*C++ function*), 632
 Acraname::BrainStem2CLI::USBCClass::getSBU2Voltage (*C++ function*), 632
 Acraname::BrainStem2CLI::USBCClass::getSBUEnable (*C++ function*), 631
 Acraname::BrainStem2CLI::USBCClass::getUpstreamBoostMode (*C++ function*), 629
 Acraname::BrainStem2CLI::USBCClass::getUpstreamMode (*C++ function*), 626
 Acraname::BrainStem2CLI::USBCClass::getUpstreamState (*C++ function*), 627
 Acraname::BrainStem2CLI::USBCClass::setAltModeConfig (*C++ function*), 632
 Acraname::BrainStem2CLI::USBCClass::setCableFlip (*C++ function*), 631
 Acraname::BrainStem2CLI::USBCClass::setCC1Enable (*C++ function*), 630
 Acraname::BrainStem2CLI::USBCClass::setCC2Enable (*C++ function*), 630
 Acraname::BrainStem2CLI::USBCClass::setConnectMode (*C++ function*), 629
 Acraname::BrainStem2CLI::USBCClass::setDataDisable (*C++ function*), 625
 Acraname::BrainStem2CLI::USBCClass::setDataEnable (*C++ function*), 625
 Acraname::BrainStem2CLI::USBCClass::setDownstreamBoostMode (*C++ function*), 628
 Acraname::BrainStem2CLI::USBCClass::setEnumerationDelay (*C++ function*), 627
 Acraname::BrainStem2CLI::USBCClass::setHiSpeedDataDisable (*C++ function*), 625
 Acraname::BrainStem2CLI::USBCClass::setHiSpeedDataEnable (*C++ function*), 625
 Acraname::BrainStem2CLI::USBCClass::setHubMode (*C++ function*), 626
 Acraname::BrainStem2CLI::USBCClass::setPortCurrentLimit (*C++ function*), 627
 Acraname::BrainStem2CLI::USBCClass::setPortDisable (*C++ function*), 624
 Acraname::BrainStem2CLI::USBCClass::setPortEnable (*C++ function*), 624
 Acraname::BrainStem2CLI::USBCClass::setPortMode (*C++ function*), 627
 Acraname::BrainStem2CLI::USBCClass::setPowerDisable (*C++ function*), 626
 Acraname::BrainStem2CLI::USBCClass::setPowerEnable (*C++ function*), 625
 Acraname::BrainStem2CLI::USBCClass::setSBUEnable (*C++ function*), 631
 Acraname::BrainStem2CLI::USBCClass::setSuperSpeedDataDisable (*C++ function*), 625
 Acraname::BrainStem2CLI::USBCClass::setSuperSpeedDataEnable (*C++ function*), 625
 Acraname::BrainStem2CLI::USBCClass::setUpstreamBoostMode (*C++ function*), 628
 Acraname::BrainStem2CLI::USBCClass::setUpstreamMode (*C++ function*), 627
 Acraname::BrainStem2CLI::USBCClass::USBCClass (*C++ function*), 624
 Acraname::BrainStem2CLI::USBSysClass (*C++ class*), 632
 Acraname::BrainStem2CLI::USBSysClass::~USBSysClass (*C++ function*), 632

Acroname::BrainStem2CLI::USBSysClass::getDataRoleBehavior (C++ function), 635
Acroname::BrainStem2CLI::USBSysClass::getDataRoleBehaviorConfig (C++ function), 635
Acroname::BrainStem2CLI::USBSysClass::getDataRoleList (C++ function), 633
Acroname::BrainStem2CLI::USBSysClass::getEnabledList (C++ function), 633
Acroname::BrainStem2CLI::USBSysClass::getEnumerationDelay (C++ function), 633
Acroname::BrainStem2CLI::USBSysClass::getModeList (C++ function), 633
Acroname::BrainStem2CLI::USBSysClass::getPowerBehavior (C++ function), 634
Acroname::BrainStem2CLI::USBSysClass::getPowerBehaviorConfig (C++ function), 634
Acroname::BrainStem2CLI::USBSysClass::getStateList (C++ function), 634
Acroname::BrainStem2CLI::USBSysClass::getUpstream (C++ function), 632
Acroname::BrainStem2CLI::USBSysClass::resetEntityToFactoryDefaults (C++ function), 635
Acroname::BrainStem2CLI::USBSysClass::setDataRoleBehavior (C++ function), 635
Acroname::BrainStem2CLI::USBSysClass::setDataRoleBehaviorConfig (C++ function), 635
Acroname::BrainStem2CLI::USBSysClass::setEnabledList (C++ function), 633
Acroname::BrainStem2CLI::USBSysClass::setEnumerationDelay (C++ function), 633
Acroname::BrainStem2CLI::USBSysClass::setModeList (C++ function), 634
Acroname::BrainStem2CLI::USBSysClass::setPowerBehavior (C++ function), 634
Acroname::BrainStem2CLI::USBSysClass::setPowerBehaviorConfig (C++ function), 635
Acroname::BrainStem2CLI::USBSysClass::setUpstream (C++ function), 633
Acroname::BrainStem2CLI::USBSysClass::USBSysClass (C++ function), 632
Acroname::BrainStem::AnalogClass (C++ class), 411
Acroname::BrainStem::AnalogClass::~AnalogClass (C++ function), 412
Acroname::BrainStem::AnalogClass::AnalogClass (C++ function), 412
Acroname::BrainStem::AnalogClass::getBulkCaptureNumberOfSamples (C++ function), 414
Acroname::BrainStem::AnalogClass::getBulkCaptureSampleRate (C++ function), 414
Acroname::BrainStem::AnalogClass::getBulkCaptureState (C++ function), 414
Acroname::BrainStem::AnalogClass::getConfiguration (C++ function), 413
Acroname::BrainStem::AnalogClass::getEnable (C++ function), 412
Acroname::BrainStem::AnalogClass::getRange (C++ function), 412
Acroname::BrainStem::AnalogClass::getValue (C++ function), 412
Acroname::BrainStem::AnalogClass::getVoltage (C++ function), 412
Acroname::BrainStem::AnalogClass::init (C++ function), 412
Acroname::BrainStem::AnalogClass::initiateBulkCapture (C++ function), 414
Acroname::BrainStem::AnalogClass::setBulkCaptureNumberOfSamples (C++ function), 414
Acroname::BrainStem::AnalogClass::setBulkCaptureSampleRate (C++ function), 414
Acroname::BrainStem::AnalogClass::setConfiguration (C++ function), 413
Acroname::BrainStem::AnalogClass::setEnable (C++ function), 413
Acroname::BrainStem::AnalogClass::setRange (C++ function), 413
Acroname::BrainStem::AnalogClass::setValue (C++ function), 412
Acroname::BrainStem::AnalogClass::setVoltage (C++ function), 413
Acroname::BrainStem::AppClass (C++ class), 415
Acroname::BrainStem::AppClass::~AppClass (C++ function), 415
Acroname::BrainStem::AppClass::AppClass (C++ function), 415
Acroname::BrainStem::AppClass::execute (C++ function), 415
Acroname::BrainStem::AppClass::init (C++ function), 415
Acroname::BrainStem::ClockClass (C++ class), 416
Acroname::BrainStem::ClockClass::~ClockClass (C++ function), 416
Acroname::BrainStem::ClockClass::ClockClass (C++ function), 416
Acroname::BrainStem::ClockClass::getDay (C++ function), 416
Acroname::BrainStem::ClockClass::getHour (C++ function), 417
Acroname::BrainStem::ClockClass::getMinute (C++ function), 417
Acroname::BrainStem::ClockClass::getMonth (C++ function), 416
Acroname::BrainStem::ClockClass::getSecond (C++ function), 417
Acroname::BrainStem::ClockClass::getYear (C++ function), 416
Acroname::BrainStem::ClockClass::init (C++ function), 416
Acroname::BrainStem::ClockClass::setDay (C++ function), 417
Acroname::BrainStem::ClockClass::setHour (C++ function), 417
Acroname::BrainStem::ClockClass::setMinute (C++ function), 417
Acroname::BrainStem::ClockClass::setMonth (C++ function), 416
Acroname::BrainStem::ClockClass::setSecond (C++ function), 417
Acroname::BrainStem::ClockClass::setYear (C++ function), 416
Acroname::BrainStem::DigitalClass (C++ class), 418
Acroname::BrainStem::DigitalClass::~DigitalClass (C++ function), 418
Acroname::BrainStem::DigitalClass::DigitalClass (C++ function), 418
Acroname::BrainStem::DigitalClass::getConfiguration (C++ function), 418
Acroname::BrainStem::DigitalClass::getState (C++ function), 418
Acroname::BrainStem::DigitalClass::getStateAll (C++ function), 419
Acroname::BrainStem::DigitalClass::init (C++ function), 418

Acroname::BrainStem::DigitalClass::setConfiguration (C++ function), 418
 Acroname::BrainStem::DigitalClass::setState (C++ function), 418
 Acroname::BrainStem::DigitalClass::setStateAll (C++ function), 419
 Acroname::BrainStem::EntityClass (C++ class), 419
 Acroname::BrainStem::EntityClass::~EntityClass (C++ function), 420
 Acroname::BrainStem::EntityClass::callUEI (C++ function), 420
 Acroname::BrainStem::EntityClass::drainUEI (C++ function), 422
 Acroname::BrainStem::EntityClass::EntityClass (C++ function), 420
 Acroname::BrainStem::EntityClass::getIndex (C++ function), 422
 Acroname::BrainStem::EntityClass::getStreamStatus (C++ function), 423
 Acroname::BrainStem::EntityClass::getUEI16 (C++ function), 421
 Acroname::BrainStem::EntityClass::getUEI32 (C++ function), 421
 Acroname::BrainStem::EntityClass::getUEI8 (C++ function), 420
 Acroname::BrainStem::EntityClass::getUEIBytes (C++ function), 422
 Acroname::BrainStem::EntityClass::getUEIBytesCheck (C++ function), 422
 Acroname::BrainStem::EntityClass::getUEIBytesContinue (C++ function), 423
 Acroname::BrainStem::EntityClass::getUEIBytesSequence (C++ function), 423
 Acroname::BrainStem::EntityClass::init (C++ function), 420
 Acroname::BrainStem::EntityClass::registerOptionCallback (C++ function), 422
 Acroname::BrainStem::EntityClass::setStreamEnabled (C++ function), 422
 Acroname::BrainStem::EntityClass::setUEI16 (C++ function), 421
 Acroname::BrainStem::EntityClass::setUEI32 (C++ function), 421
 Acroname::BrainStem::EntityClass::setUEI8 (C++ function), 420
 Acroname::BrainStem::EntityClass::setUEIBytes (C++ function), 422
 Acroname::BrainStem::EntityClass::sUEIBytesFilter (C++ function), 423
 Acroname::BrainStem::EqualizerClass (C++ class), 424
 Acroname::BrainStem::EqualizerClass::~EqualizerClass (C++ function), 424
 Acroname::BrainStem::EqualizerClass::EqualizerClass (C++ function), 424
 Acroname::BrainStem::EqualizerClass::getReceiverConfig (C++ function), 424
 Acroname::BrainStem::EqualizerClass::getTransmitterConfig (C++ function), 424
 Acroname::BrainStem::EqualizerClass::init (C++ function), 424
 Acroname::BrainStem::EqualizerClass::setReceiverConfig (C++ function), 424
 Acroname::BrainStem::EqualizerClass::setTransmitterConfig (C++ function), 424
 Acroname::BrainStem::I2CClass (C++ class), 425
 Acroname::BrainStem::I2CClass::~I2CClass (C++ function), 425
 Acroname::BrainStem::I2CClass::getSpeed (C++ function), 426
 Acroname::BrainStem::I2CClass::I2CClass (C++ function), 425
 Acroname::BrainStem::I2CClass::init (C++ function), 425
 Acroname::BrainStem::I2CClass::read (C++ function), 425
 Acroname::BrainStem::I2CClass::setPullup (C++ function), 425
 Acroname::BrainStem::I2CClass::setSpeed (C++ function), 425
 Acroname::BrainStem::I2CClass::write (C++ function), 425
 Acroname::BrainStem::Link (C++ class), 426
 Acroname::BrainStem::Link::~Link (C++ function), 428
 Acroname::BrainStem::Link::connect (C++ function), 428
 Acroname::BrainStem::Link::disablePacketLog (C++ function), 435
 Acroname::BrainStem::Link::disconnect (C++ function), 429
 Acroname::BrainStem::Link::discoverAndConnect (C++ function), 428
 Acroname::BrainStem::Link::dropMatchingUEIPackets (C++ function), 431
 Acroname::BrainStem::Link::enablePacketLog (C++ function), 435
 Acroname::BrainStem::Link::enableStream (C++ function), 434
 Acroname::BrainStem::Link::filterActiveStreamKeys (C++ function), 435
 Acroname::BrainStem::Link::getFactoryData (C++ function), 435
 Acroname::BrainStem::Link::getLinkSpecifier (C++ function), 429
 Acroname::BrainStem::Link::getModuleAddress (C++ function), 429
 Acroname::BrainStem::Link::getName (C++ function), 429
 Acroname::BrainStem::Link::getStatus (C++ function), 429
 Acroname::BrainStem::Link::getStreamKeyElement (C++ function), 437
 Acroname::BrainStem::Link::getStreamPacketType (C++ function), 436
 Acroname::BrainStem::Link::getStreamSample (C++ function), 437
 Acroname::BrainStem::Link::getStreamStatus (C++ function), 435
 Acroname::BrainStem::Link::getStreamValue (C++ function), 434
 Acroname::BrainStem::Link::getTimestampParts (C++ function), 437
 Acroname::BrainStem::Link::isConnected (C++ function), 429
 Acroname::BrainStem::Link::isLinkStreaming (C++ function), 434
 Acroname::BrainStem::Link::isStreamPacket (C++ function), 437
 Acroname::BrainStem::Link::isStreamSample (C++ function), 437
 Acroname::BrainStem::Link::isSubindexType (C++ function), 437

Acroname::BrainStem::Link::Link (C++ function), 428
Acroname::BrainStem::Link::linkStreamFilter (C++ function), 438
Acroname::BrainStem::Link::loadStoreSlot (C++ function), 432
Acroname::BrainStem::Link::receivePacket (C++ function), 431
Acroname::BrainStem::Link::receiveUEI (C++ function), 430
Acroname::BrainStem::Link::registerStreamCallback (C++ function), 434
Acroname::BrainStem::Link::reset (C++ function), 429
Acroname::BrainStem::Link::sDiscover (C++ function), 436
Acroname::BrainStem::Link::sendPacket (C++ function), 431
Acroname::BrainStem::Link::sendUEI (C++ function), 429, 430
Acroname::BrainStem::Link::setFactoryData (C++ function), 435
Acroname::BrainStem::Link::setLinkSpecifier (C++ function), 429
Acroname::BrainStem::Link::sFindAll (C++ function), 436
Acroname::BrainStem::Link::storeSlotCapacity (C++ function), 433
Acroname::BrainStem::Link::storeSlotSize (C++ function), 433
Acroname::BrainStem::Link::STREAM_KEY (C++ enum), 427
Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_CMD (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_INDEX (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_MODULE_ADDRESS (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_OPTION (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_KEY::STREAM_KEY_SUBINDEX (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_KEY_t (C++ type), 427
Acroname::BrainStem::Link::STREAM_PACKET (C++ enum), 426
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_BYTES (C++ enumerator), 426
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_LAST (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_SUBINDEX_U16 (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_SUBINDEX_U32 (C++ enumerator), 427
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_SUBINDEX_U8 (C++ enumerator), 426
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_U16 (C++ enumerator), 426
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_U32 (C++ enumerator), 426
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_U8 (C++ enumerator), 426
Acroname::BrainStem::Link::STREAM_PACKET::kSTREAM_PACKET_UNKNOWN (C++ enumerator), 426
Acroname::BrainStem::Link::STREAM_PACKET_t (C++ type), 427
Acroname::BrainStem::Link::streamCallback_t (C++ type), 427
Acroname::BrainStem::Link::unloadStoreSlot (C++ function), 432
Acroname::BrainStem::Module (C++ class), 438
Acroname::BrainStem::Module::~~Module (C++ function), 438
Acroname::BrainStem::Module::classQuantity (C++ function), 441
Acroname::BrainStem::Module::connect (C++ function), 438
Acroname::BrainStem::Module::connectFromSpec (C++ function), 439
Acroname::BrainStem::Module::connectThroughLinkModule (C++ function), 439
Acroname::BrainStem::Module::debug (C++ function), 442
Acroname::BrainStem::Module::disconnect (C++ function), 440
Acroname::BrainStem::Module::discoverAndConnect (C++ function), 439
Acroname::BrainStem::Module::entityGroup (C++ function), 441
Acroname::BrainStem::Module::getLink (C++ function), 440
Acroname::BrainStem::Module::getLinkSpecifier (C++ function), 440
Acroname::BrainStem::Module::getModuleAddress (C++ function), 440
Acroname::BrainStem::Module::getStatus (C++ function), 439
Acroname::BrainStem::Module::hasUEI (C++ function), 440
Acroname::BrainStem::Module::isConnected (C++ function), 439
Acroname::BrainStem::Module::Module (C++ function), 438
Acroname::BrainStem::Module::reconnect (C++ function), 440
Acroname::BrainStem::Module::setModuleAddress (C++ function), 440
Acroname::BrainStem::Module::setNetworkingMode (C++ function), 442
Acroname::BrainStem::Module::subclassQuantity (C++ function), 441
Acroname::BrainStem::MuxClass (C++ class), 442
Acroname::BrainStem::MuxClass::~~MuxClass (C++ function), 443
Acroname::BrainStem::MuxClass::getChannel (C++ function), 443
Acroname::BrainStem::MuxClass::getChannelVoltage (C++ function), 443
Acroname::BrainStem::MuxClass::getConfiguration (C++ function), 443
Acroname::BrainStem::MuxClass::getEnable (C++ function), 443
Acroname::BrainStem::MuxClass::getSplitMode (C++ function), 444
Acroname::BrainStem::MuxClass::init (C++ function), 443
Acroname::BrainStem::MuxClass::MuxClass (C++ function), 443
Acroname::BrainStem::MuxClass::setChannel (C++ function), 443
Acroname::BrainStem::MuxClass::setConfiguration (C++ function), 444
Acroname::BrainStem::MuxClass::setEnable (C++ function), 443

Acraname::BrainStem::MuxClass::setSplitMode (C++ function), 444
 Acraname::BrainStem::PointerClass (C++ class), 444
 Acraname::BrainStem::PointerClass::~~PointerClass (C++ function), 445
 Acraname::BrainStem::PointerClass::getChar (C++ function), 446
 Acraname::BrainStem::PointerClass::getInt (C++ function), 446
 Acraname::BrainStem::PointerClass::getMode (C++ function), 445
 Acraname::BrainStem::PointerClass::getOffset (C++ function), 445
 Acraname::BrainStem::PointerClass::getShort (C++ function), 446
 Acraname::BrainStem::PointerClass::getTransferStore (C++ function), 445
 Acraname::BrainStem::PointerClass::init (C++ function), 445
 Acraname::BrainStem::PointerClass::initiateTransferFromStore (C++ function), 446
 Acraname::BrainStem::PointerClass::initiateTransferToStore (C++ function), 446
 Acraname::BrainStem::PointerClass::PointerClass (C++ function), 445
 Acraname::BrainStem::PointerClass::setChar (C++ function), 446
 Acraname::BrainStem::PointerClass::setInt (C++ function), 446
 Acraname::BrainStem::PointerClass::setMode (C++ function), 445
 Acraname::BrainStem::PointerClass::setOffset (C++ function), 445
 Acraname::BrainStem::PointerClass::setShort (C++ function), 446
 Acraname::BrainStem::PointerClass::setTransferStore (C++ function), 445
 Acraname::BrainStem::PortClass (C++ class), 447
 Acraname::BrainStem::PortClass::~~PortClass (C++ function), 447
 Acraname::BrainStem::PortClass::getAllocatedPower (C++ function), 454
 Acraname::BrainStem::PortClass::getAvailablePower (C++ function), 453
 Acraname::BrainStem::PortClass::getCC1Enabled (C++ function), 451
 Acraname::BrainStem::PortClass::getCC2Enabled (C++ function), 452
 Acraname::BrainStem::PortClass::getCCEnabled (C++ function), 451
 Acraname::BrainStem::PortClass::getCurrentLimit (C++ function), 453
 Acraname::BrainStem::PortClass::getCurrentLimitMode (C++ function), 453
 Acraname::BrainStem::PortClass::getDataEnabled (C++ function), 448
 Acraname::BrainStem::PortClass::getDataHS1Enabled (C++ function), 449
 Acraname::BrainStem::PortClass::getDataHS2Enabled (C++ function), 449
 Acraname::BrainStem::PortClass::getDataHSEnabled (C++ function), 448
 Acraname::BrainStem::PortClass::getDataHSRoutingBehavior (C++ function), 455
 Acraname::BrainStem::PortClass::getDataRole (C++ function), 450
 Acraname::BrainStem::PortClass::getDataSpeed (C++ function), 452
 Acraname::BrainStem::PortClass::getDataSS1Enabled (C++ function), 449
 Acraname::BrainStem::PortClass::getDataSS2Enabled (C++ function), 450
 Acraname::BrainStem::PortClass::getDataSSEnabled (C++ function), 449
 Acraname::BrainStem::PortClass::getDataSSRoutingBehavior (C++ function), 455
 Acraname::BrainStem::PortClass::getEnabled (C++ function), 448
 Acraname::BrainStem::PortClass::getErrors (C++ function), 453
 Acraname::BrainStem::PortClass::getHSBoost (C++ function), 456
 Acraname::BrainStem::PortClass::getMode (C++ function), 452
 Acraname::BrainStem::PortClass::getName (C++ function), 454
 Acraname::BrainStem::PortClass::getPowerEnabled (C++ function), 450
 Acraname::BrainStem::PortClass::getPowerLimit (C++ function), 454
 Acraname::BrainStem::PortClass::getPowerLimitMode (C++ function), 454
 Acraname::BrainStem::PortClass::getPowerMode (C++ function), 448
 Acraname::BrainStem::PortClass::getState (C++ function), 452
 Acraname::BrainStem::PortClass::getVbusAccumulatedPower (C++ function), 455
 Acraname::BrainStem::PortClass::getVbusCurrent (C++ function), 447
 Acraname::BrainStem::PortClass::getVbusVoltage (C++ function), 447
 Acraname::BrainStem::PortClass::getVconn1Enabled (C++ function), 451
 Acraname::BrainStem::PortClass::getVconn2Enabled (C++ function), 451
 Acraname::BrainStem::PortClass::getVconnAccumulatedPower (C++ function), 456
 Acraname::BrainStem::PortClass::getVconnCurrent (C++ function), 447
 Acraname::BrainStem::PortClass::getVconnEnabled (C++ function), 450
 Acraname::BrainStem::PortClass::getVconnVoltage (C++ function), 447
 Acraname::BrainStem::PortClass::getVoltageSetpoint (C++ function), 452
 Acraname::BrainStem::PortClass::PortClass (C++ function), 447
 Acraname::BrainStem::PortClass::resetEntityToFactoryDefaults (C++ function), 456
 Acraname::BrainStem::PortClass::resetVbusAccumulatedPower (C++ function), 456
 Acraname::BrainStem::PortClass::resetVconnAccumulatedPower (C++ function), 456
 Acraname::BrainStem::PortClass::setCC1Enabled (C++ function), 452
 Acraname::BrainStem::PortClass::setCC2Enabled (C++ function), 452
 Acraname::BrainStem::PortClass::setCCEnabled (C++ function), 451
 Acraname::BrainStem::PortClass::setCurrentLimit (C++ function), 453
 Acraname::BrainStem::PortClass::setCurrentLimitMode (C++ function), 453

Acroname::BrainStem::PortClass::setDataEnabled (C++ function), 448
Acroname::BrainStem::PortClass::setDataHS1Enabled (C++ function), 449
Acroname::BrainStem::PortClass::setDataHS2Enabled (C++ function), 449
Acroname::BrainStem::PortClass::setDataHSEnabled (C++ function), 448
Acroname::BrainStem::PortClass::setDataHSRoutingBehavior (C++ function), 455
Acroname::BrainStem::PortClass::setDataSS1Enabled (C++ function), 450
Acroname::BrainStem::PortClass::setDataSS2Enabled (C++ function), 450
Acroname::BrainStem::PortClass::setDataSSEnabled (C++ function), 449
Acroname::BrainStem::PortClass::setDataSSRoutingBehavior (C++ function), 455
Acroname::BrainStem::PortClass::setEnabled (C++ function), 448
Acroname::BrainStem::PortClass::setHSBoost (C++ function), 456
Acroname::BrainStem::PortClass::setMode (C++ function), 453
Acroname::BrainStem::PortClass::setName (C++ function), 455
Acroname::BrainStem::PortClass::setPowerEnabled (C++ function), 450
Acroname::BrainStem::PortClass::setPowerLimit (C++ function), 454
Acroname::BrainStem::PortClass::setPowerLimitMode (C++ function), 454
Acroname::BrainStem::PortClass::setPowerMode (C++ function), 448
Acroname::BrainStem::PortClass::setVconn1Enabled (C++ function), 451
Acroname::BrainStem::PortClass::setVconn2Enabled (C++ function), 451
Acroname::BrainStem::PortClass::setVconnEnabled (C++ function), 451
Acroname::BrainStem::PortClass::setVoltageSetpoint (C++ function), 452
Acroname::BrainStem::PowerDeliveryClass (C++ class), 456
Acroname::BrainStem::PowerDeliveryClass::~~PowerDeliveryClass (C++ function), 457
Acroname::BrainStem::PowerDeliveryClass::getCableCurrentMax (C++ function), 460
Acroname::BrainStem::PowerDeliveryClass::getCableOrientation (C++ function), 461
Acroname::BrainStem::PowerDeliveryClass::getCableSpeedMax (C++ function), 460
Acroname::BrainStem::PowerDeliveryClass::getCableType (C++ function), 461
Acroname::BrainStem::PowerDeliveryClass::getCableVoltageMax (C++ function), 460
Acroname::BrainStem::PowerDeliveryClass::getConnectionState (C++ function), 457
Acroname::BrainStem::PowerDeliveryClass::getFastRoleSwapCurrent (C++ function), 463
Acroname::BrainStem::PowerDeliveryClass::getFlagMode (C++ function), 462
Acroname::BrainStem::PowerDeliveryClass::getNumberOfPowerDataObjects (C++ function), 457
Acroname::BrainStem::PowerDeliveryClass::getOverride (C++ function), 462
Acroname::BrainStem::PowerDeliveryClass::getPeakCurrentConfiguration (C++ function), 462
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObject (C++ function), 457
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObjectEnabled (C++ function), 458
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObjectEnabledList (C++ function), 459
Acroname::BrainStem::PowerDeliveryClass::getPowerDataObjectList (C++ function), 458
Acroname::BrainStem::PowerDeliveryClass::getPowerRole (C++ function), 459
Acroname::BrainStem::PowerDeliveryClass::getPowerRolePreferred (C++ function), 460
Acroname::BrainStem::PowerDeliveryClass::getRequestDataObject (C++ function), 459
Acroname::BrainStem::PowerDeliveryClass::packDataObjectAttributes (C++ function), 463
Acroname::BrainStem::PowerDeliveryClass::PowerDeliveryClass (C++ function), 457
Acroname::BrainStem::PowerDeliveryClass::request (C++ function), 461
Acroname::BrainStem::PowerDeliveryClass::requestStatus (C++ function), 461
Acroname::BrainStem::PowerDeliveryClass::resetEntityToFactoryDefaults (C++ function), 462
Acroname::BrainStem::PowerDeliveryClass::resetPowerDataObjectToDefault (C++ function), 458
Acroname::BrainStem::PowerDeliveryClass::setFastRoleSwapCurrent (C++ function), 463
Acroname::BrainStem::PowerDeliveryClass::setFlagMode (C++ function), 462
Acroname::BrainStem::PowerDeliveryClass::setOverride (C++ function), 462
Acroname::BrainStem::PowerDeliveryClass::setPeakCurrentConfiguration (C++ function), 463
Acroname::BrainStem::PowerDeliveryClass::setPowerDataObject (C++ function), 457
Acroname::BrainStem::PowerDeliveryClass::setPowerDataObjectEnabled (C++ function), 458
Acroname::BrainStem::PowerDeliveryClass::setPowerRole (C++ function), 459
Acroname::BrainStem::PowerDeliveryClass::setPowerRolePreferred (C++ function), 460
Acroname::BrainStem::PowerDeliveryClass::setRequestDataObject (C++ function), 459
Acroname::BrainStem::PowerDeliveryClass::unpackDataObjectAttributes (C++ function), 464
Acroname::BrainStem::RailClass (C++ class), 464
Acroname::BrainStem::RailClass::~~RailClass (C++ function), 464
Acroname::BrainStem::RailClass::clearFaults (C++ function), 469
Acroname::BrainStem::RailClass::getCurrent (C++ function), 464
Acroname::BrainStem::RailClass::getCurrentLimit (C++ function), 465
Acroname::BrainStem::RailClass::getCurrentSetpoint (C++ function), 465
Acroname::BrainStem::RailClass::getEnable (C++ function), 465
Acroname::BrainStem::RailClass::getKelvinSensingEnable (C++ function), 468
Acroname::BrainStem::RailClass::getKelvinSensingState (C++ function), 468
Acroname::BrainStem::RailClass::getOperationalMode (C++ function), 469
Acroname::BrainStem::RailClass::getOperationalState (C++ function), 469

Acroname::BrainStem::RailClass::getPower (C++ function), 467
 Acroname::BrainStem::RailClass::getPowerLimit (C++ function), 467
 Acroname::BrainStem::RailClass::getPowerSetpoint (C++ function), 467
 Acroname::BrainStem::RailClass::getResistance (C++ function), 468
 Acroname::BrainStem::RailClass::getResistanceSetpoint (C++ function), 468
 Acroname::BrainStem::RailClass::getTemperature (C++ function), 465
 Acroname::BrainStem::RailClass::getVoltage (C++ function), 466
 Acroname::BrainStem::RailClass::getVoltageMaxLimit (C++ function), 467
 Acroname::BrainStem::RailClass::getVoltageMinLimit (C++ function), 466
 Acroname::BrainStem::RailClass::getVoltageSetpoint (C++ function), 466
 Acroname::BrainStem::RailClass::init (C++ function), 464
 Acroname::BrainStem::RailClass::RailClass (C++ function), 464
 Acroname::BrainStem::RailClass::setCurrentLimit (C++ function), 465
 Acroname::BrainStem::RailClass::setCurrentSetpoint (C++ function), 464
 Acroname::BrainStem::RailClass::setEnabled (C++ function), 465
 Acroname::BrainStem::RailClass::setKelvinSensingEnable (C++ function), 468
 Acroname::BrainStem::RailClass::setOperationalMode (C++ function), 469
 Acroname::BrainStem::RailClass::setPowerLimit (C++ function), 467
 Acroname::BrainStem::RailClass::setPowerSetpoint (C++ function), 467
 Acroname::BrainStem::RailClass::setResistanceSetpoint (C++ function), 468
 Acroname::BrainStem::RailClass::setVoltageMaxLimit (C++ function), 466
 Acroname::BrainStem::RailClass::setVoltageMinLimit (C++ function), 466
 Acroname::BrainStem::RailClass::setVoltageSetpoint (C++ function), 466
 Acroname::BrainStem::RCServoClass (C++ class), 469
 Acroname::BrainStem::RCServoClass::~RCServoClass (C++ function), 470
 Acroname::BrainStem::RCServoClass::getEnable (C++ function), 470
 Acroname::BrainStem::RCServoClass::getPosition (C++ function), 470
 Acroname::BrainStem::RCServoClass::getReverse (C++ function), 470
 Acroname::BrainStem::RCServoClass::init (C++ function), 470
 Acroname::BrainStem::RCServoClass::RCServoClass (C++ function), 470
 Acroname::BrainStem::RCServoClass::setEnabled (C++ function), 470
 Acroname::BrainStem::RCServoClass::setPosition (C++ function), 470
 Acroname::BrainStem::RCServoClass::setReverse (C++ function), 470
 Acroname::BrainStem::RelayClass (C++ class), 471
 Acroname::BrainStem::RelayClass::~RelayClass (C++ function), 471
 Acroname::BrainStem::RelayClass::getEnable (C++ function), 471
 Acroname::BrainStem::RelayClass::getVoltage (C++ function), 471
 Acroname::BrainStem::RelayClass::init (C++ function), 471
 Acroname::BrainStem::RelayClass::RelayClass (C++ function), 471
 Acroname::BrainStem::RelayClass::setEnabled (C++ function), 471
 Acroname::BrainStem::SignalClass (C++ class), 472
 Acroname::BrainStem::SignalClass::~SignalClass (C++ function), 472
 Acroname::BrainStem::SignalClass::getEnable (C++ function), 472
 Acroname::BrainStem::SignalClass::getInvert (C++ function), 472
 Acroname::BrainStem::SignalClass::getT2Time (C++ function), 473
 Acroname::BrainStem::SignalClass::getT3Time (C++ function), 473
 Acroname::BrainStem::SignalClass::init (C++ function), 472
 Acroname::BrainStem::SignalClass::setEnabled (C++ function), 472
 Acroname::BrainStem::SignalClass::setInvert (C++ function), 472
 Acroname::BrainStem::SignalClass::setT2Time (C++ function), 473
 Acroname::BrainStem::SignalClass::setT3Time (C++ function), 473
 Acroname::BrainStem::SignalClass::SignalClass (C++ function), 472
 Acroname::BrainStem::StoreClass (C++ class), 473
 Acroname::BrainStem::StoreClass::~StoreClass (C++ function), 474
 Acroname::BrainStem::StoreClass::getSlotCapacity (C++ function), 475
 Acroname::BrainStem::StoreClass::getSlotLocked (C++ function), 475
 Acroname::BrainStem::StoreClass::getSlotSize (C++ function), 475
 Acroname::BrainStem::StoreClass::getSlotState (C++ function), 474
 Acroname::BrainStem::StoreClass::init (C++ function), 474
 Acroname::BrainStem::StoreClass::loadSlot (C++ function), 474
 Acroname::BrainStem::StoreClass::setSlotLocked (C++ function), 475
 Acroname::BrainStem::StoreClass::slotDisable (C++ function), 474
 Acroname::BrainStem::StoreClass::slotEnable (C++ function), 474
 Acroname::BrainStem::StoreClass::StoreClass (C++ function), 474
 Acroname::BrainStem::StoreClass::unloadSlot (C++ function), 474
 Acroname::BrainStem::SystemClass (C++ class), 475
 Acroname::BrainStem::SystemClass::~SystemClass (C++ function), 476
 Acroname::BrainStem::SystemClass::getBootSlot (C++ function), 477

Acroname::BrainStem::SystemClass::getErrors (C++ function), 482
Acroname::BrainStem::SystemClass::getHardwareVersion (C++ function), 478
Acroname::BrainStem::SystemClass::getHBInterval (C++ function), 477
Acroname::BrainStem::SystemClass::getInputCurrent (C++ function), 479
Acroname::BrainStem::SystemClass::getInputPowerBehavior (C++ function), 481
Acroname::BrainStem::SystemClass::getInputPowerBehaviorConfig (C++ function), 481
Acroname::BrainStem::SystemClass::getInputPowerSource (C++ function), 481
Acroname::BrainStem::SystemClass::getInputVoltage (C++ function), 479
Acroname::BrainStem::SystemClass::getLED (C++ function), 477
Acroname::BrainStem::SystemClass::getLinkInterface (C++ function), 482
Acroname::BrainStem::SystemClass::getMaximumTemperature (C++ function), 479
Acroname::BrainStem::SystemClass::getMinimumTemperature (C++ function), 478
Acroname::BrainStem::SystemClass::getModel (C++ function), 477
Acroname::BrainStem::SystemClass::getModule (C++ function), 476
Acroname::BrainStem::SystemClass::getModuleBaseAddress (C++ function), 476
Acroname::BrainStem::SystemClass::getModuleHardwareOffset (C++ function), 479
Acroname::BrainStem::SystemClass::getModuleSoftwareOffset (C++ function), 479
Acroname::BrainStem::SystemClass::getName (C++ function), 481
Acroname::BrainStem::SystemClass::getPowerLimit (C++ function), 480
Acroname::BrainStem::SystemClass::getPowerLimitMax (C++ function), 480
Acroname::BrainStem::SystemClass::getPowerLimitState (C++ function), 480
Acroname::BrainStem::SystemClass::getRouter (C++ function), 476
Acroname::BrainStem::SystemClass::getRouterAddressSetting (C++ function), 479
Acroname::BrainStem::SystemClass::getSerialNumber (C++ function), 478
Acroname::BrainStem::SystemClass::getTemperature (C++ function), 478
Acroname::BrainStem::SystemClass::getUnregulatedCurrent (C++ function), 481
Acroname::BrainStem::SystemClass::getUnregulatedVoltage (C++ function), 480
Acroname::BrainStem::SystemClass::getUptime (C++ function), 478
Acroname::BrainStem::SystemClass::getVersion (C++ function), 477
Acroname::BrainStem::SystemClass::init (C++ function), 476
Acroname::BrainStem::SystemClass::logEvents (C++ function), 478
Acroname::BrainStem::SystemClass::reset (C++ function), 478
Acroname::BrainStem::SystemClass::resetDeviceToFactoryDefaults (C++ function), 482
Acroname::BrainStem::SystemClass::resetEntityToFactoryDefaults (C++ function), 482
Acroname::BrainStem::SystemClass::routeToMe (C++ function), 480
Acroname::BrainStem::SystemClass::save (C++ function), 478
Acroname::BrainStem::SystemClass::setBootSlot (C++ function), 477
Acroname::BrainStem::SystemClass::setHBInterval (C++ function), 476
Acroname::BrainStem::SystemClass::setInputPowerBehavior (C++ function), 481
Acroname::BrainStem::SystemClass::setInputPowerBehaviorConfig (C++ function), 481
Acroname::BrainStem::SystemClass::setLED (C++ function), 477
Acroname::BrainStem::SystemClass::setLinkInterface (C++ function), 482
Acroname::BrainStem::SystemClass::setModuleSoftwareOffset (C++ function), 479
Acroname::BrainStem::SystemClass::setName (C++ function), 482
Acroname::BrainStem::SystemClass::setPowerLimitMax (C++ function), 480
Acroname::BrainStem::SystemClass::setRouter (C++ function), 476
Acroname::BrainStem::SystemClass::SystemClass (C++ function), 476
Acroname::BrainStem::TemperatureClass (C++ class), 483
Acroname::BrainStem::TemperatureClass::~TemperatureClass (C++ function), 483
Acroname::BrainStem::TemperatureClass::getValue (C++ function), 483
Acroname::BrainStem::TemperatureClass::getValueMax (C++ function), 483
Acroname::BrainStem::TemperatureClass::getValueMin (C++ function), 483
Acroname::BrainStem::TemperatureClass::init (C++ function), 483
Acroname::BrainStem::TemperatureClass::resetEntityToFactoryDefaults (C++ function), 483
Acroname::BrainStem::TemperatureClass::TemperatureClass (C++ function), 483
Acroname::BrainStem::TimerClass (C++ class), 484
Acroname::BrainStem::TimerClass::~TimerClass (C++ function), 484
Acroname::BrainStem::TimerClass::getExpiration (C++ function), 484
Acroname::BrainStem::TimerClass::getMode (C++ function), 484
Acroname::BrainStem::TimerClass::init (C++ function), 484
Acroname::BrainStem::TimerClass::setExpiration (C++ function), 484
Acroname::BrainStem::TimerClass::TimerClass (C++ function), 484
Acroname::BrainStem::TimerClass::TimerClass (C++ function), 484
Acroname::BrainStem::UARTClass (C++ class), 485
Acroname::BrainStem::UARTClass::~UARTClass (C++ function), 485
Acroname::BrainStem::UARTClass::getBaudRate (C++ function), 485
Acroname::BrainStem::UARTClass::getEnable (C++ function), 485
Acroname::BrainStem::UARTClass::getProtocol (C++ function), 486

Acroname::BrainStem::UARTClass::init (C++ function), 485
 Acroname::BrainStem::UARTClass::setBaudRate (C++ function), 485
 Acroname::BrainStem::UARTClass::setEnabled (C++ function), 485
 Acroname::BrainStem::UARTClass::setProtocol (C++ function), 485
 Acroname::BrainStem::UARTClass::UARTClass (C++ function), 485
 Acroname::BrainStem::USBCClass (C++ class), 486
 Acroname::BrainStem::USBCClass::~USBCClass (C++ function), 486
 Acroname::BrainStem::USBCClass::clearPortErrorStatus (C++ function), 488
 Acroname::BrainStem::USBCClass::getAltModeConfig (C++ function), 493
 Acroname::BrainStem::USBCClass::getCableFlip (C++ function), 493
 Acroname::BrainStem::USBCClass::getCC1Current (C++ function), 492
 Acroname::BrainStem::USBCClass::getCC1Enable (C++ function), 491
 Acroname::BrainStem::USBCClass::getCC1Voltage (C++ function), 492
 Acroname::BrainStem::USBCClass::getCC2Current (C++ function), 492
 Acroname::BrainStem::USBCClass::getCC2Enable (C++ function), 492
 Acroname::BrainStem::USBCClass::getCC2Voltage (C++ function), 492
 Acroname::BrainStem::USBCClass::getConnectMode (C++ function), 491
 Acroname::BrainStem::USBCClass::getDownstreamBoostMode (C++ function), 491
 Acroname::BrainStem::USBCClass::getDownstreamDataSpeed (C++ function), 491
 Acroname::BrainStem::USBCClass::getEnumerationDelay (C++ function), 489
 Acroname::BrainStem::USBCClass::getHubMode (C++ function), 488
 Acroname::BrainStem::USBCClass::getPortCurrent (C++ function), 487
 Acroname::BrainStem::USBCClass::getPortCurrentLimit (C++ function), 489
 Acroname::BrainStem::USBCClass::getPortError (C++ function), 490
 Acroname::BrainStem::USBCClass::getPortMode (C++ function), 489
 Acroname::BrainStem::USBCClass::getPortState (C++ function), 490
 Acroname::BrainStem::USBCClass::getPortVoltage (C++ function), 488
 Acroname::BrainStem::USBCClass::getSBU1Voltage (C++ function), 493
 Acroname::BrainStem::USBCClass::getSBU2Voltage (C++ function), 494
 Acroname::BrainStem::USBCClass::getSBUEnable (C++ function), 493
 Acroname::BrainStem::USBCClass::getUpstreamBoostMode (C++ function), 490
 Acroname::BrainStem::USBCClass::getUpstreamMode (C++ function), 488
 Acroname::BrainStem::USBCClass::getUpstreamState (C++ function), 488
 Acroname::BrainStem::USBCClass::init (C++ function), 486
 Acroname::BrainStem::USBCClass::setAltModeConfig (C++ function), 493
 Acroname::BrainStem::USBCClass::setCableFlip (C++ function), 493
 Acroname::BrainStem::USBCClass::setCC1Enable (C++ function), 491
 Acroname::BrainStem::USBCClass::setCC2Enable (C++ function), 492
 Acroname::BrainStem::USBCClass::setConnectMode (C++ function), 491
 Acroname::BrainStem::USBCClass::setDataDisable (C++ function), 487
 Acroname::BrainStem::USBCClass::setDataEnable (C++ function), 486
 Acroname::BrainStem::USBCClass::setDownstreamBoostMode (C++ function), 490
 Acroname::BrainStem::USBCClass::setEnumerationDelay (C++ function), 489
 Acroname::BrainStem::USBCClass::setHiSpeedDataDisable (C++ function), 487
 Acroname::BrainStem::USBCClass::setHiSpeedDataEnable (C++ function), 487
 Acroname::BrainStem::USBCClass::setHubMode (C++ function), 488
 Acroname::BrainStem::USBCClass::setPortCurrentLimit (C++ function), 489
 Acroname::BrainStem::USBCClass::setPortDisable (C++ function), 486
 Acroname::BrainStem::USBCClass::setPortEnable (C++ function), 486
 Acroname::BrainStem::USBCClass::setPortMode (C++ function), 489
 Acroname::BrainStem::USBCClass::setPowerDisable (C++ function), 487
 Acroname::BrainStem::USBCClass::setPowerEnable (C++ function), 487
 Acroname::BrainStem::USBCClass::setSBUEnable (C++ function), 493
 Acroname::BrainStem::USBCClass::setSuperSpeedDataDisable (C++ function), 487
 Acroname::BrainStem::USBCClass::setSuperSpeedDataEnable (C++ function), 487
 Acroname::BrainStem::USBCClass::setUpstreamBoostMode (C++ function), 490
 Acroname::BrainStem::USBCClass::setUpstreamMode (C++ function), 488
 Acroname::BrainStem::USBCClass::USBCClass (C++ function), 486
 Acroname::BrainStem::USBSysSystemClass (C++ class), 494
 Acroname::BrainStem::USBSysSystemClass::~USBSysSystemClass (C++ function), 494
 Acroname::BrainStem::USBSysSystemClass::getDataRoleBehavior (C++ function), 496
 Acroname::BrainStem::USBSysSystemClass::getDataRoleBehaviorConfig (C++ function), 497
 Acroname::BrainStem::USBSysSystemClass::getDataRoleList (C++ function), 495
 Acroname::BrainStem::USBSysSystemClass::getEnabledList (C++ function), 495
 Acroname::BrainStem::USBSysSystemClass::getEnumerationDelay (C++ function), 494
 Acroname::BrainStem::USBSysSystemClass::getModeList (C++ function), 495
 Acroname::BrainStem::USBSysSystemClass::getPowerBehavior (C++ function), 496
 Acroname::BrainStem::USBSysSystemClass::getPowerBehaviorConfig (C++ function), 496

Acroname::BrainStem::USBSysClass::getSelectorMode (C++ function), 497
Acroname::BrainStem::USBSysClass::getStateList (C++ function), 496
Acroname::BrainStem::USBSysClass::getUpstream (C++ function), 494
Acroname::BrainStem::USBSysClass::init (C++ function), 494
Acroname::BrainStem::USBSysClass::resetEntityToFactoryDefaults (C++ function), 497
Acroname::BrainStem::USBSysClass::setDataRoleBehavior (C++ function), 497
Acroname::BrainStem::USBSysClass::setDataRoleBehaviorConfig (C++ function), 497
Acroname::BrainStem::USBSysClass::setEnabledList (C++ function), 495
Acroname::BrainStem::USBSysClass::setEnumerationDelay (C++ function), 495
Acroname::BrainStem::USBSysClass::setModeList (C++ function), 495
Acroname::BrainStem::USBSysClass::setPowerBehavior (C++ function), 496
Acroname::BrainStem::USBSysClass::setPowerBehaviorConfig (C++ function), 496
Acroname::BrainStem::USBSysClass::setSelectorMode (C++ function), 497
Acroname::BrainStem::USBSysClass::setUpstream (C++ function), 494
Acroname::BrainStem::USBSysClass::USBSysClass (C++ function), 494
address (brainstem.module.Module property), 314
aDefs_GetModelName (C++ function), 501
aDiscovery_EnumerateModules (C++ function), 503
aDiscovery_FindFirstModule (C++ function), 503
aDiscovery_FindModule (C++ function), 503
aDiscoveryModuleFoundProc (C++ type), 502
aErr (C++ enum), 504
aErr::aErrAsyncReturn (C++ enumerator), 506
aErr::aErrBusy (C++ enumerator), 504
aErr::aErrCancel (C++ enumerator), 506
aErr::aErrConfiguration (C++ enumerator), 505
aErr::aErrConnection (C++ enumerator), 506
aErr::aErrDuplicate (C++ enumerator), 506
aErr::aErrEOF (C++ enumerator), 505
aErr::aErrFileNameLength (C++ enumerator), 504
aErr::aErrIndexRange (C++ enumerator), 506
aErr::aErrInitialization (C++ enumerator), 505
aErr::aErrInvalidEntity (C++ enumerator), 506
aErr::aErrInvalidOption (C++ enumerator), 506
aErr::aErrIO (C++ enumerator), 505
aErr::aErrMedia (C++ enumerator), 506
aErr::aErrMemory (C++ enumerator), 504
aErr::aErrMode (C++ enumerator), 505
aErr::aErrNone (C++ enumerator), 504
aErr::aErrNotFound (C++ enumerator), 504
aErr::aErrNotReady (C++ enumerator), 505
aErr::aErrOverrun (C++ enumerator), 505
aErr::aErrPacket (C++ enumerator), 506
aErr::aErrParam (C++ enumerator), 504
aErr::aErrParse (C++ enumerator), 505
aErr::aErrPermission (C++ enumerator), 505
aErr::aErrRange (C++ enumerator), 505
aErr::aErrRead (C++ enumerator), 505
aErr::aErrResource (C++ enumerator), 506
aErr::aErrShortCommand (C++ enumerator), 506
aErr::aErrSize (C++ enumerator), 505
aErr::aErrStreamStale (C++ enumerator), 506
aErr::aErrTimeout (C++ enumerator), 505
aErr::aErrUnimplemented (C++ enumerator), 506
aErr::aErrUnknown (C++ enumerator), 506
aErr::aErrVersion (C++ enumerator), 505
aErr::aErrWrite (C++ enumerator), 505
aError_GetErrorText (C++ function), 506
aFile_Close (C++ function), 508
aFile_Delete (C++ function), 509
aFile_Exists (C++ function), 507
aFile_GetSize (C++ function), 509
aFile_Open (C++ function), 508
aFile_Read (C++ function), 508
aFile_Seek (C++ function), 509
aFile_Write (C++ function), 508
aFileMode (C++ enum), 507
aFileMode::aFileModeAppend (C++ enumerator), 507

aFileMode::aFileModeReadOnly (*C++ enumerator*), 507
aFileMode::aFileModeUnknown (*C++ enumerator*), 507
aFileMode::aFileModeWriteOnly (*C++ enumerator*), 507
aFileRef (*C++ type*), 507
aFileSeekMode (*C++ enum*), 507
aFileSeekMode::aSeekCurrent (*C++ enumerator*), 507
aFileSeekMode::aSeekEnd (*C++ enumerator*), 507
aFileSeekMode::aSeekStart (*C++ enumerator*), 507
aLIBEXPORT (*C macro*), 500
aLink_AwaitFirst (*C++ function*), 513
aLink_AwaitPacket (*C++ function*), 512
aLink_CreateTCPIP (*C++ function*), 511
aLink_CreateUSB (*C++ function*), 511
aLink_Destroy (*C++ function*), 511
aLink_DrainPackets (*C++ function*), 513
aLink_GetFirst (*C++ function*), 512
aLink_GetPacket (*C++ function*), 512
aLink_GetStatus (*C++ function*), 512
aLink_PutPacket (*C++ function*), 513
aLink_Reset (*C++ function*), 512
aLinkRef (*C++ type*), 510
aLinkSpec_Create (*C++ function*), 504
aLinkSpec_Destroy (*C++ function*), 504
aMemPtr (*C macro*), 501
aMTM_ETHERSTEM_BULK_CAPTURE_MAX_HZ (*C macro*), 397
aMTM_ETHERSTEM_BULK_CAPTURE_MIN_HZ (*C macro*), 397
aMTM_ETHERSTEM_MODULE_BASE_ADDRESS (*C macro*), 396
aMTM_ETHERSTEM_NUM_A2D (*C macro*), 397
aMTM_ETHERSTEM_NUM_APPS (*C macro*), 397
aMTM_ETHERSTEM_NUM_CLOCK (*C macro*), 397
aMTM_ETHERSTEM_NUM_DIG (*C macro*), 397
aMTM_ETHERSTEM_NUM_I2C (*C macro*), 397
aMTM_ETHERSTEM_NUM_INPUT_SIGNALS (*C macro*), 397
aMTM_ETHERSTEM_NUM_INTERNAL_SLOTS (*C macro*), 396
aMTM_ETHERSTEM_NUM_OUTPUT_SIGNALS (*C macro*), 397
aMTM_ETHERSTEM_NUM_POINTERS (*C macro*), 397
aMTM_ETHERSTEM_NUM_RAM_SLOTS (*C macro*), 396, 397
aMTM_ETHERSTEM_NUM_SD_SLOTS (*C macro*), 397
aMTM_ETHERSTEM_NUM_SERVOS (*C macro*), 397
aMTM_ETHERSTEM_NUM_SIGNALS (*C macro*), 397
aMTM_ETHERSTEM_NUM_STORES (*C macro*), 396
aMTM_ETHERSTEM_NUM_TIMERS (*C macro*), 398
aMTM_STEM_BULK_CAPTURE_MAX_HZ (*C macro*), 410
aMTM_STEM_BULK_CAPTURE_MIN_HZ (*C macro*), 410
aMTM_STEM_MODULE_BASE_ADDRESS (*C macro*), 410
aMTM_STEM_NUM_A2D (*C macro*), 410
aMTM_STEM_NUM_APPS (*C macro*), 410
aMTM_STEM_NUM_CLOCK (*C macro*), 410
aMTM_STEM_NUM_DIG (*C macro*), 410
aMTM_STEM_NUM_I2C (*C macro*), 410
aMTM_STEM_NUM_INPUT_SIGNALS (*C macro*), 411
aMTM_STEM_NUM_INTERNAL_SLOTS (*C macro*), 411
aMTM_STEM_NUM_OUTPUT_SIGNALS (*C macro*), 411
aMTM_STEM_NUM_POINTERS (*C macro*), 410
aMTM_STEM_NUM_RAM_SLOTS (*C macro*), 411
aMTM_STEM_NUM_SD_SLOTS (*C macro*), 411
aMTM_STEM_NUM_SERVOS (*C macro*), 411
aMTM_STEM_NUM_SIGNALS (*C macro*), 411
aMTM_STEM_NUM_STORES (*C macro*), 411
aMTM_STEM_NUM_TIMERS (*C macro*), 411
aMTM_USBSTEM_BULK_CAPTURE_MAX_HZ (*C macro*), 408
aMTM_USBSTEM_BULK_CAPTURE_MIN_HZ (*C macro*), 408
aMTM_USBSTEM_MODULE_BASE_ADDRESS (*C macro*), 408
aMTM_USBSTEM_NUM_A2D (*C macro*), 408
aMTM_USBSTEM_NUM_APPS (*C macro*), 408
aMTM_USBSTEM_NUM_CLOCK (*C macro*), 408
aMTM_USBSTEM_NUM_DIG (*C macro*), 408
aMTM_USBSTEM_NUM_I2C (*C macro*), 408

aMTM_USBSTEM_NUM_INPUT_SIGNALS (*C macro*), 408
aMTM_USBSTEM_NUM_INTERNAL_SLOTS (*C macro*), 409
aMTM_USBSTEM_NUM_OUTPUT_SIGNALS (*C macro*), 408
aMTM_USBSTEM_NUM_POINTERS (*C macro*), 408
aMTM_USBSTEM_NUM_RAM_SLOTS (*C macro*), 409
aMTM_USBSTEM_NUM_SD_SLOTS (*C macro*), 409
aMTM_USBSTEM_NUM_SERVOS (*C macro*), 408
aMTM_USBSTEM_NUM_SIGNALS (*C macro*), 408
aMTM_USBSTEM_NUM_STORES (*C macro*), 409
aMTM_USBSTEM_NUM_TIMERS (*C macro*), 409
aMTMDAQ2 (*C++ class*), 394
aMTMDAQ2::analog (*C++ member*), 394
aMTMDAQ2::app (*C++ member*), 394
aMTMDAQ2::digital (*C++ member*), 394
aMTMDAQ2::getDifferentialInputRanges (*C++ function*), 395
aMTMDAQ2::getOutputRanges (*C++ function*), 395
aMTMDAQ2::getSingleEndedInputRanges (*C++ function*), 395
aMTMDAQ2::i2c (*C++ member*), 394
aMTMDAQ2::pointer (*C++ member*), 394
aMTMDAQ2::store (*C++ member*), 394
aMTMDAQ2::system (*C++ member*), 394
aMTMDAQ2::timer (*C++ member*), 394
aMTMDAQ2_BULK_CAPTURE_MAX_HZ (*C macro*), 395
aMTMDAQ2_BULK_CAPTURE_MIN_HZ (*C macro*), 395
aMTMDAQ2_MODULE_BASE_ADDRESS (*C macro*), 395
aMTMDAQ2_NUM_ANALOG_INPUTS (*C macro*), 395
aMTMDAQ2_NUM_ANALOG_OUTPUTS (*C macro*), 395
aMTMDAQ2_NUM_ANALOGS (*C macro*), 395
aMTMDAQ2_NUM_APPS (*C macro*), 395
aMTMDAQ2_NUM_DIGITALS (*C macro*), 395
aMTMDAQ2_NUM_I2C (*C macro*), 395
aMTMDAQ2_NUM_INTERNAL_SLOTS (*C macro*), 396
aMTMDAQ2_NUM_POINTERS (*C macro*), 396
aMTMDAQ2_NUM_RAM_SLOTS (*C macro*), 396
aMTMDAQ2_NUM_STORES (*C macro*), 396
aMTMDAQ2_NUM_TIMERS (*C macro*), 396
aMTMEtherStem (*C++ class*), 396
aMTMIOSerial (*C++ class*), 398
aMTMIOSerial::app (*C++ member*), 398
aMTMIOSerial::digital (*C++ member*), 398
aMTMIOSerial::i2c (*C++ member*), 398
aMTMIOSerial::pointer (*C++ member*), 398
aMTMIOSerial::rail (*C++ member*), 398
aMTMIOSerial::servo (*C++ member*), 398
aMTMIOSerial::signal (*C++ member*), 398
aMTMIOSerial::store (*C++ member*), 398
aMTMIOSerial::system (*C++ member*), 398
aMTMIOSerial::temperature (*C++ member*), 399
aMTMIOSerial::timer (*C++ member*), 399
aMTMIOSerial::uart (*C++ member*), 398
aMTMIOSerial::usb (*C++ member*), 399
aMTMIOSERIAL_5VRAIL (*C macro*), 399
aMTMIOSERIAL_ADJRAIL1 (*C macro*), 399
aMTMIOSERIAL_ADJRAIL2 (*C macro*), 399
aMTMIOSERIAL_ERROR_VBUS_OVERCURRENT (*C macro*), 401
aMTMIOSERIAL_MAX_MICROVOLTAGE (*C macro*), 399
aMTMIOSERIAL_MIN_MICROVOLTAGE (*C macro*), 399
aMTMIOSERIAL_MODULE_BASE_ADDRESS (*C macro*), 399
aMTMIOSERIAL_NUM_APPS (*C macro*), 399
aMTMIOSERIAL_NUM_DIGITALS (*C macro*), 399
aMTMIOSERIAL_NUM_I2C (*C macro*), 399
aMTMIOSERIAL_NUM_INPUT_SIGNALS (*C macro*), 400
aMTMIOSERIAL_NUM_INTERNAL_SLOTS (*C macro*), 400
aMTMIOSERIAL_NUM_OUTPUT_SIGNALS (*C macro*), 400
aMTMIOSERIAL_NUM_POINTERS (*C macro*), 399
aMTMIOSERIAL_NUM_RAILS (*C macro*), 399
aMTMIOSERIAL_NUM_RAM_SLOTS (*C macro*), 400
aMTMIOSERIAL_NUM_SERVOS (*C macro*), 400

aMTMIOSERIAL_NUM_SIGNALS (*C macro*), 400
 aMTMIOSERIAL_NUM_STORES (*C macro*), 400
 aMTMIOSERIAL_NUM_TIMERS (*C macro*), 400
 aMTMIOSERIAL_NUM_UART (*C macro*), 400
 aMTMIOSERIAL_NUM_USB (*C macro*), 400
 aMTMIOSERIAL_USB2_BOOST_ENABLED (*C macro*), 401
 aMTMIOSERIAL_USB2_DATA_ENABLED (*C macro*), 401
 aMTMIOSERIAL_USB_ERROR_FLAG (*C macro*), 401
 aMTMIOSERIAL_USB_NUM_CHANNELS (*C macro*), 400
 aMTMIOSERIAL_USB_VBUS_ENABLED (*C macro*), 401
 aMTMLoad1 (*C++ class*), 401
 aMTMLoad1::app (*C++ member*), 402
 aMTMLoad1::digital (*C++ member*), 402
 aMTMLoad1::i2c (*C++ member*), 402
 aMTMLoad1::pointer (*C++ member*), 402
 aMTMLoad1::rail (*C++ member*), 402
 aMTMLoad1::store (*C++ member*), 402
 aMTMLoad1::system (*C++ member*), 402
 aMTMLoad1::temperature (*C++ member*), 402
 aMTMLoad1::timer (*C++ member*), 402
 aMTMLoad1_MAX_CURRENT_LIMIT_MICROAMPS (*C macro*), 403
 aMTMLoad1_MAX_MICROAMPS (*C macro*), 403
 aMTMLoad1_MAX_MICROVOLTAGE (*C macro*), 403
 aMTMLoad1_MAX_MILLIOHMS (*C macro*), 403
 aMTMLoad1_MAX_MILLIWATTS (*C macro*), 403
 aMTMLoad1_MAX_POWER_LIMIT_MILLIWATTS (*C macro*), 403
 aMTMLoad1_MAX_VOLTAGE_LIMIT_MICROVOLTS (*C macro*), 403
 aMTMLoad1_MIN_CURRENT_LIMIT_MICROAMPS (*C macro*), 403
 aMTMLoad1_MIN_MICROAMPS (*C macro*), 403
 aMTMLoad1_MIN_MICROVOLTAGE (*C macro*), 403
 aMTMLoad1_MIN_MILLIOHMS (*C macro*), 403
 aMTMLoad1_MIN_MILLIWATTS (*C macro*), 403
 aMTMLoad1_MIN_POWER_LIMIT_MILLIWATTS (*C macro*), 403
 aMTMLoad1_MIN_VOLTAGE_LIMIT_MICROVOLTS (*C macro*), 403
 aMTMLoad1_MODULE_BASE_ADDRESS (*C macro*), 402
 aMTMLoad1_NUM_APPS (*C macro*), 402
 aMTMLoad1_NUM_DIGITALS (*C macro*), 402
 aMTMLoad1_NUM_I2C (*C macro*), 402
 aMTMLoad1_NUM_INTERNAL_SLOTS (*C macro*), 404
 aMTMLoad1_NUM_POINTERS (*C macro*), 402
 aMTMLoad1_NUM_RAILS (*C macro*), 402
 aMTMLoad1_NUM_RAM_SLOTS (*C macro*), 404
 aMTMLoad1_NUM_STORES (*C macro*), 404
 aMTMLoad1_NUM_TEMPERATURES (*C macro*), 404
 aMTMLoad1_NUM_TIMERS (*C macro*), 404
 aMTMLoad1_RAIL0 (*C macro*), 403
 aMTMPM1 (*C++ class*), 404
 aMTMPM1::app (*C++ member*), 404
 aMTMPM1::digital (*C++ member*), 404
 aMTMPM1::i2c (*C++ member*), 404
 aMTMPM1::pointer (*C++ member*), 404
 aMTMPM1::rail (*C++ member*), 404
 aMTMPM1::store (*C++ member*), 404
 aMTMPM1::system (*C++ member*), 405
 aMTMPM1::temperature (*C++ member*), 405
 aMTMPM1::timer (*C++ member*), 405
 aMTMPM1_MAX_CURRENT_LIMIT_MICROAMPS (*C macro*), 405
 aMTMPM1_MAX_MICROVOLTAGE (*C macro*), 405
 aMTMPM1_MIN_CURRENT_LIMIT_MICROAMPS (*C macro*), 406
 aMTMPM1_MIN_MICROVOLTAGE (*C macro*), 405
 aMTMPM1_MODULE_BASE_ADDRESS (*C macro*), 405
 aMTMPM1_NUM_APPS (*C macro*), 405
 aMTMPM1_NUM_DIGITALS (*C macro*), 405
 aMTMPM1_NUM_I2C (*C macro*), 405
 aMTMPM1_NUM_INTERNAL_SLOTS (*C macro*), 406
 aMTMPM1_NUM_POINTERS (*C macro*), 405
 aMTMPM1_NUM_RAILS (*C macro*), 405
 aMTMPM1_NUM_RAM_SLOTS (*C macro*), 406

aMTMPM1_NUM_STORES (*C macro*), 406
aMTMPM1_NUM_TEMPERATURES (*C macro*), 406
aMTMPM1_NUM_TIMERS (*C macro*), 406
aMTMPM1_RAILO (*C macro*), 405
aMTMPM1_RAIL1 (*C macro*), 405
aMTMRelay (*C++ class*), 406
aMTMRelay::app (*C++ member*), 406
aMTMRelay::digital (*C++ member*), 406
aMTMRelay::i2c (*C++ member*), 406
aMTMRelay::pointer (*C++ member*), 406
aMTMRelay::relay (*C++ member*), 406
aMTMRelay::store (*C++ member*), 407
aMTMRelay::system (*C++ member*), 407
aMTMRelay::timer (*C++ member*), 407
aMTMRELAY_MODULE_BASE_ADDRESS (*C macro*), 407
aMTMRELAY_NUM_APPS (*C macro*), 407
aMTMRELAY_NUM_DIGITALS (*C macro*), 407
aMTMRELAY_NUM_I2C (*C macro*), 407
aMTMRELAY_NUM_INTERNAL_SLOTS (*C macro*), 407
aMTMRELAY_NUM_POINTERS (*C macro*), 407
aMTMRELAY_NUM_RAM_SLOTS (*C macro*), 407
aMTMRELAY_NUM_RELAYS (*C macro*), 407
aMTMRELAY_NUM_STORES (*C macro*), 407
aMTMRELAY_NUM_TIMERS (*C macro*), 407
aMTMStemModule (*C++ class*), 409
aMTMStemModule::analog (*C++ member*), 409
aMTMStemModule::app (*C++ member*), 409
aMTMStemModule::clock (*C++ member*), 409
aMTMStemModule::digital (*C++ member*), 409
aMTMStemModule::i2c (*C++ member*), 409
aMTMStemModule::pointer (*C++ member*), 409
aMTMStemModule::servo (*C++ member*), 410
aMTMStemModule::signal (*C++ member*), 410
aMTMStemModule::store (*C++ member*), 410
aMTMStemModule::system (*C++ member*), 410
aMTMStemModule::timer (*C++ member*), 410
aMTMUSBStem (*C++ class*), 408
aMutex_Create (*C++ function*), 514
aMutex_Destroy (*C++ function*), 514
aMutex_Identifier (*C++ function*), 514
aMutex_Lock (*C++ function*), 514
aMutex_TryLock (*C++ function*), 515
aMutex_Unlock (*C++ function*), 515
aMutexRef (*C++ type*), 514
Analog (*class in brainstem.entity*), 288
analog_getBulkCaptureNumberOfSamples (*C++ function*), 643
analog_getBulkCaptureSampleRate (*C++ function*), 642
analog_getBulkCaptureState (*C++ function*), 643
analog_getConfiguration (*C++ function*), 642
analog_getEnable (*C++ function*), 640
analog_getRange (*C++ function*), 640
analog_getValue (*C++ function*), 639
analog_getVoltage (*C++ function*), 640
analog_Hz_Maximum (*C macro*), 527
analog_Hz_Minimum (*C macro*), 526
analog_initiateBulkCapture (*C++ function*), 643
analog_setBulkCaptureNumberOfSamples (*C++ function*), 643
analog_setBulkCaptureSampleRate (*C++ function*), 642
analog_setConfiguration (*C++ function*), 642
analog_setEnable (*C++ function*), 641
analog_setRange (*C++ function*), 641
analog_setValue (*C++ function*), 640
analog_setVoltage (*C++ function*), 641
analogBulkCapture (*C macro*), 527
analogBulkCaptureNumberOfSamples (*C macro*), 527
analogBulkCaptureSampleRate (*C macro*), 526
analogBulkCaptureState (*C macro*), 527
analogConfiguration (*C macro*), 526

analogConfigurationHiZ (*C macro*), 526
 analogConfigurationInput (*C macro*), 526
 analogConfigurationOutput (*C macro*), 526
 analogEnable (*C macro*), 528
 analogRange (*C macro*), 527
 analogRange_P0V064N0V064 (*C macro*), 527
 analogRange_P0V128N0V128 (*C macro*), 527
 analogRange_P0V256N0V256 (*C macro*), 527
 analogRange_P0V512N0V512 (*C macro*), 528
 analogRange_P0V64N0V64 (*C macro*), 527
 analogRange_P10V24N0V0 (*C macro*), 528
 analogRange_P10V24N10V24 (*C macro*), 528
 analogRange_P1V024N1V024 (*C macro*), 528
 analogRange_P1V28N0V0 (*C macro*), 527
 analogRange_P1V28N1V28 (*C macro*), 527
 analogRange_P2V048N0V0 (*C macro*), 528
 analogRange_P2V56N0V0 (*C macro*), 528
 analogRange_P2V56N2V56 (*C macro*), 528
 analogRange_P4V096N0V0 (*C macro*), 528
 analogRange_P5V12N0V0 (*C macro*), 528
 analogRange_P5V12N5V12 (*C macro*), 528
 analogValue (*C macro*), 526
 analogVoltage (*C macro*), 526
 aPacket (*C++ struct*), 515
 aPacket_AddByte (*C++ function*), 516
 aPacket_Create (*C++ function*), 516
 aPacket_CreateWithData (*C++ function*), 516
 aPacket_Destroy (*C++ function*), 517
 aPacket_IsComplete (*C++ function*), 516
 aPacket_Reset (*C++ function*), 516
 App (*class in brainstem.entity*), 292
 app_execute (*C++ function*), 644
 app_executeAndReturn (*C++ function*), 644
 appExecute (*C macro*), 524
 appReturn (*C macro*), 524
 aSerial_Bits (*C++ enum*), 547
 aSerial_Bits::aBITS_7 (*C++ enumerator*), 547
 aSerial_Bits::aBITS_8 (*C++ enumerator*), 547
 aSerial_Stop_bits (*C++ enum*), 547
 aSerial_Stop_bits::aSTOP_BITS_1 (*C++ enumerator*), 547
 aSerial_Stop_bits::aSTOP_BITS_2 (*C++ enumerator*), 547
 aSHOWERR (*C macro*), 500
 aSNPRINTF (*C macro*), 500
 aStream_Create (*C++ function*), 547
 aStream_CreateFileInput (*C++ function*), 547
 aStream_CreateFileOutput (*C++ function*), 548
 aStream_CreateLogStream (*C++ function*), 551
 aStream_CreateMemory (*C++ function*), 549
 aStream_CreatePipe (*C++ function*), 550
 aStream_CreateSerial (*C++ function*), 548
 aStream_CreateSocket (*C++ function*), 548
 aStream_CreateUSB (*C++ function*), 549
 aStream_Destroy (*C++ function*), 555
 aStream_Flush (*C++ function*), 555
 aStream_Read (*C++ function*), 551
 aStream_ReadCString (*C++ function*), 553
 aStream_ReadCStringRecord (*C++ function*), 554
 aStream_ReadRecord (*C++ function*), 552
 aStream_Write (*C++ function*), 552
 aStream_WriteCString (*C++ function*), 553
 aStream_WriteCStringRecord (*C++ function*), 554
 aStream_WriteRecord (*C++ function*), 553
 aStreamBuffer_Create (*C++ function*), 550
 aStreamBuffer_Flush (*C++ function*), 551
 aStreamBuffer_Get (*C++ function*), 550
 aStreamDeleteProc (*C++ type*), 545
 aStreamGetProc (*C++ type*), 545
 aStreamPutProc (*C++ type*), 545

aStreamRef (C++ type), 544
aStreamWriteProc (C++ type), 546
aStringCatSafe (C macro), 500
aStringCopySafe (C macro), 500
aSystemBootSlotNone (C macro), 519
aTime_GetMSTicks (C++ function), 555
aTime_MSSleep (C++ function), 555
aUEI_RetrieveInt (C++ function), 557
aUEI_RetrieveShort (C++ function), 557
aUEI_StoreInt (C++ function), 557
aUEI_StoreShort (C++ function), 557
aUSB_UPSTREAM_CONFIG_AUTO (C macro), 400
aUSB_UPSTREAM_CONFIG_EDGE (C macro), 400
aUSB_UPSTREAM_CONFIG_ONBOARD (C macro), 400
aUSB_UPSTREAM_EDGE (C macro), 401
aUSB_UPSTREAM_ONBOARD (C macro), 400
aUSBCSwitch (C++ class), 388
aUSBCSwitch::app (C++ member), 390
aUSBCSwitch::daughtercard_type (C++ enum), 390
aUSBCSwitch::daughtercard_type::NO_DAUGHTERCARD (C++ enumerator), 390
aUSBCSwitch::daughtercard_type::PASSIVE_DAUGHTERCARD (C++ enumerator), 390
aUSBCSwitch::daughtercard_type::REDRIVER_DAUGHTERCARD (C++ enumerator), 390
aUSBCSwitch::daughtercard_type::UNKNOWN_DAUGHTERCARD (C++ enumerator), 390
aUSBCSwitch::equalizer (C++ member), 391
aUSBCSwitch::EQUALIZER_2P0_RECEIVER_CONFIGS (C++ enum), 389
aUSBCSwitch::EQUALIZER_2P0_RECEIVER_CONFIGS::LEVEL_1_2P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_2P0_RECEIVER_CONFIGS::LEVEL_2_2P0 (C++ enumerator), 390
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS (C++ enum), 389
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_0mV (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_40mV (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_60mV (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_2P0_TRANSMITTER_CONFIGS::TRANSMITTER_2P0_80mV (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS (C++ enum), 388
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_10_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_11_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_12_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_13_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_14_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_15_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_16_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_1_3P0 (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_2_3P0 (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_3_3P0 (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_4_3P0 (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_5_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_6_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_7_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_8_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_RECEIVER_CONFIGS::LEVEL_9_3P0 (C++ enumerator), 389
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS (C++ enum), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_0db_1100mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_0db_1300mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_0db_900mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_1db_1100mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_0db_COM_1db_900mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_1db_COM_0db_1100mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_1db_COM_0db_900mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_1db_COM_1db_900mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_3P0_TRANSMITTER_CONFIGS::MUX_2db_COM_2db_1100mV (C++ enumerator), 388
aUSBCSwitch::EQUALIZER_CHANNELS (C++ enum), 390
aUSBCSwitch::EQUALIZER_CHANNELS::BOTH (C++ enumerator), 390
aUSBCSwitch::EQUALIZER_CHANNELS::COMMON (C++ enumerator), 390
aUSBCSwitch::EQUALIZER_CHANNELS::MUX (C++ enumerator), 390
aUSBCSwitch::mux (C++ member), 390
aUSBCSwitch::pointer (C++ member), 390
aUSBCSwitch::store (C++ member), 390
aUSBCSwitch::system (C++ member), 390
aUSBCSwitch::timer (C++ member), 390

aUSBCSwitch::usb (*C++ member*), 391
 aUSBCSWITCH_MODULE (*C macro*), 391
 aUSBCSWITCH_NUM_APPS (*C macro*), 391
 aUSBCSWITCH_NUM_EQ (*C macro*), 391
 aUSBCSWITCH_NUM_INTERNAL_SLOTS (*C macro*), 391
 aUSBCSWITCH_NUM_MUX (*C macro*), 391
 aUSBCSWITCH_NUM_MUX_CHANNELS (*C macro*), 391
 aUSBCSWITCH_NUM_POINTERS (*C macro*), 391
 aUSBCSWITCH_NUM_RAM_SLOTS (*C macro*), 391
 aUSBCSWITCH_NUM_STORES (*C macro*), 391
 aUSBCSWITCH_NUM_TIMERS (*C macro*), 391
 aUSBCSWITCH_NUM_USB (*C macro*), 391
 aUSBHub2x4 (*C++ class*), 385
 aUSBHub2x4::app (*C++ member*), 386
 aUSBHub2x4::pointer (*C++ member*), 386
 aUSBHub2x4::store (*C++ member*), 386
 aUSBHub2x4::system (*C++ member*), 386
 aUSBHub2x4::temperature (*C++ member*), 386
 aUSBHub2x4::timer (*C++ member*), 386
 aUSBHub2x4::usb (*C++ member*), 386
 aUSBHUB2X4_CONSTANT_CURRENT (*C macro*), 387
 aUSBHUB2X4_DEVICE_ATTACHED (*C macro*), 387
 aUSBHub2X4_ERROR_DISCHARGE (*C macro*), 387
 aUSBHUB2X4_ERROR_OVER_TEMPERATURE (*C macro*), 387
 aUSBHUB2X4_ERROR_VBUS_OVERCURRENT (*C macro*), 387
 aUSBHUB2X4_MODULE (*C macro*), 386
 aUSBHUB2X4_NUM_APPS (*C macro*), 386
 aUSBHUB2X4_NUM_INTERNAL_SLOTS (*C macro*), 386
 aUSBHUB2X4_NUM_POINTERS (*C macro*), 386
 aUSBHUB2X4_NUM_RAM_SLOTS (*C macro*), 386
 aUSBHUB2X4_NUM_STORES (*C macro*), 386
 aUSBHUB2X4_NUM_TIMERS (*C macro*), 386
 aUSBHUB2X4_NUM_USB (*C macro*), 386
 aUSBHUB2x4_NUM_USB_PORTS (*C macro*), 387
 aUSBHUB2X4_USB2_BOOST_ENABLED (*C macro*), 387
 aUSBHUB2X4_USB2_DATA_ENABLED (*C macro*), 387
 aUSBHUB2X4_USB_ERROR_FLAG (*C macro*), 387
 aUSBHUB2X4_USB_VBUS_ENABLED (*C macro*), 387
 aUSBHub3c (*C++ class*), 380
 aUSBHub3c::app (*C++ member*), 381
 aUSBHub3c::hub (*C++ member*), 381
 aUSBHub3c::HubClass (*C++ class*), 381
 aUSBHub3c::i2c (*C++ member*), 381
 aUSBHub3c::pd (*C++ member*), 381
 aUSBHub3c::pointer (*C++ member*), 381
 aUSBHub3c::PORT_ID (*C++ enum*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_0 (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_1 (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_2 (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_3 (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_4 (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_5 (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_CONTROL (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID::kPORT_ID_POWER_C (*C++ enumerator*), 380
 aUSBHub3c::PORT_ID_t (*C++ type*), 380
 aUSBHub3c::rail (*C++ member*), 381
 aUSBHub3c::store (*C++ member*), 381
 aUSBHub3c::system (*C++ member*), 381
 aUSBHub3c::temperature (*C++ member*), 381
 aUSBHub3c::timer (*C++ member*), 381
 aUSBHub3c::uart (*C++ member*), 381
 aUSBHub3c::usb (*C++ member*), 381
 aUSBHUB3C_MODULE (*C macro*), 382
 aUSBHUB3C_NUM_APPS (*C macro*), 382
 aUSBHUB3C_NUM_I2C (*C macro*), 383
 aUSBHUB3C_NUM_INTERNAL_SLOTS (*C macro*), 382
 aUSBHUB3C_NUM_PD_PORTS (*C macro*), 382
 aUSBHUB3C_NUM_PD_RULES_PER_PORT (*C macro*), 382

aUSBHUB3C_NUM_POINTERS (*C macro*), 382
aUSBHUB3C_NUM_RAILS (*C macro*), 383
aUSBHUB3C_NUM_RAM_SLOTS (*C macro*), 382
aUSBHUB3C_NUM_STORES (*C macro*), 382
aUSBHUB3C_NUM_TEMPERATURES (*C macro*), 382
aUSBHUB3C_NUM_TIMERS (*C macro*), 382
aUSBHUB3C_NUM_UART (*C macro*), 383
aUSBHUB3C_NUM_USB (*C macro*), 382
aUSBHUB3C_NUM_USB_PORTS (*C macro*), 382
aUSBHUB3C_STORE_EEPROM_INDEX (*C macro*), 382
aUSBHUB3C_STORE_INTERNAL_INDEX (*C macro*), 382
aUSBHUB3C_STORE_RAM_INDEX (*C macro*), 382
aUSBHub3p (*C++ class*), 383
aUSBHub3p::app (*C++ member*), 383
aUSBHub3p::pointer (*C++ member*), 383
aUSBHub3p::store (*C++ member*), 383
aUSBHub3p::system (*C++ member*), 383
aUSBHub3p::temperature (*C++ member*), 383
aUSBHub3p::timer (*C++ member*), 383
aUSBHub3p::usb (*C++ member*), 383
aUSBHUB3P_DEVICE_ATTACHED (*C macro*), 385
aUSBHUB3P_ERROR_DISCHARGE_ERR (*C macro*), 385
aUSBHUB3P_ERROR_HUB_POWER (*C macro*), 385
aUSBHUB3P_ERROR_OVER_TEMPERATURE (*C macro*), 385
aUSBHUB3P_ERROR_SHORT_CIRCUIT (*C macro*), 385
aUSBHUB3P_ERROR_VBUS_BACKDRIVE (*C macro*), 385
aUSBHUB3P_ERROR_VBUS_OVERCURRENT (*C macro*), 385
aUSBHUB3P_MODULE (*C macro*), 384
aUSBHUB3P_NUM_APPS (*C macro*), 384
aUSBHUB3P_NUM_INTERNAL_SLOTS (*C macro*), 384
aUSBHUB3P_NUM_POINTERS (*C macro*), 384
aUSBHUB3P_NUM_RAM_SLOTS (*C macro*), 384
aUSBHUB3P_NUM_STORES (*C macro*), 384
aUSBHUB3P_NUM_TIMERS (*C macro*), 384
aUSBHUB3P_NUM_USB (*C macro*), 384
aUSBHUB3P_NUM_USB_PORTS (*C macro*), 384
aUSBHUB3P_USB2_BOOST_ENABLED (*C macro*), 385
aUSBHUB3P_USB2_DATA_ENABLED (*C macro*), 384
aUSBHUB3P_USB3_DATA_ENABLED (*C macro*), 384
aUSBHUB3P_USB_ERROR_FLAG (*C macro*), 385
aUSBHUB3P_USB_SPEED_USB2 (*C macro*), 384
aUSBHUB3P_USB_SPEED_USB3 (*C macro*), 384
aUSBHUB3P_USB_VBUS_ENABLED (*C macro*), 384
aVALIDPACKET (*C++ function*), 515
aVersion_DestroyFeatureList (*C++ function*), 560
aVersion_GetFeatureList (*C++ function*), 560
aVersion_GetMajor (*C++ function*), 559
aVersion_GetMinor (*C++ function*), 559
aVersion_GetPatch (*C++ function*), 559
aVersion_GetString (*C++ function*), 559
aVersion_IsAtLeast (*C++ function*), 559
aVERSION_MAJOR (*C macro*), 558
aVERSION_MINOR (*C macro*), 558
aVersion_ParseMajor (*C++ function*), 558
aVersion_ParseMinor (*C++ function*), 558
aVersion_ParsePatch (*C++ function*), 559
aVersion_ParseString (*C++ function*), 559
aVERSION_PATCH (*C macro*), 558

B

baudrate (*brainstem.link.Spec attribute*), 310
bAutoNetworking (*brainstem.module.Module property*), 314
bContinueSearch (*C++ type*), 502
bitSlotError (*C macro*), 523
brainstem.defs
 module, 295
brainstem.discover

module, 298
 brainstem.entity
 module, 307
 brainstem.link
 module, 309
 brainstem.module
 module, 311
 brainstem.result
 module, 349
 brainstem.version
 module, 375
 bulkCaptureError (*C macro*), 527
 bulkCaptureFinished (*C macro*), 527
 bulkCaptureIdle (*C macro*), 527
 bulkCapturePending (*C macro*), 527

C

call_UEI() (*brainstem.module.Entity method*), 311
 capacityBuild (*C macro*), 535
 capacityClassQuantity (*C macro*), 535
 capacityEntityGroup (*C macro*), 535
 capacitySubClassQuantity (*C macro*), 535
 capacitySubClassSize (*C macro*), 535
 capacityUEI (*C macro*), 535
 check_UEIBytes() (*brainstem.module.Entity method*), 311
 clearFaults() (*brainstem.entity.Rail method*), 341
 clearPortErrorStatus() (*brainstem.entity.USB method*), 365
 Clock (*class in brainstem.entity*), 293
 clock_getDay (*C++ function*), 646
 clock_getHour (*C++ function*), 646
 clock_getMinute (*C++ function*), 647
 clock_getMonth (*C++ function*), 645
 clock_getSecond (*C++ function*), 647
 clock_getYear (*C++ function*), 645
 clock_setDay (*C++ function*), 646
 clock_setHour (*C++ function*), 646
 clock_setMinute (*C++ function*), 647
 clock_setMonth (*C++ function*), 645
 clock_setSecond (*C++ function*), 647
 clock_setYear (*C++ function*), 645
 clockDay (*C macro*), 538
 clockHour (*C macro*), 538
 clockMinute (*C macro*), 538
 clockMonth (*C macro*), 538
 clockSecond (*C macro*), 538
 clockYear (*C macro*), 538
 cmdANALOG (*C macro*), 526
 cmdAPP (*C macro*), 523
 cmdCAPACITY (*C macro*), 535
 cmdCLOCK (*C macro*), 537
 cmdDEBUG (*C macro*), 526
 cmdDIGITAL (*C macro*), 529
 cmdLAST (*C macro*), 544
 cmdMUX (*C macro*), 524
 cmdPOINTER (*C macro*), 525
 cmdRAIL (*C macro*), 530
 cmdSLOT (*C macro*), 522
 cmdSTORE (*C macro*), 536
 cmdSYSTEM (*C macro*), 519
 cmdTEMPERATURE (*C macro*), 534
 cmdTIMER (*C macro*), 537
 cmdUPGRADE (*C macro*), 544
 cmdUSB (*C macro*), 538
 command (*brainstem.module.Entity property*), 311
 connect() (*brainstem.module.Module method*), 315
 connect() (*brainstem.stem.EtherStem method*), 285
 connect() (*brainstem.stem.MTMDAQ1 method*), 284

`connect()` (*brainstem.stem.MTMDAQ2 method*), 276
`connect()` (*brainstem.stem.MTMEtherStem method*), 277
`connect()` (*brainstem.stem.MTMIOSerial method*), 278
`connect()` (*brainstem.stem.MTMLOAD1 method*), 279
`connect()` (*brainstem.stem.MTMPM1 method*), 280
`connect()` (*brainstem.stem.MTMRelay method*), 281
`connect()` (*brainstem.stem.MTMUSBStem method*), 282
`connect()` (*brainstem.stem.USBCSwitch method*), 274
`connect()` (*brainstem.stem.USBHub2x4 method*), 272
`connect()` (*brainstem.stem.USBHub3p method*), 271
`connect()` (*brainstem.stem.USBStem method*), 286
`connectFromSpec()` (*brainstem.module.Module method*), 315
`connectThroughLinkModule()` (*brainstem.module.Module method*), 315

D

`dataType` (C++ enum), 556
`dataType::aUEI_BYTE` (C++ enumerator), 556
`dataType::aUEI_BYTES` (C++ enumerator), 556
`dataType::aUEI_INT` (C++ enumerator), 556
`dataType::aUEI_SHORT` (C++ enumerator), 556
`dataType::aUEI_VOID` (C++ enumerator), 556
`DefaultOperationalRailMode_Value` (C macro), 531
`DefaultPointerMode` (C macro), 525
`DefaultTimerMode` (C macro), 537
`DeviceNode` (C++ struct), 560
`DeviceNode` (class in *brainstem.discover*), 298
`DeviceNode::hubPort` (C++ member), 561
`DeviceNode::hubSerialNumber` (C++ member), 561
`DeviceNode::idProduct` (C++ member), 561
`DeviceNode::idVendor` (C++ member), 561
`DeviceNode::manufacturer` (C++ member), 561
`DeviceNode::productName` (C++ member), 561
`DeviceNode::serialNumber` (C++ member), 561
`DeviceNode::speed` (C++ member), 561
`Digital` (class in *brainstem.entity*), 296
`digital_getConfiguration` (C++ function), 648
`digital_getState` (C++ function), 649
`digital_getStateAll` (C++ function), 649
`digital_setConfiguration` (C++ function), 648
`digital_setState` (C++ function), 648
`digital_setStateAll` (C++ function), 649
`digitalConfiguration` (C macro), 529
`digitalConfigurationHiZ` (C macro), 529
`digitalConfigurationInput` (C macro), 529
`digitalConfigurationInputNoPull` (C macro), 529
`digitalConfigurationInputPullDown` (C macro), 529
`digitalConfigurationInputPullUp` (C macro), 529
`digitalConfigurationOutput` (C macro), 529
`digitalConfigurationRCServoInput` (C macro), 529
`digitalConfigurationRCServoOutput` (C macro), 529
`digitalConfigurationSignalCounterInput` (C macro), 529
`digitalConfigurationSignalInput` (C macro), 529
`digitalConfigurationSignalOutput` (C macro), 529
`digitalState` (C macro), 529
`digitalStateAll` (C macro), 530
`disconnect()` (*brainstem.module.Module method*), 315
`discoverAndConnect()` (*brainstem.module.Module method*), 315
`drain_UEI()` (*brainstem.module.Entity method*), 311

E

`Entity` (class in *brainstem.module*), 311
`Equalizer` (class in *brainstem.entity*), 307
`equalizer_getReceiverConfig` (C++ function), 650
`equalizer_getTransmitterConfig` (C++ function), 650
`equalizer_setReceiverConfig` (C++ function), 650
`equalizer_setTransmitterConfig` (C++ function), 650
`error` (*brainstem.result.Result property*), 349

EtherStem (*class in brainstem.stem*), 284
 execute() (*brainstem.entity.App method*), 292
 executeAndWaitForReturn() (*brainstem.entity.App method*), 292

F

findAllModules() (*in module brainstem.discover*), 299
 findFirstModule() (*in module brainstem.discover*), 299
 findModule() (*in module brainstem.discover*), 299

G

get_UEI() (*brainstem.module.Entity method*), 312
 get_UEI_with_param() (*brainstem.module.Entity method*), 312
 get_UEIBytes() (*brainstem.module.Entity method*), 312
 get_usbPortStateCOM_ORIENT_STATUS (*C macro*), 392
 get_usbPortStateDaughterCard (*C macro*), 393
 get_usbPortStateMUX_ORIENT_STATUS (*C macro*), 392
 get_usbPortStateSPEED_STATUS (*C macro*), 392
 get_version_string() (*in module brainstem.version*), 375
 getAllocatedPower() (*brainstem.entity.Port method*), 322
 getAltModeConfig() (*brainstem.entity.USB method*), 365
 getAvailablePower() (*brainstem.entity.Port method*), 322
 getBaudRate() (*brainstem.entity.UART method*), 363
 getBootSlot() (*brainstem.entity.System method*), 351
 getBulkCaptureNumberOfSamples() (*brainstem.entity.Analog method*), 288
 getBulkCaptureSampleRate() (*brainstem.entity.Analog method*), 288
 getBulkCaptureState() (*brainstem.entity.Analog method*), 289
 getCableCurrentMax() (*brainstem.entity.PowerDelivery method*), 333
 getCableFlip() (*brainstem.entity.USB method*), 366
 getCableOrientation() (*brainstem.entity.PowerDelivery method*), 333
 getCableSpeedMax() (*brainstem.entity.PowerDelivery method*), 333
 getCableType() (*brainstem.entity.PowerDelivery method*), 333
 getCableVoltageMax() (*brainstem.entity.PowerDelivery method*), 334
 getCC1Current() (*brainstem.entity.USB method*), 365
 getCC1Enabled() (*brainstem.entity.Port method*), 322
 getCC1Voltage() (*brainstem.entity.USB method*), 365
 getCC2Current() (*brainstem.entity.USB method*), 365
 getCC2Enabled() (*brainstem.entity.Port method*), 322
 getCC2Voltage() (*brainstem.entity.USB method*), 366
 getCCEnabled() (*brainstem.entity.Port method*), 322
 getChannel() (*brainstem.entity.Mux method*), 317
 getChar() (*brainstem.entity.Pointer method*), 319
 getConfiguration() (*brainstem.entity.Analog method*), 289
 getConfiguration() (*brainstem.entity.Digital method*), 296
 getConfiguration() (*brainstem.entity.Mux method*), 317
 getConnectionState() (*brainstem.entity.PowerDelivery method*), 334
 getConnectMode() (*brainstem.entity.USB method*), 366
 getCurrent() (*brainstem.entity.Rail method*), 341
 getCurrentLimit() (*brainstem.entity.Port method*), 322
 getCurrentLimit() (*brainstem.entity.Rail method*), 341
 getCurrentLimitMode() (*brainstem.entity.Port method*), 323
 getCurrentSetpoint() (*brainstem.entity.Rail method*), 341
 getDataEnabled() (*brainstem.entity.Port method*), 323
 getDataHS1Enabled() (*brainstem.entity.Port method*), 323
 getDataHS2Enabled() (*brainstem.entity.Port method*), 323
 getDataHSEnabled() (*brainstem.entity.Port method*), 323
 getDataHSRoutingBehavior() (*brainstem.entity.Port method*), 324
 getDataRole() (*brainstem.entity.Port method*), 324
 getDataRoleBehavior() (*brainstem.entity.USBSystem method*), 372
 getDataRoleBehaviorConfig() (*brainstem.entity.USBSystem method*), 372
 getDataRoleList() (*brainstem.entity.USBSystem method*), 372
 getDataSpeed() (*brainstem.entity.Port method*), 325
 getDataSS1Enabled() (*brainstem.entity.Port method*), 324
 getDataSS2Enabled() (*brainstem.entity.Port method*), 324
 getDataSSEnabled() (*brainstem.entity.Port method*), 324
 getDataSSRoutingBehavior() (*brainstem.entity.Port method*), 324
 getDay() (*brainstem.entity.Clock method*), 293
 getDownstreamBoostMode() (*brainstem.entity.USB method*), 366

`getDownstreamDataSpeed()` (*brainstem.entity.USB method*), 366
`getDownstreamDevices` (C++ function), 561
`getDownstreamDevices()` (in module *brainstem.discover*), 299
`getEnable()` (*brainstem.entity.Analog method*), 289
`getEnable()` (*brainstem.entity.Mux method*), 317
`getEnable()` (*brainstem.entity.Rail method*), 341
`getEnable()` (*brainstem.entity.RCServo method*), 347
`getEnable()` (*brainstem.entity.Relay method*), 348
`getEnable()` (*brainstem.entity.Signal method*), 349
`getEnable()` (*brainstem.entity.UART method*), 363
`getEnabled()` (*brainstem.entity.Port method*), 325
`getEnabledList()` (*brainstem.entity.USBSystem method*), 372
`getEnumerationDelay()` (*brainstem.entity.USB method*), 366
`getEnumerationDelay()` (*brainstem.entity.USBSystem method*), 372
`getErrors()` (*brainstem.entity.Port method*), 325
`getErrors()` (*brainstem.entity.System method*), 351
`getExpiration()` (*brainstem.entity.Timer method*), 362
`getFastRoleSwapCurrent()` (*brainstem.entity.PowerDelivery method*), 334
`getFlagMode()` (*brainstem.entity.PowerDelivery method*), 334
`getHardwareVersion()` (*brainstem.entity.System method*), 351
`getHBInterval()` (*brainstem.entity.System method*), 351
`getHour()` (*brainstem.entity.Clock method*), 293
`getHSBoost()` (*brainstem.entity.Port method*), 325
`getHubMode()` (*brainstem.entity.USB method*), 366
`getInputCurrent()` (*brainstem.entity.System method*), 351
`getInputPowerBehavior()` (*brainstem.entity.System method*), 352
`getInputPowerBehaviorConfig()` (*brainstem.entity.System method*), 352
`getInputPowerSource()` (*brainstem.entity.System method*), 352
`getInputVoltage()` (*brainstem.entity.System method*), 352
`getInt()` (*brainstem.entity.Pointer method*), 319
`getInvert()` (*brainstem.entity.Signal method*), 349
`getKelvinSensingEnable()` (*brainstem.entity.Rail method*), 341
`getKelvinSensingState()` (*brainstem.entity.Rail method*), 342
`getLED()` (*brainstem.entity.System method*), 352
`getLinkInterface()` (*brainstem.entity.System method*), 352
`getMaximumTemperature()` (*brainstem.entity.System method*), 353
`getMinimumTemperature()` (*brainstem.entity.System method*), 353
`getMinute()` (*brainstem.entity.Clock method*), 293
`getMode()` (*brainstem.entity.Pointer method*), 319
`getMode()` (*brainstem.entity.Port method*), 325
`getMode()` (*brainstem.entity.Timer method*), 362
`getModel()` (*brainstem.entity.System method*), 353
`getModelList()` (*brainstem.entity.USBSystem method*), 372
`getModule()` (*brainstem.entity.System method*), 353
`getModuleBaseAddress()` (*brainstem.entity.System method*), 353
`getModuleHardwareOffset()` (*brainstem.entity.System method*), 353
`getModuleSoftwareOffset()` (*brainstem.entity.System method*), 354
`getMonth()` (*brainstem.entity.Clock method*), 293
`getName()` (*brainstem.entity.Port method*), 325
`getName()` (*brainstem.entity.System method*), 354
`getNumberOfPowerDataObjects()` (*brainstem.entity.PowerDelivery method*), 334
`getOffset()` (*brainstem.entity.Pointer method*), 319
`getOperationalMode()` (*brainstem.entity.Rail method*), 342
`getOperationalState()` (*brainstem.entity.Rail method*), 342
`getOverride()` (*brainstem.entity.PowerDelivery method*), 335
`getPeakCurrentConfiguration()` (*brainstem.entity.PowerDelivery method*), 335
`getPortCurrent()` (*brainstem.entity.USB method*), 366
`getPortCurrentLimit()` (*brainstem.entity.USB method*), 366
`getPortError()` (*brainstem.entity.USB method*), 367
`getPortMode()` (*brainstem.entity.USB method*), 367
`getPortState()` (*brainstem.entity.USB method*), 367
`getPortVoltage()` (*brainstem.entity.USB method*), 367
`getPosition()` (*brainstem.entity.RCServo method*), 347
`getPower()` (*brainstem.entity.Rail method*), 342
`getPowerBehavior()` (*brainstem.entity.USBSystem method*), 373
`getPowerBehaviorConfig()` (*brainstem.entity.USBSystem method*), 373
`getPowerDataObject()` (*brainstem.entity.PowerDelivery method*), 335
`getPowerDataObjectEnabled()` (*brainstem.entity.PowerDelivery method*), 335

getPowerDataObjectEnabledList () (*brainstem.entity.PowerDelivery method*), 336
 getPowerDataObjectList () (*brainstem.entity.PowerDelivery method*), 336
 getPowerEnabled () (*brainstem.entity.Port method*), 326
 getPowerLimit () (*brainstem.entity.Port method*), 326
 getPowerLimit () (*brainstem.entity.Rail method*), 342
 getPowerLimit () (*brainstem.entity.System method*), 354
 getPowerLimitMax () (*brainstem.entity.System method*), 354
 getPowerLimitMode () (*brainstem.entity.Port method*), 326
 getPowerLimitState () (*brainstem.entity.System method*), 354
 getPowerMode () (*brainstem.entity.Port method*), 326
 getPowerRole () (*brainstem.entity.PowerDelivery method*), 336
 getPowerRolePreferred () (*brainstem.entity.PowerDelivery method*), 336
 getPowerSetpoint () (*brainstem.entity.Rail method*), 342
 getProtocol () (*brainstem.entity.UART method*), 363
 getRange () (*brainstem.entity.Analog method*), 289
 getReceiverConfig () (*brainstem.entity.Equalizer method*), 307
 getRequestDataObject () (*brainstem.entity.PowerDelivery method*), 337
 getResistance () (*brainstem.entity.Rail method*), 343
 getResistanceSetpoint () (*brainstem.entity.Rail method*), 343
 getReverse () (*brainstem.entity.RCServo method*), 347
 getRouter () (*brainstem.entity.System method*), 355
 getRouterAddressSetting () (*brainstem.entity.System method*), 355
 getSBU1Voltage () (*brainstem.entity.USB method*), 367
 getSBU2Voltage () (*brainstem.entity.USB method*), 367
 getSecond () (*brainstem.entity.Clock method*), 293
 getSelectorMode () (*brainstem.entity.USBSystem method*), 373
 getSerialNumber () (*brainstem.entity.System method*), 355
 getShort () (*brainstem.entity.Pointer method*), 319
 getSlotCapacity () (*brainstem.entity.Store method*), 359
 getSlotLocked () (*brainstem.entity.Store method*), 360
 getSlotSize () (*brainstem.entity.Store method*), 360
 getSlotState () (*brainstem.entity.Store method*), 360
 getSpeed () (*brainstem.entity.I2C method*), 308
 getSplitMode () (*brainstem.entity.Mux method*), 317
 getState () (*brainstem.entity.Digital method*), 296
 getState () (*brainstem.entity.Port method*), 326
 getStateAll () (*brainstem.entity.Digital method*), 297
 getStateList () (*brainstem.entity.USBSystem method*), 373
 getStatus () (*brainstem.module.Module method*), 316
 getT2Time () (*brainstem.entity.Signal method*), 350
 getT3Time () (*brainstem.entity.Signal method*), 350
 getTemperature () (*brainstem.entity.Rail method*), 343
 getTemperature () (*brainstem.entity.System method*), 355
 getTransferStore () (*brainstem.entity.Pointer method*), 320
 getTransmitterConfig () (*brainstem.entity.Equalizer method*), 307
 getUnregulatedCurrent () (*brainstem.entity.System method*), 355
 getUnregulatedVoltage () (*brainstem.entity.System method*), 355
 getUpstream () (*brainstem.entity.USBSystem method*), 373
 getUpstreamBoostMode () (*brainstem.entity.USB method*), 367
 getUpstreamMode () (*brainstem.entity.USB method*), 367
 getUpstreamState () (*brainstem.entity.USB method*), 367
 getUptime () (*brainstem.entity.System method*), 355
 getValue () (*brainstem.entity.Analog method*), 289
 getValue () (*brainstem.entity.Temperature method*), 361
 getValueMax () (*brainstem.entity.Temperature method*), 361
 getValueMin () (*brainstem.entity.Temperature method*), 361
 getVbusAccumulatedPower () (*brainstem.entity.Port method*), 326
 getVbusCurrent () (*brainstem.entity.Port method*), 327
 getVbusVoltage () (*brainstem.entity.Port method*), 327
 getVconn1Enabled () (*brainstem.entity.Port method*), 327
 getVconn2Enabled () (*brainstem.entity.Port method*), 327
 getVconnAccumulatedPower () (*brainstem.entity.Port method*), 327
 getVconnCurrent () (*brainstem.entity.Port method*), 327
 getVconnEnabled () (*brainstem.entity.Port method*), 327
 getVconnVoltage () (*brainstem.entity.Port method*), 328
 getVersion () (*brainstem.entity.System method*), 356
 getVoltage () (*brainstem.entity.Analog method*), 290
 getVoltage () (*brainstem.entity.Mux method*), 318

`getVoltage()` (*brainstem.entity.Rail method*), 343
`getVoltage()` (*brainstem.entity.Relay method*), 348
`getVoltageMaxLimit()` (*brainstem.entity.Rail method*), 343
`getVoltageMinLimit()` (*brainstem.entity.Rail method*), 344
`getVoltageSetpoint()` (*brainstem.entity.Port method*), 328
`getVoltageSetpoint()` (*brainstem.entity.Rail method*), 344
`getYear()` (*brainstem.entity.Clock method*), 293

I

`I2C` (*class in brainstem.entity*), 308
`i2c_getSpeed` (*C++ function*), 652
`i2c_read` (*C++ function*), 651
`i2c_setPullup` (*C++ function*), 652
`i2c_setSpeed` (*C++ function*), 652
`i2c_write` (*C++ function*), 651
`index` (*brainstem.module.Entity property*), 312
`initiateBulkCapture()` (*brainstem.entity.Analog method*), 290
`ip_address` (*brainstem.link.Spec attribute*), 310
`isConnected()` (*brainstem.module.Module method*), 316

K

`kelvinSensingOff_Value` (*C macro*), 530
`kelvinSensingOn_Value` (*C macro*), 530

L

`link` (*brainstem.module.Module property*), 316
`linkSpec` (*C++ struct*), 501
`linkSpec::model` (*C++ member*), 502
`linkSpec::module` (*C++ member*), 502
`linkSpec::router` (*C++ member*), 502
`linkSpec::router_serial_num` (*C++ member*), 502
`linkSpec::serial_num` (*C++ member*), 502
`linkSpec::t` (*C++ member*), 502
`linkSpec::type` (*C++ member*), 502
`linkStatus` (*C++ enum*), 510
`linkStatus::INITIALIZING` (*C++ enumerator*), 510
`linkStatus::INVALID_LINK_STREAM` (*C++ enumerator*), 510
`linkStatus::IO_ERROR` (*C++ enumerator*), 511
`linkStatus::RESETTING` (*C++ enumerator*), 511
`linkStatus::RUNNING` (*C++ enumerator*), 510
`linkStatus::STOPPED` (*C++ enumerator*), 510
`linkStatus::STOPPING` (*C++ enumerator*), 510
`linkStatus::SYNCING` (*C++ enumerator*), 510
`linkStatus::UNKNOWN_ERROR` (*C++ enumerator*), 511
`linkType` (*C++ enum*), 501
`linkType::INVALID` (*C++ enumerator*), 501
`linkType::SERIAL` (*C++ enumerator*), 501
`linkType::TCPIP` (*C++ enumerator*), 501
`linkType::USB` (*C++ enumerator*), 501
`loadSlot()` (*brainstem.entity.Store method*), 360
`logEvents()` (*brainstem.entity.System method*), 356

M

`model` (*brainstem.link.Spec attribute*), 310
`model` (*brainstem.module.Module property*), 316
`MODEL_ETHERSTEM` (*in module brainstem.defs*), 295
`model_info()` (*in module brainstem.defs*), 295
`MODEL_MTM_DAQ_1` (*in module brainstem.defs*), 295
`MODEL_MTM_DAQ_2` (*in module brainstem.defs*), 295
`MODEL_MTM_ETHERSTEM` (*in module brainstem.defs*), 295
`MODEL_MTM_IOSERIAL` (*in module brainstem.defs*), 295
`MODEL_MTM_PM_1` (*in module brainstem.defs*), 295
`MODEL_MTM_RELAY` (*in module brainstem.defs*), 295
`MODEL_MTM_USBSTEM` (*in module brainstem.defs*), 295
`model_name()` (*in module brainstem.defs*), 296

MODEL_USB_C_SWITCH (*in module brainstem.defs*), 295
 MODEL_USBHUB_2X4 (*in module brainstem.defs*), 295
 MODEL_USBHUB_3C (*in module brainstem.defs*), 295
 MODEL_USBHUB_3P (*in module brainstem.defs*), 295
 MODEL_USBSTEM (*in module brainstem.defs*), 295
 module
 brainstem.defs, 295
 brainstem.discover, 298
 brainstem.entity, 307
 brainstem.link, 309
 brainstem.module, 311
 brainstem.result, 349
 brainstem.version, 375
 module (*brainstem.link.Spec attribute*), 309
 module (*brainstem.module.Entity property*), 312
 Module (*class in brainstem.module*), 314
 module_clearAllStems (*C++ function*), 655
 module_connectThroughLinkModule (*C++ function*), 654
 module_createStem (*C++ function*), 653
 module_disconnect (*C++ function*), 653
 module_disconnectAndDestoryStem (*C++ function*), 653
 module_discoverAndConnect (*C++ function*), 653
 module_getModuleAddress (*C++ function*), 654
 module_isConnected (*C++ function*), 654
 module_reconnect (*C++ function*), 654
 module_sDiscover (*C++ function*), 653
 module_setModuleAddress (*C++ function*), 654
 module_setNetworkingMode (*C++ function*), 655
 MTMDAQ1 (*class in brainstem.stem*), 283
 MTMDAQ2 (*class in brainstem.stem*), 275
 MTMEtherStem (*class in brainstem.stem*), 276
 MTMIOSerial (*class in brainstem.stem*), 277
 MTMLOAD1 (*class in brainstem.stem*), 279
 MTMPM1 (*class in brainstem.stem*), 280
 MTMRelay (*class in brainstem.stem*), 281
 MTMUSBStem (*class in brainstem.stem*), 282
 Mux (*class in brainstem.entity*), 317
 mux_getChannel (*C++ function*), 656
 mux_getChannelVoltage (*C++ function*), 656
 mux_getConfiguration (*C++ function*), 656
 mux_getEnable (*C++ function*), 655
 mux_getSplitMode (*C++ function*), 657
 mux_setChannel (*C++ function*), 656
 mux_setConfiguration (*C++ function*), 657
 mux_setEnable (*C++ function*), 655
 mux_setSplitMode (*C++ function*), 657
 muxChannel (*C macro*), 524
 muxConfig (*C macro*), 524
 muxConfig_channelpriority (*C macro*), 524
 muxConfig_default (*C macro*), 524
 muxConfig_splitMode (*C macro*), 524
 muxEnable (*C macro*), 524
 muxSplit (*C macro*), 524
 muxVoltage (*C macro*), 524

O

OS_NEW_LN (*C macro*), 500

P

Pointer (*class in brainstem.entity*), 319
 pointer_getChar (*C++ function*), 660
 pointer_getInt (*C++ function*), 661
 pointer_getMode (*C++ function*), 658
 pointer_getOffset (*C++ function*), 658
 pointer_getShort (*C++ function*), 660
 pointer_getTransferStore (*C++ function*), 659
 pointer_initiateTransferFromStore (*C++ function*), 659

pointer_initiateTransferToStore (C++ function), 659
pointer_setChar (C++ function), 660
pointer_setInt (C++ function), 661
pointer_setMode (C++ function), 658
pointer_setOffset (C++ function), 658
pointer_setShort (C++ function), 660
pointer_setTransferStore (C++ function), 659
pointerChar (C macro), 525
pointerInt (C macro), 525
pointerMode (C macro), 525
pointerModeIncrement (C macro), 525
pointerModeStatic (C macro), 525
pointerOffset (C macro), 525
pointerShort (C macro), 525
pointerTransferFromStore (C macro), 525
pointerTransferStore (C macro), 525
pointerTransferToStore (C macro), 525
Port (class in *brainstem.entity*), 322
port_getAllocatedPower (C++ function), 673
port_getAvailablePower (C++ function), 673
port_getCC1Enabled (C++ function), 669
port_getCC2Enabled (C++ function), 670
port_getCCEnabled (C++ function), 669
port_getCurrentLimit (C++ function), 672
port_getCurrentLimitMode (C++ function), 673
port_getDataEnabled (C++ function), 663
port_getDataHS1Enabled (C++ function), 664
port_getDataHS2Enabled (C++ function), 665
port_getDataHSEnabled (C++ function), 664
port_getDataHSRoutingBehavior (C++ function), 675
port_getDataRole (C++ function), 667
port_getDataSpeed (C++ function), 671
port_getDataSS1Enabled (C++ function), 666
port_getDataSS2Enabled (C++ function), 666
port_getDataSSEnabled (C++ function), 665
port_getDataSSRoutingBehavior (C++ function), 676
port_getEnabled (C++ function), 663
port_getErrors (C++ function), 672
port_getHSBoost (C++ function), 677
port_getMode (C++ function), 671
port_getName (C++ function), 675
port_getPowerEnabled (C++ function), 667
port_getPowerLimit (C++ function), 674
port_getPowerLimitMode (C++ function), 674
port_getPowerMode (C++ function), 662
port_getState (C++ function), 671
port_getVbusAccumulatedPower (C++ function), 676
port_getVbusCurrent (C++ function), 662
port_getVbusVoltage (C++ function), 661
port_getVconn1Enabled (C++ function), 668
port_getVconn2Enabled (C++ function), 668
port_getVconnAccumulatedPower (C++ function), 677
port_getVconnCurrent (C++ function), 662
port_getVconnEnabled (C++ function), 667
port_getVconnVoltage (C++ function), 662
port_getVoltageSetpoint (C++ function), 670
port_resetEntityToFactoryDefaults (C++ function), 678
port_resetVbusAccumulatedPower (C++ function), 676
port_resetVconnAccumulatedPower (C++ function), 677
port_setCC1Enabled (C++ function), 670
port_setCC2Enabled (C++ function), 670
port_setCCEnabled (C++ function), 669
port_setCurrentLimit (C++ function), 672
port_setCurrentLimitMode (C++ function), 673
port_setDataEnabled (C++ function), 663
port_setDataHS1Enabled (C++ function), 664
port_setDataHS2Enabled (C++ function), 665
port_setDataHSEnabled (C++ function), 664

port_setDataHSRoutingBehavior (C++ function), 675
 port_setDataSS1Enabled (C++ function), 666
 port_setDataSS2Enabled (C++ function), 666
 port_setDataSSEnabled (C++ function), 665
 port_setDataSSRoutingBehavior (C++ function), 676
 port_setEnabled (C++ function), 663
 port_setHSBoost (C++ function), 677
 port_setMode (C++ function), 672
 port_setName (C++ function), 675
 port_setPowerEnabled (C++ function), 667
 port_setPowerLimit (C++ function), 674
 port_setPowerLimitMode (C++ function), 674
 port_setPowerMode (C++ function), 663
 port_setVconn1Enabled (C++ function), 668
 port_setVconn2Enabled (C++ function), 669
 port_setVconnEnabled (C++ function), 668
 port_setVoltageSetpoint (C++ function), 671
 PORT_SPEED (C++ enum), 560
 PORT_SPEED::kPORT_SPEED_FULL (C++ enumerator), 560
 PORT_SPEED::kPORT_SPEED_HIGH (C++ enumerator), 560
 PORT_SPEED::kPORT_SPEED_LOW (C++ enumerator), 560
 PORT_SPEED::kPORT_SPEED_SUPER (C++ enumerator), 560
 PORT_SPEED::kPORT_SPEED_SUPER_PLUS (C++ enumerator), 560
 PORT_SPEED::kPORT_SPEED_UNKNOWN (C++ enumerator), 560
 PowerDelivery (class in *brainstem.entity*), 333
 powerdelivery_getCableCurrentMax (C++ function), 683
 powerdelivery_getCableOrientation (C++ function), 684
 powerdelivery_getCableSpeedMax (C++ function), 684
 powerdelivery_getCableType (C++ function), 684
 powerdelivery_getCableVoltageMax (C++ function), 683
 powerdelivery_getConnectionState (C++ function), 678
 powerdelivery_getFastRoleSwapCurrent (C++ function), 687
 powerdelivery_getFlagMode (C++ function), 686
 powerdelivery_getNumberOfPowerDataObjects (C++ function), 678
 powerdelivery_getOverride (C++ function), 685
 powerdelivery_getPeakCurrentConfiguration (C++ function), 686
 powerdelivery_getPowerDataObject (C++ function), 679
 powerdelivery_getPowerDataObjectEnabled (C++ function), 680
 powerdelivery_getPowerDataObjectEnabledList (C++ function), 681
 powerdelivery_getPowerDataObjectList (C++ function), 680
 powerdelivery_getPowerRole (C++ function), 682
 powerdelivery_getPowerRolePreferred (C++ function), 683
 powerdelivery_getRequestDataObject (C++ function), 681
 powerdelivery_packDataObjectAttributes (C++ function), 688
 powerdelivery_request (C++ function), 684
 powerdelivery_requestStatus (C++ function), 685
 powerdelivery_resetEntityToFactoryDefaults (C++ function), 686
 powerdelivery_resetPowerDataObjectToDefault (C++ function), 679
 powerdelivery_setFastRoleSwapCurrent (C++ function), 687
 powerdelivery_setFlagMode (C++ function), 686
 powerdelivery_setOverride (C++ function), 685
 powerdelivery_setPeakCurrentConfiguration (C++ function), 687
 powerdelivery_setPowerDataObject (C++ function), 679
 powerdelivery_setPowerDataObjectEnabled (C++ function), 681
 powerdelivery_setPowerRole (C++ function), 682
 powerdelivery_setPowerRolePreferred (C++ function), 683
 powerdelivery_setRequestDataObject (C++ function), 682
 powerdelivery_unpackDataObjectAttributes (C++ function), 688
 prep_UIBytes () (*brainstem.module.Entity* method), 312

R

Rail (class in *brainstem.entity*), 340
 rail_clearFaults (C++ function), 696
 rail_getCurrent (C++ function), 689
 rail_getCurrentLimit (C++ function), 690
 rail_getCurrentSetpoint (C++ function), 689
 rail_getEnable (C++ function), 690

`rail_getKelvinSensingEnable` (*C++ function*), 695
`rail_getKelvinSensingState` (*C++ function*), 695
`rail_getOperationalMode` (*C++ function*), 696
`rail_getOperationalState` (*C++ function*), 696
`rail_getPower` (*C++ function*), 693
`rail_getPowerLimit` (*C++ function*), 694
`rail_getPowerSetpoint` (*C++ function*), 693
`rail_getResistance` (*C++ function*), 694
`rail_getResistanceSetpoint` (*C++ function*), 694
`rail_getTemperature` (*C++ function*), 690
`rail_getVoltage` (*C++ function*), 691
`rail_getVoltageMaxLimit` (*C++ function*), 692
`rail_getVoltageMinLimit` (*C++ function*), 692
`rail_getVoltageSetpoint` (*C++ function*), 691
`rail_setCurrentLimit` (*C++ function*), 690
`rail_setCurrentSetpoint` (*C++ function*), 689
`rail_setEnable` (*C++ function*), 691
`rail_setKelvinSensingEnable` (*C++ function*), 695
`rail_setOperationalMode` (*C++ function*), 695
`rail_setPowerLimit` (*C++ function*), 693
`rail_setPowerSetpoint` (*C++ function*), 693
`rail_setResistanceSetpoint` (*C++ function*), 694
`rail_setVoltageMaxLimit` (*C++ function*), 692
`rail_setVoltageMinLimit` (*C++ function*), 692
`rail_setVoltageSetpoint` (*C++ function*), 691
`railClearFaults` (*C macro*), 533
`railCurrent` (*C macro*), 530
`railCurrentLimit` (*C macro*), 530
`railCurrentSetpoint` (*C macro*), 533
`railEnable` (*C macro*), 530
`railFactoryReserved` (*C macro*), 533
`railFactoryReserved2` (*C macro*), 534
`railKelvinSensingEnable` (*C macro*), 530
`railKelvinSensingState` (*C macro*), 531
`railOperationalMode` (*C macro*), 531
`railOperationalMode_HardwareConfiguration_Offset` (*C macro*), 531
`railOperationalMode_Mode_Offset` (*C macro*), 531
`railOperationalModeAuto_Value` (*C macro*), 531
`railOperationalModeConstantCurrent_Value` (*C macro*), 531
`railOperationalModeConstantPower_Value` (*C macro*), 531
`railOperationalModeConstantResistance_Value` (*C macro*), 531
`railOperationalModeConstantVoltage_Value` (*C macro*), 531
`railOperationalModeFactoryReserved_Value` (*C macro*), 531
`railOperationalModeLinear_Value` (*C macro*), 531
`railOperationalModeSwitcher_Value` (*C macro*), 531
`railOperationalModeSwitcherLinear_Value` (*C macro*), 531
`railOperationalState` (*C macro*), 531
`railOperationalState_Enabled_Bit` (*C macro*), 532
`railOperationalState_Fault_Bit` (*C macro*), 532
`railOperationalState_HardwareConfiguration_Offset` (*C macro*), 532
`railOperationalState_Initializing_Bit` (*C macro*), 532
`railOperationalStateConstantCurrent_Value` (*C macro*), 533
`railOperationalStateConstantPower_Value` (*C macro*), 533
`railOperationalStateConstantResistance_Value` (*C macro*), 533
`railOperationalStateConstantVoltage_Value` (*C macro*), 533
`railOperationalStateLinear_Value` (*C macro*), 532
`railOperationalStateOperatingMode_Offset` (*C macro*), 532
`railOperationalStateOverCurrentFault_Bit` (*C macro*), 532
`railOperationalStateOverPowerFault_Bit` (*C macro*), 532
`railOperationalStateOverTemperatureFault_Bit` (*C macro*), 532
`railOperationalStateOverVoltageFault_Bit` (*C macro*), 532
`railOperationalStateReverseCurrentFault_Bit` (*C macro*), 532
`railOperationalStateReversePolarityFault_Bit` (*C macro*), 532
`railOperationalStateSwitcher_Value` (*C macro*), 532
`railOperationalStateSwitcherLinear_Value` (*C macro*), 532
`railOperationalStateUnderVoltageFault_Bit` (*C macro*), 532
`railPower` (*C macro*), 533
`railPowerLimit` (*C macro*), 533

railPowerSetpoint (*C macro*), 533
 railResistance (*C macro*), 533
 railResistanceSetpoint (*C macro*), 533
 railTemperature (*C macro*), 530
 railValue (*C macro*), 530
 railVoltage (*C macro*), 530
 railVoltageMaxLimit (*C macro*), 533
 railVoltageMinLimit (*C macro*), 533
 railVoltageSetpoint (*C macro*), 533
 RCServo (*class in brainstem.entity*), 347
 rcservo_getEnable (*C++ function*), 697
 rcservo_getPosition (*C++ function*), 697
 rcservo_getReverse (*C++ function*), 698
 rcservo_setEnable (*C++ function*), 697
 rcservo_setPosition (*C++ function*), 697
 rcservo_setReverse (*C++ function*), 698
 read () (*brainstem.entity.I2C method*), 308
 reconnect () (*brainstem.module.Module method*), 316
 Relay (*class in brainstem.entity*), 348
 relay_getEnable (*C++ function*), 699
 relay_getVoltage (*C++ function*), 699
 relay_setEnable (*C++ function*), 698
 request () (*brainstem.entity.PowerDelivery method*), 337
 requestStatus () (*brainstem.entity.PowerDelivery method*), 337
 reset () (*brainstem.entity.System method*), 356
 reset () (*brainstem.entity.Temperature method*), 361
 resetDeviceToFactoryDefaults () (*brainstem.entity.System method*), 356
 resetEntityToFactoryDefaults () (*brainstem.entity.Port method*), 328
 resetEntityToFactoryDefaults () (*brainstem.entity.PowerDelivery method*), 337
 resetEntityToFactoryDefaults () (*brainstem.entity.System method*), 356
 resetEntityToFactoryDefaults () (*brainstem.entity.Temperature method*), 362
 resetEntityToFactoryDefaults () (*brainstem.entity.USBSystem method*), 373
 resetPowerDataObjectToDefault () (*brainstem.entity.PowerDelivery method*), 337
 resetVbusAccumulatedPower () (*brainstem.entity.Port method*), 328
 resetVconnAccumulatedPower () (*brainstem.entity.Port method*), 328
 Result (*class in brainstem.result*), 349
 routeToMe () (*brainstem.entity.System method*), 356

S

save () (*brainstem.entity.System method*), 357
 SERIAL (*brainstem.link.Spec attribute*), 310
 serial_number (*brainstem.link.Spec attribute*), 309
 serial_port (*brainstem.link.Spec attribute*), 310
 set_UEI16 () (*brainstem.module.Entity method*), 313
 set_UEI32 () (*brainstem.module.Entity method*), 313
 set_UEI32_with_subindex () (*brainstem.module.Entity method*), 313
 set_UEI8 () (*brainstem.module.Entity method*), 313
 set_UEI8_with_subindex () (*brainstem.module.Entity method*), 314
 set_UEIBytes () (*brainstem.module.Entity method*), 314
 set_usbPortStateCOM_ORIENT_STATUS (*C macro*), 392
 set_usbPortStateMUX_ORIENT_STATUS (*C macro*), 392
 set_usbPortStateSPEED_STATUS (*C macro*), 392
 setAltModeConfig () (*brainstem.entity.USB method*), 367
 setBaudRate () (*brainstem.entity.UART method*), 363
 setBootSlot () (*brainstem.entity.System method*), 357
 setBulkCaptureNumberOfSamples () (*brainstem.entity.Analog method*), 290
 setBulkCaptureSampleRate () (*brainstem.entity.Analog method*), 290
 setCableFlip () (*brainstem.entity.USB method*), 368
 setCC1Enable () (*brainstem.entity.USB method*), 368
 setCC1Enabled () (*brainstem.entity.Port method*), 328
 setCC2Enable () (*brainstem.entity.USB method*), 368
 setCC2Enabled () (*brainstem.entity.Port method*), 328
 setCCEnabled () (*brainstem.entity.Port method*), 329
 setChannel () (*brainstem.entity.Mux method*), 318
 setChar () (*brainstem.entity.Pointer method*), 320
 setConfiguration () (*brainstem.entity.Analog method*), 291
 setConfiguration () (*brainstem.entity.Digital method*), 297

setConfiguration() (*brainstem.entity.Mux method*), 318
setConnectMode() (*brainstem.entity.USB method*), 368
setCurrentLimit() (*brainstem.entity.Port method*), 329
setCurrentLimit() (*brainstem.entity.Rail method*), 344
setCurrentLimitMode() (*brainstem.entity.Port method*), 329
setCurrentSetpoint() (*brainstem.entity.Rail method*), 344
setDataDisable() (*brainstem.entity.USB method*), 368
setDataEnable() (*brainstem.entity.USB method*), 368
setDataEnabled() (*brainstem.entity.Port method*), 329
setDataHS1Enabled() (*brainstem.entity.Port method*), 329
setDataHS2Enabled() (*brainstem.entity.Port method*), 329
setDataHSEnabled() (*brainstem.entity.Port method*), 330
setDataHSRoutingBehavior() (*brainstem.entity.Port method*), 330
setDataRoleBehavior() (*brainstem.entity.USBSystem method*), 373
setDataRoleBehaviorConfig() (*brainstem.entity.USBSystem method*), 374
setDataSS1Enabled() (*brainstem.entity.Port method*), 330
setDataSS2Enabled() (*brainstem.entity.Port method*), 330
setDataSSEnabled() (*brainstem.entity.Port method*), 330
setDataSSRoutingBehavior() (*brainstem.entity.Port method*), 330
setDay() (*brainstem.entity.Clock method*), 293
setDownstreamBoostMode() (*brainstem.entity.USB method*), 369
setEnabled() (*brainstem.entity.Analog method*), 291
setEnabled() (*brainstem.entity.Mux method*), 318
setEnabled() (*brainstem.entity.Rail method*), 344
setEnabled() (*brainstem.entity.RCServo method*), 347
setEnabled() (*brainstem.entity.Relay method*), 349
setEnabled() (*brainstem.entity.Signal method*), 350
setEnabled() (*brainstem.entity.UART method*), 363
setEnabled() (*brainstem.entity.Port method*), 331
setEnabledList() (*brainstem.entity.USBSystem method*), 374
setEnumerationDelay() (*brainstem.entity.USB method*), 369
setEnumerationDelay() (*brainstem.entity.USBSystem method*), 374
setExpiration() (*brainstem.entity.Timer method*), 362
setFastRoleSwapCurrent() (*brainstem.entity.PowerDelivery method*), 338
setFlagMode() (*brainstem.entity.PowerDelivery method*), 338
setHBInterval() (*brainstem.entity.System method*), 357
setHiSpeedDataDisable() (*brainstem.entity.USB method*), 369
setHiSpeedDataEnable() (*brainstem.entity.USB method*), 369
setHour() (*brainstem.entity.Clock method*), 294
setHSBoost() (*brainstem.entity.Port method*), 331
setHubMode() (*brainstem.entity.USB method*), 369
setInputPowerBehavior() (*brainstem.entity.System method*), 357
setInputPowerBehaviorConfig() (*brainstem.entity.System method*), 358
setInt() (*brainstem.entity.Pointer method*), 320
setInvert() (*brainstem.entity.Signal method*), 350
setKelvinSensingEnable() (*brainstem.entity.Rail method*), 345
setLED() (*brainstem.entity.System method*), 358
setLinkInterface() (*brainstem.entity.System method*), 358
setMinute() (*brainstem.entity.Clock method*), 294
setMode() (*brainstem.entity.Pointer method*), 320
setMode() (*brainstem.entity.Port method*), 331
setMode() (*brainstem.entity.Timer method*), 362
setModeList() (*brainstem.entity.USBSystem method*), 374
setModuleAddress() (*brainstem.module.Module method*), 316
setModuleSoftwareOffset() (*brainstem.entity.System method*), 358
setMonth() (*brainstem.entity.Clock method*), 294
setName() (*brainstem.entity.Port method*), 331
setName() (*brainstem.entity.System method*), 358
setNetworkingMode() (*brainstem.module.Module method*), 316
setOffset() (*brainstem.entity.Pointer method*), 320
setOperationalMode() (*brainstem.entity.Rail method*), 345
setOverride() (*brainstem.entity.PowerDelivery method*), 338
setPeakCurrentConfiguration() (*brainstem.entity.PowerDelivery method*), 338
setPortCurrentLimit() (*brainstem.entity.USB method*), 369
setPortDisable() (*brainstem.entity.USB method*), 370
setPortEnable() (*brainstem.entity.USB method*), 370
setPortMode() (*brainstem.entity.USB method*), 370
setPosition() (*brainstem.entity.RCServo method*), 347

setPowerBehavior() (*brainstem.entity.USBSystem method*), 374
 setPowerBehaviorConfig() (*brainstem.entity.USBSystem method*), 374
 setPowerDataObject() (*brainstem.entity.PowerDelivery method*), 338
 setPowerDataObjectEnabled() (*brainstem.entity.PowerDelivery method*), 339
 setPowerDisable() (*brainstem.entity.USB method*), 370
 setPowerEnable() (*brainstem.entity.USB method*), 370
 setPowerEnabled() (*brainstem.entity.Port method*), 331
 setPowerLimit() (*brainstem.entity.Port method*), 331
 setPowerLimit() (*brainstem.entity.Rail method*), 345
 setPowerLimitMax() (*brainstem.entity.System method*), 359
 setPowerLimitMode() (*brainstem.entity.Port method*), 332
 setPowerMode() (*brainstem.entity.Port method*), 332
 setPowerRole() (*brainstem.entity.PowerDelivery method*), 339
 setPowerRolePreferred() (*brainstem.entity.PowerDelivery method*), 339
 setPowerSetpoint() (*brainstem.entity.Rail method*), 345
 setProtocol() (*brainstem.entity.UART method*), 364
 setPullup() (*brainstem.entity.I2C method*), 308
 setRange() (*brainstem.entity.Analog method*), 291
 setReceiverConfig() (*brainstem.entity.Equalizer method*), 307
 setRequestDataObject() (*brainstem.entity.PowerDelivery method*), 339
 setResistanceSetpoint() (*brainstem.entity.Rail method*), 346
 setReverse() (*brainstem.entity.RCServo method*), 348
 setRouter() (*brainstem.entity.System method*), 359
 setSBUEnable() (*brainstem.entity.USB method*), 370
 setSecond() (*brainstem.entity.Clock method*), 294
 setSelectorMode() (*brainstem.entity.USBSystem method*), 374
 setShort() (*brainstem.entity.Pointer method*), 321
 setSlotLocked() (*brainstem.entity.Store method*), 360
 setSpeed() (*brainstem.entity.I2C method*), 308
 setSplitMode() (*brainstem.entity.Mux method*), 318
 setState() (*brainstem.entity.Digital method*), 297
 setStateAll() (*brainstem.entity.Digital method*), 297
 setSuperSpeedDataDisable() (*brainstem.entity.USB method*), 371
 setSuperSpeedDataEnable() (*brainstem.entity.USB method*), 371
 setT2Time() (*brainstem.entity.Signal method*), 350
 setT3Time() (*brainstem.entity.Signal method*), 350
 setTransferStore() (*brainstem.entity.Pointer method*), 321
 setTransmitterConfig() (*brainstem.entity.Equalizer method*), 307
 setUpstream() (*brainstem.entity.USBSystem method*), 375
 setUpstreamBoostMode() (*brainstem.entity.USB method*), 371
 setUpstreamMode() (*brainstem.entity.USB method*), 371
 setValue() (*brainstem.entity.Analog method*), 291
 setVconn1Enabled() (*brainstem.entity.Port method*), 332
 setVconn2Enabled() (*brainstem.entity.Port method*), 332
 setVconnEnabled() (*brainstem.entity.Port method*), 332
 setVoltage() (*brainstem.entity.Analog method*), 291
 setVoltageMaxLimit() (*brainstem.entity.Rail method*), 346
 setVoltageMinLimit() (*brainstem.entity.Rail method*), 346
 setVoltageSetpoint() (*brainstem.entity.Port method*), 332
 setVoltageSetpoint() (*brainstem.entity.Rail method*), 346
 setYear() (*brainstem.entity.Clock method*), 294
 Signal (*class in brainstem.entity*), 349
 signal_getEnable (*C++ function*), 699
 signal_getInvert (*C++ function*), 700
 signal_getT2Time (*C++ function*), 701
 signal_getT3Time (*C++ function*), 701
 signal_setEnable (*C++ function*), 699
 signal_setInvert (*C++ function*), 700
 signal_setT2Time (*C++ function*), 701
 signal_setT3Time (*C++ function*), 700
 slotCapacity (*C macro*), 523
 slotClose (*C macro*), 523
 slotDisable() (*brainstem.entity.Store method*), 361
 slotEnable() (*brainstem.entity.Store method*), 361
 slotOpenRead (*C macro*), 523
 slotOpenWrite (*C macro*), 523
 slotRead (*C macro*), 523
 slotSeek (*C macro*), 523

slotSize (*C macro*), 523
slotWrite (*C macro*), 523
spec (*brainstem.module.Module property*), 316
Spec (*class in brainstem.link*), 309
Status (*class in brainstem.link*), 310
Store (*class in brainstem.entity*), 359
store_getSlotCapacity (*C++ function*), 703
store_getSlotLocked (*C++ function*), 704
store_getSlotSize (*C++ function*), 703
store_getSlotState (*C++ function*), 702
store_loadSlot (*C++ function*), 702
store_setSlotLocked (*C++ function*), 704
store_slotDisable (*C++ function*), 703
store_slotEnable (*C++ function*), 702
store_unloadSlot (*C++ function*), 702
storeCloseSlot (*C macro*), 536
storeLock (*C macro*), 536
storeNumberOfOptions (*C macro*), 536
storeReadSlot (*C macro*), 536
storeSlotDisable (*C macro*), 536
storeSlotEnable (*C macro*), 536
storeSlotState (*C macro*), 536
storeWriteSlot (*C macro*), 536
System (*class in brainstem.entity*), 351
system_getBootSlot (*C++ function*), 706
system_getHardwareVersion (*C++ function*), 707
system_getHBInterval (*C++ function*), 705
system_getInputCurrent (*C++ function*), 709
system_getInputPowerBehavior (*C++ function*), 712
system_getInputPowerBehaviorConfig (*C++ function*), 713
system_getInputPowerSource (*C++ function*), 712
system_getInputVoltage (*C++ function*), 709
system_getLED (*C++ function*), 706
system_getLinkInterface (*C++ function*), 714
system_getMaximumTemperature (*C++ function*), 709
system_getMinimumTemperature (*C++ function*), 708
system_getModel (*C++ function*), 707
system_getModule (*C++ function*), 704
system_getModuleBaseAddress (*C++ function*), 705
system_getModuleHardwareOffset (*C++ function*), 709
system_getModuleSoftwareOffset (*C++ function*), 710
system_getName (*C++ function*), 713
system_getPowerLimit (*C++ function*), 711
system_getPowerLimitMax (*C++ function*), 711
system_getPowerLimitState (*C++ function*), 711
system_getRouter (*C++ function*), 705
system_getRouterAddressSetting (*C++ function*), 710
system_getSerialNumber (*C++ function*), 707
system_getTemperature (*C++ function*), 708
system_getUnregulatedCurrent (*C++ function*), 712
system_getUnregulatedVoltage (*C++ function*), 711
system_getUptime (*C++ function*), 708
system_getVersion (*C++ function*), 707
system_logEvents (*C++ function*), 708
system_reset (*C++ function*), 708
system_resetDeviceToFactoryDefaults (*C++ function*), 714
system_resetEntityToFactoryDefaults (*C++ function*), 714
system_routeToMe (*C++ function*), 710
system_save (*C++ function*), 707
system_setBootSlot (*C++ function*), 706
system_setHBInterval (*C++ function*), 705
system_setInputPowerBehavior (*C++ function*), 712
system_setInputPowerBehaviorConfig (*C++ function*), 713
system_setLED (*C++ function*), 706
system_setLinkInterface (*C++ function*), 714
system_setModuleSoftwareOffset (*C++ function*), 710
system_setName (*C++ function*), 713
system_setPowerLimitMax (*C++ function*), 711

system_setRouter (C++ function), 705
 systemBootSlot (C macro), 519
 systemErrors (C macro), 522
 systemErrors_OutputPowerProtection_Bit (C macro), 522
 systemErrors_ThermalProtection_Bit (C macro), 522
 systemHardwareVersion (C macro), 522
 systemHBInterval (C macro), 519
 systemInputCurrent (C macro), 520
 systemInputPowerBehavior (C macro), 521
 systemInputPowerBehaviorConfig (C macro), 521
 systemInputPowerSource (C macro), 521
 systemInputVoltage (C macro), 519
 systemIPAddress (C macro), 520
 systemIPConfiguration (C macro), 520
 systemIPModeDefault (C macro), 520
 systemIPModeDHCP (C macro), 520
 systemIPModeStatic (C macro), 520
 systemIPStaticAddressSetting (C macro), 520
 systemLED (C macro), 519
 systemLinkAuto (C macro), 521
 systemLinkInterface (C macro), 521
 systemLinkUSBControl (C macro), 521
 systemLinkUSBHub (C macro), 522
 systemLogEvents (C macro), 520
 systemMaxTemperature (C macro), 520
 systemMinTemperature (C macro), 521
 systemModel (C macro), 519
 systemModule (C macro), 519
 systemModuleBaseAddress (C macro), 520
 systemModuleHardwareOffset (C macro), 520
 systemModuleSoftwareOffset (C macro), 520
 systemName (C macro), 521
 systemNumberOfOptions (C macro), 522
 systemPowerLimit (C macro), 521
 systemPowerLimitMax (C macro), 521
 systemPowerLimitState (C macro), 521
 systemReserved (C macro), 522
 systemReset (C macro), 519
 systemResetDeviceToFactoryDefaults (C macro), 521
 systemResetEntityToFactoryDefaults (C macro), 521
 systemRouter (C macro), 519
 systemRouterAddressSetting (C macro), 520
 systemRouteToMe (C macro), 520
 systemSave (C macro), 519
 systemSerialNumber (C macro), 519
 systemSleep (C macro), 519
 systemTemperature (C macro), 521
 systemUnregulatedCurrent (C macro), 521
 systemUnregulatedVoltage (C macro), 520
 systemUptime (C macro), 520
 systemVersion (C macro), 519

T

tcp_port (brainstem.link.Spec attribute), 310
 TCP/IP (brainstem.link.Spec attribute), 310
 Temperature (class in brainstem.entity), 361
 temperature_getValue (C++ function), 715
 temperature_getValueMax (C++ function), 715
 temperature_getValueMin (C++ function), 715
 temperature_resetEntityToFactoryDefaults (C++ function), 715
 temperatureMaximumMicroCelsius (C macro), 534
 temperatureMicroCelsius (C macro), 534
 temperatureMinimumMicroCelsius (C macro), 534
 temperatureNumberOfOptions (C macro), 534
 temperatureResetEntityToFactoryDefaults (C macro), 534
 Timer (class in brainstem.entity), 362
 timer_getExpiration (C++ function), 716

timer_getMode (C++ function), 716
timer_setExpiration (C++ function), 716
timer_setMode (C++ function), 716
timerExpiration (C macro), 537
timerMode (C macro), 537
timerModeRepeat (C macro), 537
timerModeSingle (C macro), 537
transferToStore () (brainstem.entity.Pointer method), 321
transport (brainstem.link.Spec attribute), 309
transferFromStore () (brainstem.entity.Pointer method), 321

U

UART (class in brainstem.entity), 363
uart_getEnable (C++ function), 717
uart_setEnable (C++ function), 717
uei (C++ struct), 556
uei::byteVal (C++ member), 557
uei::command (C++ member), 557
uei::intVal (C++ member), 557
uei::module (C++ member), 557
uei::option (C++ member), 557
uei::shortVal (C++ member), 557
uei::specifier (C++ member), 557
uei::type (C++ member), 557
ueiBYTES_CONTINUE (C macro), 518
ueiBYTES_CONTINUE_MASK (C macro), 518
ueiOPTION_ACK (C macro), 518
ueiOPTION_GET (C macro), 518
ueiOPTION_MASK (C macro), 518
ueiOPTION_OP_MASK (C macro), 518
ueiOPTION_SET (C macro), 518
ueiOPTION_VAL (C macro), 518
ueiREPLY_ERROR (C macro), 518
ueiREPLY_STREAM (C macro), 518
ueiSPECIFIER_INDEX_MASK (C macro), 517
ueiSPECIFIER_RETURN_HOST (C macro), 517
ueiSPECIFIER_RETURN_I2C (C macro), 517
ueiSPECIFIER_RETURN_MASK (C macro), 517
ueiSPECIFIER_RETURN_VM (C macro), 518
unloadSlot () (brainstem.entity.Store method), 361
unpack_version () (in module brainstem.version), 375
USB (brainstem.link.Spec attribute), 310
USB (class in brainstem.entity), 364
usb_clearPortErrorStatus (C++ function), 721
usb_getAltModeConfig (C++ function), 729
usb_getCableFlip (C++ function), 729
usb_getCC1Current (C++ function), 727
usb_getCC1Enable (C++ function), 726
usb_getCC1Voltage (C++ function), 727
usb_getCC2Current (C++ function), 727
usb_getCC2Enable (C++ function), 727
usb_getCC2Voltage (C++ function), 727
usb_getConnectMode (C++ function), 725
usb_getDownstreamBoostMode (C++ function), 724
usb_getDownstreamDataSpeed (C++ function), 725
usb_getEnumerationDelay (C++ function), 722
usb_getHubMode (C++ function), 720
usb_getPortCurrent (C++ function), 720
usb_getPortCurrentLimit (C++ function), 722
usb_getPortError (C++ function), 723
usb_getPortMode (C++ function), 723
usb_getPortState (C++ function), 723
usb_getPortVoltage (C++ function), 720
usb_getSBU1Voltage (C++ function), 729
usb_getSBU2Voltage (C++ function), 730
usb_getSBUEnable (C++ function), 728
usb_getUpstreamBoostMode (C++ function), 724

usb_getUpstreamMode (C++ function), 721
 usb_getUpstreamState (C++ function), 721
 usb_setAltModeConfig (C++ function), 729
 usb_setCableFlip (C++ function), 728
 usb_setCC1Enable (C++ function), 726
 usb_setCC2Enable (C++ function), 726
 usb_setConnectMode (C++ function), 725
 usb_setDataDisable (C++ function), 718
 usb_setDataEnable (C++ function), 718
 usb_setDownstreamBoostMode (C++ function), 724
 usb_setEnumerationDelay (C++ function), 722
 usb_setHiSpeedDataDisable (C++ function), 719
 usb_setHiSpeedDataEnable (C++ function), 718
 usb_setHubMode (C++ function), 720
 usb_setPortCurrentLimit (C++ function), 722
 usb_setPortDisable (C++ function), 718
 usb_setPortEnable (C++ function), 718
 usb_setPortMode (C++ function), 723
 usb_setPowerDisable (C++ function), 720
 usb_setPowerEnable (C++ function), 719
 usb_setSBUEnable (C++ function), 728
 usb_setSuperSpeedDataDisable (C++ function), 719
 usb_setSuperSpeedDataEnable (C++ function), 719
 usb_setUpstreamBoostMode (C++ function), 724
 usb_setUpstreamMode (C++ function), 721
 usbAltMode (C macro), 543
 usbAltMode_2LaneDP_ComToHost_wUSB3 (C macro), 543
 usbAltMode_2LaneDP_ComToHost_wUSB3_Inverted (C macro), 544
 usbAltMode_2LaneDP_MuxToHost_wUSB3 (C macro), 543
 usbAltMode_2LaneDP_MuxToHost_wUSB3_Inverted (C macro), 544
 usbAltMode_4LaneDP_ComToHost (C macro), 543
 usbAltMode_4LaneDP_MuxToHost (C macro), 543
 usbAltMode_disabled (C macro), 543
 usbAltMode_normal (C macro), 543
 usbAutoConnect (C macro), 542
 usbBoostMode_0 (C macro), 540
 usbBoostMode_12 (C macro), 540
 usbBoostMode_4 (C macro), 540
 usbBoostMode_8 (C macro), 540
 usbCableFlip (C macro), 543
 usbCC1Current (C macro), 543
 usbCC1Enable (C macro), 542
 usbCC1Voltage (C macro), 543
 usbCC2Current (C macro), 543
 usbCC2Enable (C macro), 542
 usbCC2Voltage (C macro), 543
 usbConnectMode (C macro), 542
 USBCSwitch (class in *brainstem.stem*), 272
 usbDataDisable (C macro), 538
 usbDataEnable (C macro), 538
 usbDownstreamBoostMode (C macro), 540
 usbDownstreamDataSpeed (C macro), 542
 usbDownstreamDataSpeed_hs (C macro), 542
 usbDownstreamDataSpeed_ls (C macro), 542
 usbDownstreamDataSpeed_na (C macro), 542
 usbDownstreamDataSpeed_ss (C macro), 542
 usbHiSpeedDataDisable (C macro), 542
 usbHiSpeedDataEnable (C macro), 542
 USBHub2x4 (class in *brainstem.stem*), 271
 USBHub3p (class in *brainstem.stem*), 270
 usbHubEnumerationDelay (C macro), 540
 usbHubMode (C macro), 539
 usbManualConnect (C macro), 542
 usbPortClearErrorStatus (C macro), 539
 usbPortCurrent (C macro), 539
 usbPortCurrentLimit (C macro), 540
 usbPortDisable (C macro), 538
 usbPortEnable (C macro), 538

usbPortError (*C macro*), 543
usbPortMode (*C macro*), 540
usbPortMode_AutoConnectEnable (*C macro*), 541
usbPortMode_CC1Enable (*C macro*), 541
usbPortMode_CC1InjectEnable (*C macro*), 541
usbPortMode_CC2Enable (*C macro*), 541
usbPortMode_CC2InjectEnable (*C macro*), 542
usbPortMode_CCFFlipEnable (*C macro*), 541
usbPortMode_cdp (*C macro*), 540
usbPortMode_charging (*C macro*), 540
usbPortMode_passive (*C macro*), 540
usbPortMode_SBUEnable (*C macro*), 541
usbPortMode_SBUFlipEnable (*C macro*), 541
usbPortMode_sdp (*C macro*), 540
usbPortMode_SSFlipEnable (*C macro*), 541
usbPortMode_SuperSpeed1Enable (*C macro*), 541
usbPortMode_SuperSpeed2Enable (*C macro*), 541
usbPortMode_USB2AEnable (*C macro*), 540
usbPortMode_USB2BEnable (*C macro*), 541
usbPortMode_USB2BoostEnable (*C macro*), 541
usbPortMode_USB2FlipEnable (*C macro*), 541
usbPortMode_USB3BoostEnable (*C macro*), 541
usbPortMode_VBusEnable (*C macro*), 541
usbPortState (*C macro*), 543
usbPortStateCC1 (*C macro*), 392
usbPortStateCC1Detect (*C macro*), 393
usbPortStateCC1Inject (*C macro*), 393
usbPortStateCC1LogicState (*C macro*), 393
usbPortStateCC2 (*C macro*), 392
usbPortStateCC2Detect (*C macro*), 393
usbPortStateCC2Inject (*C macro*), 393
usbPortStateCC2LogicState (*C macro*), 393
usbPortStateCCFlip (*C macro*), 392
usbPortStateConnectionEstablished (*C macro*), 393
usbPortStateErrorFlag (*C macro*), 393
usbPortStateOff (*C macro*), 394
usbPortStateSBU (*C macro*), 392
usbPortStateSBUFlip (*C macro*), 392
usbPortStateSideA (*C macro*), 394
usbPortStateSideB (*C macro*), 394
usbPortStateSideUndefined (*C macro*), 394
usbPortStateSS1 (*C macro*), 392
usbPortStateSS2 (*C macro*), 392
usbPortStateSSFlip (*C macro*), 392
usbPortStateUSB2A (*C macro*), 392
usbPortStateUSB2B (*C macro*), 392
usbPortStateUSB2Boost (*C macro*), 393
usbPortStateUSB2Flip (*C macro*), 393
usbPortStateUSB3Boost (*C macro*), 393
usbPortStateVBUS (*C macro*), 392
usbPortVoltage (*C macro*), 539
usbPowerDisable (*C macro*), 539
usbPowerEnable (*C macro*), 539
usbSBU1Voltage (*C macro*), 544
usbSBU2Voltage (*C macro*), 544
usbSBUEnable (*C macro*), 543
USBStem (*class in brainstem.stem*), 285
usbSuperSpeedDataDisable (*C macro*), 542
usbSuperSpeedDataEnable (*C macro*), 542
USBSystem (*class in brainstem.entity*), 372
usbsystem_getDataRoleBehavior (*C++ function*), 734
usbsystem_getDataRoleBehaviorConfig (*C++ function*), 734
usbsystem_getDataRoleList (*C++ function*), 731
usbsystem_getEnabledList (*C++ function*), 731
usbsystem_getEnumerationDelay (*C++ function*), 730
usbsystem_getModeList (*C++ function*), 732
usbsystem_getPowerBehavior (*C++ function*), 733
usbsystem_getPowerBehaviorConfig (*C++ function*), 733

usbsystem_getStateList (*C++ function*), 732
usbsystem_getUpstream (*C++ function*), 730
usbsystem_resetEntityToFactoryDefaults (*C++ function*), 735
usbsystem_setDataRoleBehavior (*C++ function*), 734
usbsystem_setDataRoleBehaviorConfig (*C++ function*), 735
usbsystem_setEnabledList (*C++ function*), 731
usbsystem_setEnumerationDelay (*C++ function*), 731
usbsystem_setModeList (*C++ function*), 732
usbsystem_setPowerBehavior (*C++ function*), 733
usbsystem_setPowerBehaviorConfig (*C++ function*), 733
usbsystem_setUpstream (*C++ function*), 730
usbUpstreamBoostMode (*C macro*), 540
usbUpstreamMode (*C macro*), 539
usbUpstreamModeAuto (*C macro*), 539
usbUpstreamModeDefault (*C macro*), 539
usbUpstreamModeNone (*C macro*), 539
usbUpstreamModePort0 (*C macro*), 539
usbUpstreamModePort1 (*C macro*), 539
usbUpstreamState (*C macro*), 539
usbUpstreamStateNone (*C macro*), 539
usbUpstreamStatePort0 (*C macro*), 539
usbUpstreamStatePort1 (*C macro*), 540

V

VALIDPACKET (*C++ member*), 515
value (*brainstem.result.Result property*), 349

W

write() (*brainstem.entity.I2C method*), 308