BrainStem Reference Manual

Release 2.11.1

Acroname, Inc.

Mar 05, 2025

Contents

1 Devices

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

1.1 USBHub3p

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

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

1.1.1 Quick Start Guide

Power

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

Data

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

Download

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

Play

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

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

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

1.1.2 Basic Example

C++

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

static const uint8_t PORT = 5;

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

    //Create an instance of the USBHub3p
    aUSBHub3p hub;

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

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

(continues on next page)

[1] https://acroname.com/api
[2] https://acroname.com/hubtool

(continued from previous page)

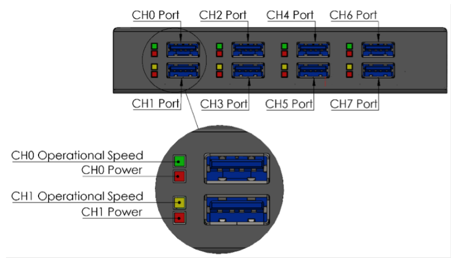
```
    ////////////
    //Do Stuff
    ////////////

    //Enable PORT
    hub.usb.setPortDisable(PORT);

    //Disconnect
    hub.disconnect();

    return 0
}
```

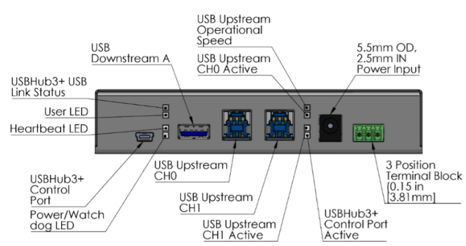
Python

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

PORT = 5

#Create an instance of the USBHub3p
hub = brainstem.stem.USBHub3p()

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

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

hub.usb.setPortEnable(PORT)

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

hub.usb.setPortDisable(PORT)

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

1.1.3 Indicators and Connections

Connections





LEDs

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

1.1.4 Programming Interface

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

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

Software Control

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

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

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

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

BrainStem Control Port

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

Using Multiple Hosts with USBHub3+

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

Device Drivers

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

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

1.1.5 USBHub3+ Module Entities

Temperature

API Documentation: *[cpp] [python] [.NET] [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-degress 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, µV)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.getPortCurrent(channel, µA)` *[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, µA)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.setPortCurrentLimit(channel, µA)` *[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 of 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.getEnumberationDelay(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

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

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

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

`stem.hub.port[x].setEnabled()` *[cpp] [python] [NET] [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 0-7.

`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 0-7.

`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 0-7.

`stem.hub.port[x].setDataHSEnabled()` *[cpp] [python] [NET] [LabVIEW]*
`stem.hub.port[x].getDataHSEnabled()` *[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 0-7.

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

Voltage and Current Measurements

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

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

Power Modes

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

`stem.hub.port[x].setPowerMode()` *[cpp] [python] [NET] [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

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

Port Mode

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

`stem.hub.port[x].setMode()` *[cpp] [python] [NET] [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
SS 1 Enable	3	0/1	portPortMode_SS1Enabled_Bit
Power Mode: Offset		16	portPortMode_portPowerMode_Offset
Power Mode: Mask		0x7	portPortMode_portPowerMode_Mask
Power Mode: None	16-18	0	portPortMode_portPowerMode_none_Value
Power Mode: SDP	16-18	1	portPortMode_portPowerMode_sdp_Value
Power Mode: CDP/DCP	16-18	2	portPortMode_cdp_dcp_Value

Data Role

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

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

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

Downstream Data Speed

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

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

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

USB System

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

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

Upstream Control

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

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

Enumeration Delay

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

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

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

Data Behavior

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

List of Available Data Behaviors for USBHub3p

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

Hard Coded (Default Configuration)

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

Port Priority

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

Relevant API's

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

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

Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
App[0-3]	execute	
	return	
system[0]	getModel	
	getHardwareVersion	
	getModule	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getVersion	
	getSerialNumber	
	save	
	reset	
	getInputVoltage	
	getInputCurrent	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	getUptime	
	getMaximumTemperature	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
timer[0-8]	getExpiration	
	setExpiration	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	
	getTransferStore	
	setTransferStore	
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	

continues on next page

Table 2 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	setInt	
temperature[0]	getTemperature	
usb[0]	setPortEnable	Channels 0-7
	setPortDisable	Channels0-7
	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	
	getEnumerationDelay	
	setEnumerationDelay	
	clearPortErrorStatus	
	getUpstreamMode	
	setUpstreamMode	
	getUpstreamState	
	getUpstreamBoostMode	
	setUpstreamBoostMode	
	getDownstreamBoostMode	
	setDownstreamBoostMode	
port[0-7]	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getDataSSEnabled	
	setDataSSEnabled	
	getPowerEnabled	
	setPowerEnabled	
	getHSBoost	
	setHSBoost	
	getMode	
	setMode	
	getCurrentLimit	
	setCurrentLimit	
	getVbusVoltage	

continues on next page

Table 2 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getVbusCurrent	
	getState	
	getName	
	setName	
	getPowerMode	
	setPowerMode	
	getDataRole	
	getDataSpeed	
	getErrors	
port[8]	getHSBoost	
	setHSBoost	
	getName	
	setName	
	getPowerMode	
	setPowerMode	
	getDataRole	
port[9-10]	getHSBoost	
	setHSBoost	
	getName	
	setName	
	getDataRole	
port[11]	getName	
	setName	
	getDataRole	
USBSystem [0]	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
	getEnumerationDelay	
	setEnumerationDelay	

1.2 USBHub3c

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

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

1.2.1 Quick Start Guide

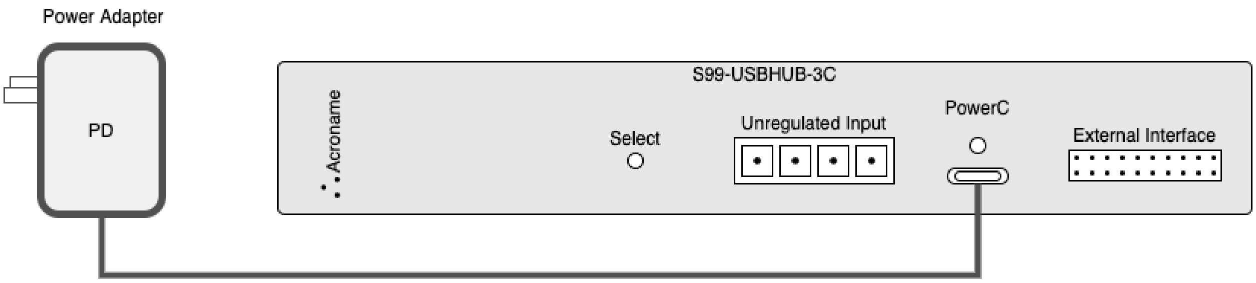
1. Download The Development Kit & HubTool

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

[3] https://acroname.com/api
[4] https://acroname.com/hubtool

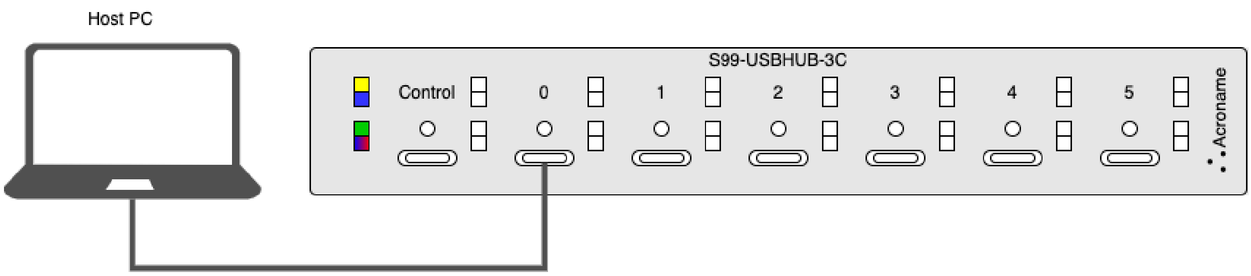
2. Connect Power

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



3. Connect Data

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



4. Play

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

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

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

1.2.2 Basic Example

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

C++

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

static const uint8_t PORT = 5;

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

    //Create an instance of the USBHub3c
    aUSBHub3c device;

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

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

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

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

    //Disconnect
    device.disconnect();

    return 0
}
```

Python

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

PORT = 5

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

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

(continues on next page)

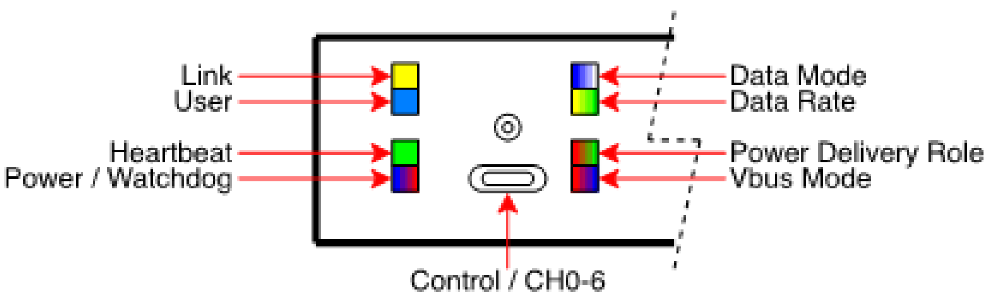
[5] https://acroname.com/api

(continued from previous page)

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

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

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

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

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

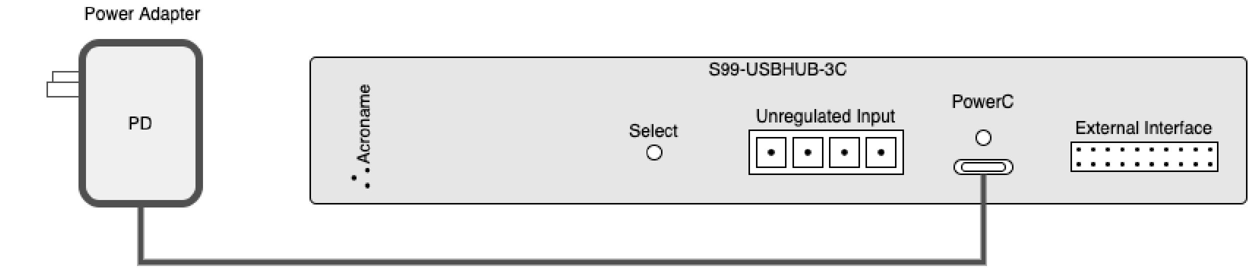
1.2.3 Indicators and Connections

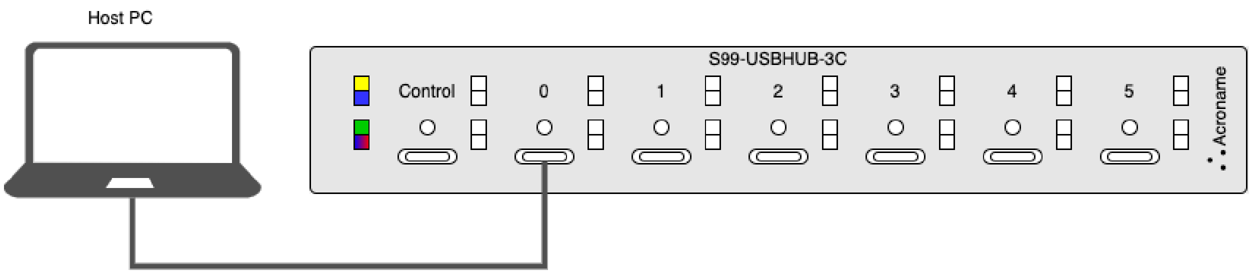
LEDs



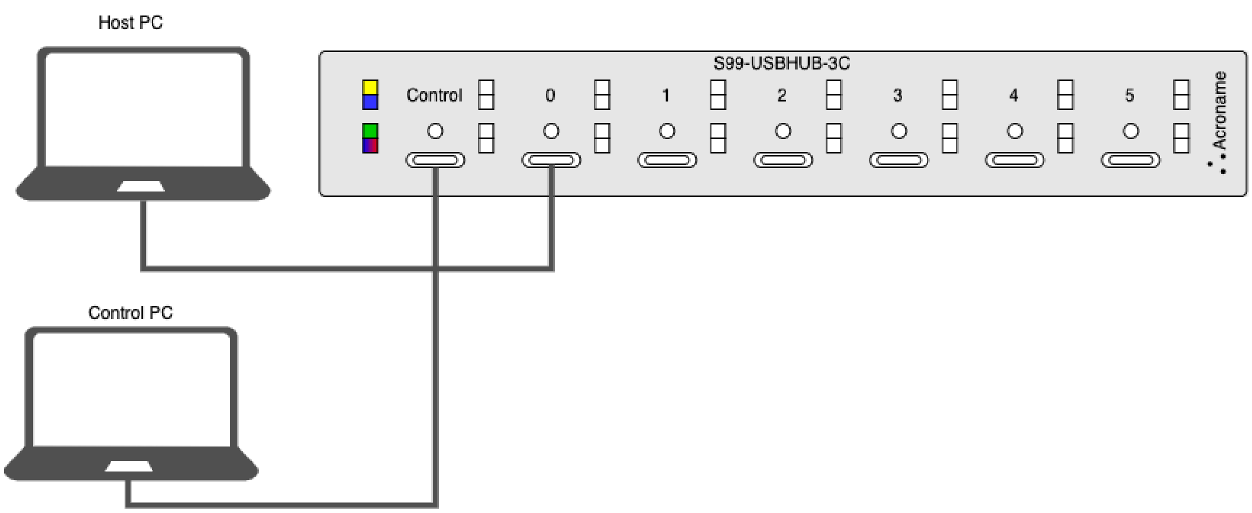
LED Name	Color	Description
Link Status LED	Yellow	On once a host device has enumerated the BrainStem controller
User LED	Blue	Can be manipulated through any of the available APIs
Heartbeat LED	Green	Indicates active BrainStem connection; pulses at a rate determined by the system heartbeat rate
Power/Watchdog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

Connections

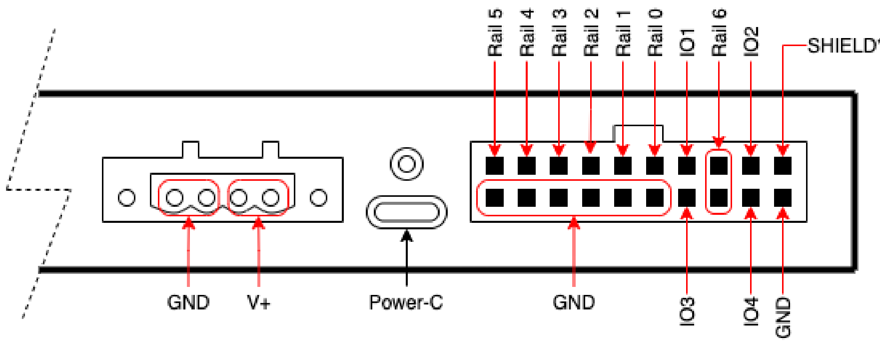




Separate Control Computer



External Connector



1.2.4 Programming Interface

Overview

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

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

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

Module class

C++

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

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

(continues on next page)

(continued from previous page)

```
        }

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

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

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

        system.init(this, 0);

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

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

        hub.init(this, 0);

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

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

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

};
```

Python

```
class USBHub3c(Module):

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

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

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


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

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

C++

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

Python

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

Supported Entites

See the *Module Entities* section of the this document for a complete list of the entities supported by the USB-Hub3c.

Ports

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

USBHub3c Hub

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

USBHub3c Power Delivery

There is a *Power Delivery* entity for each port on the USBHub3c. The *Power Delivery* entity controls much of the Power Delivery functionality of each of the Type-C ports.

USBHub3c System and Supporting entities

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

Connections

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

The USBHub3c is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the USBHub3c.

A complete list of all entities and functions can be found at the bottom of the *page*

Establishing Software Control

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

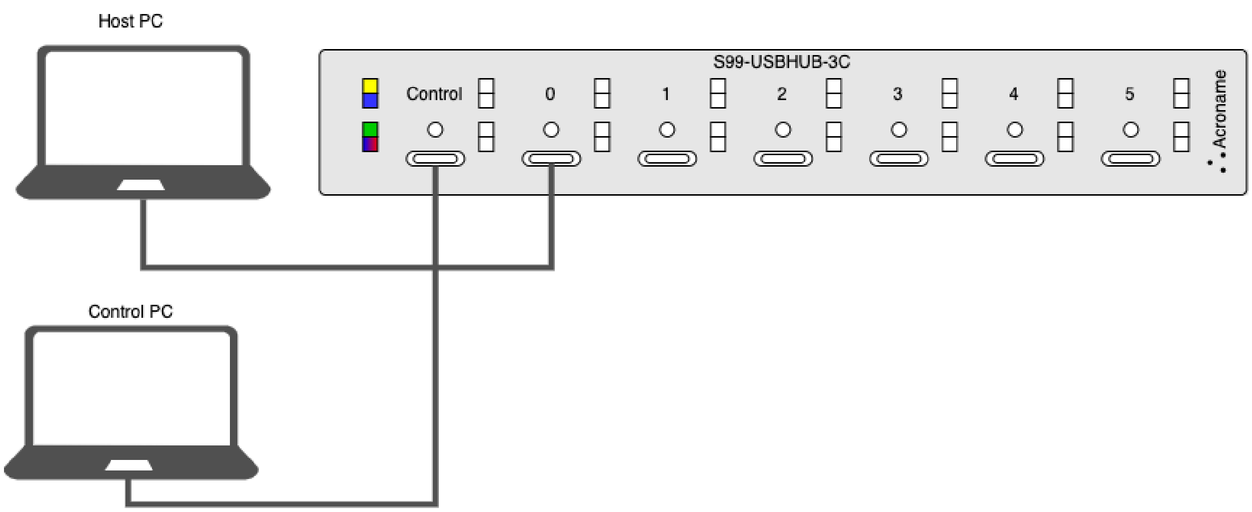
```
device.link.discoverAndConnect(USB)
```

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

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

BrainStem Control Port

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



Using Multiple Hosts with USBHub3c

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

Device Drivers

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

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

1.2.5 USBHub3c Module Entities

I2C

API Documentation: *[cpp] [python] [.NET] [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 granualar 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
USB-C	6	portPowerMode_usbc_Value

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

Port Mode

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

`stem.hub.port[x].setMode()` *[cpp] [python] [NET] [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	5	portPortMode_portPowerMode_ps_Value
Power Mode: USB-C	16-18	6	portPortMode_portPowerMode_usbc_Value

Data Role

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

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

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]*

Downstream Data Speed

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

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

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

Power Delivery

API Documentation: *[cpp] [python] [.NET] [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.pd[x].setPowerDataObject()` *[cpp] [python] [NET] [LabVIEW]*
`stem.pd[x].getPowerDataObject()` *[cpp] [python] [NET] [LabVIEW]*

`stem.pd[x].setRequestDataObject()` *[cpp] [python] [NET] [LabVIEW]*
`stem.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.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
Remote Source Extended Capabilities	10	pdRequestRemoteSourceExtendedCapabilities
Remote Sink Extended Capabilities	11	pdRequestRemoteSinkExtendedCapabilities
Status	12	pdRequestStatus
PPS Status	13	pdRequestPPSStatus
Battery Capabilities	14	pdRequestBatteryCapabilities
Battery Status	15	pdRequestBatteryStatus
Manufacturer Info Sop	16	pdRequestManufacturerInfoSop
Manufacturer Info Sopp	17	pdRequestManufacturerInfoSopp
Manufacturer Inof Soppp	18	pdRequestManufacturerInfoSoppp
Discover Identity Sop	19	pdRequestDiscoverIdentitySop
Discover Identity Sopp	20	pdRequestDiscoverIdentitySopp
Discover Identity Soppp	21	pdRequestDiscoverIdentitySoppp
Revision	22	pdRequestRevision
Source Info	23	pdRequestSourceInfo
Country Code	24	pdRequestCountryCode
Country Info	25	pdRequestCountryInfo

Errors return from this function call only indicate the success of sending the request and do not reflect the success of the actual request. To find the status of the request you can investigate the outcome of the connection or check the most recent status of the PD stack.

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

Cable Orientation

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

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

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

Cable Type

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

Override

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

`stem.pd[x].getOverride()` *[cpp] [python] [NET] [LabVIEW]*
`stem.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
Auto Discovery	2	Overrides the auto discovery feature. With override true the hub will only establish a basic power connection and wont ask for additional vendor information.

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].setEnable() [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-degress 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].setEnable() [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]
```

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 cleaver 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	usbsystemDataBehavior_HardCoded
Reserved	1	usbsystemDataBehavior_Reserved
Port Priority	2	usbsystemDataBehavior_PortPriority

Hard Coded (Default Configuration)

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

Port Priority

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

Relevant API's

`stem.hub.setDataRoleBehavior()` *[cpp] [python] [NET] [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 allication.

`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.getEnabledList()` *[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.setModeList()` *[cpp] [python] [NET] [LabVIEW]*

`stem.hub.getModeList()` *[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 functionally. The following functions of the *USB Entity* will still behave as they did in previous Acroname USB products.

Port

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

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

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

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

Power

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

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

Voltage

`stem.usb.getPortVoltage(channel, µV)` *[cpp] [python] [NET] [LabVIEW]*

Current

`stem.usb.getPortCurrent(channel, µA)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.getPortCurrentLimit(channel, µA)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.setPortCurrentLimit(channel, µA)` *[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	
	getModuleHardwareOffest	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	routeToMe	
	getUnregulatedVoltage	
	getUnregulatedCurrent	
	getPowerLimit	
	getPowerLimitMax	
	setPowerLimitMax	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
temperature[0-2]	getValue	
	getValueMax	
	getValueMin	

continues on next page

Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
port[0-7]	getVbusVoltage	
	getVbusCurrent	
	getVconnVoltage	
	getVconnCurrent	
	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	
	getName	
	setName	
	getCurrentLimit	
	setCurrentLimit	
	getAllocatedPower	
	getAvailablePower	
	getPowerLimit	

continues on next page

Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	setPowerLimit	
	getVbusAccumulatedPower	
	resetVbusAccumulatedPower	
	getVconnAccumulatedPower	
	resetVconnAccumulatedPower	
	getHSBoost	
	setHSBoost	
	getDataHSRoutingBehavior	
	setDataHSRoutingBehavior	
	getDataSSRoutingBehavior	
	setDataSSRoutingBehavior	
USBSystem [0]	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
PowerDelivery [0-8]	getConnectionState	
	getNumberOfPowerDataObjects	
	setPowerDataObject	
	getPowerDataObject	
	resetPowerDataObjectToDefault	
	getPowerDataObjectList	
	setPowerDataObjectEnabled	
	getPowerDataObjectEnabled	
	getPowerDataObjectEnabledList	
	setRequestDataObject	
	getRequestDataObject	
	getPowerRole	
	setPowerRole	
	getPowerRolePreferred	
	setPowerRolePreferred	
	getCableVoltageMax	
	getCableCurrentMax	
	getCableSpeedMax	
	getCableType	
	getCableOrientation	
	setOverrides	
	getOverrides	
	request	
	setCurrentLimitBehavior	
	getCurrentLimitBehavior	
	getPeakCurrentConfiguration	
	setPeakCurrentConfiguration	
	getFastRoleSwapCurrent	
	setFastRoleSwapCurrent	
	resetEntityToFactoryDefaults	
UART[0]	setEnable	
	getEnable	
	setBaudRate	
	getBaudRate	
	setProtocol	

continues on next page

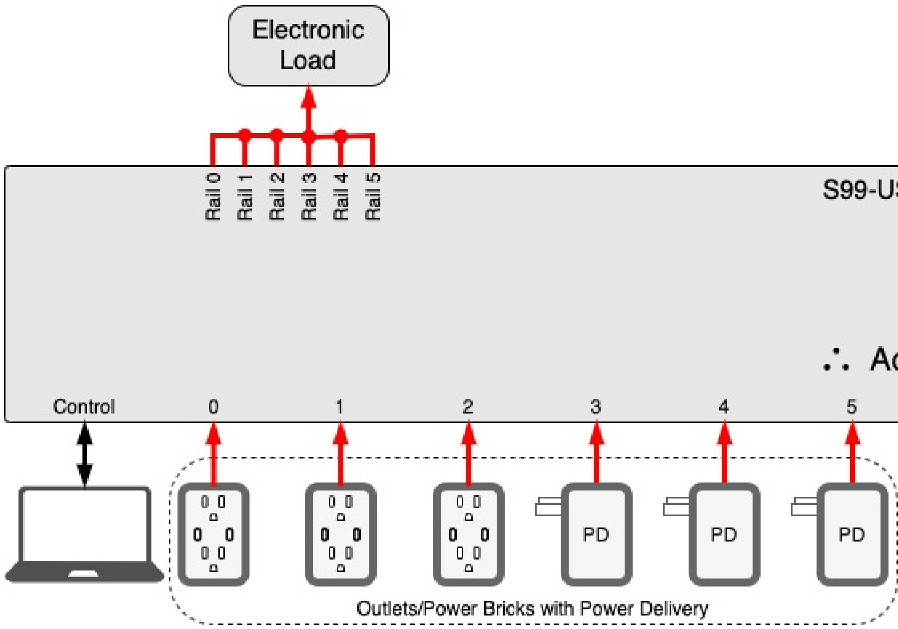
Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getProtocol	
rail[0-6]	setEnable	
	getEnable	
i2c[0]	read	
	write	
	setPullup	
	setSpeed	
	getSpeed	
usb[0]	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

1.2.6 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].setEnable(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].setEnable(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].setEnable(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].setEnable(false)

stem.disconnect()
```

Relevant API's

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

`stem.rail[x].getEnable()` *[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

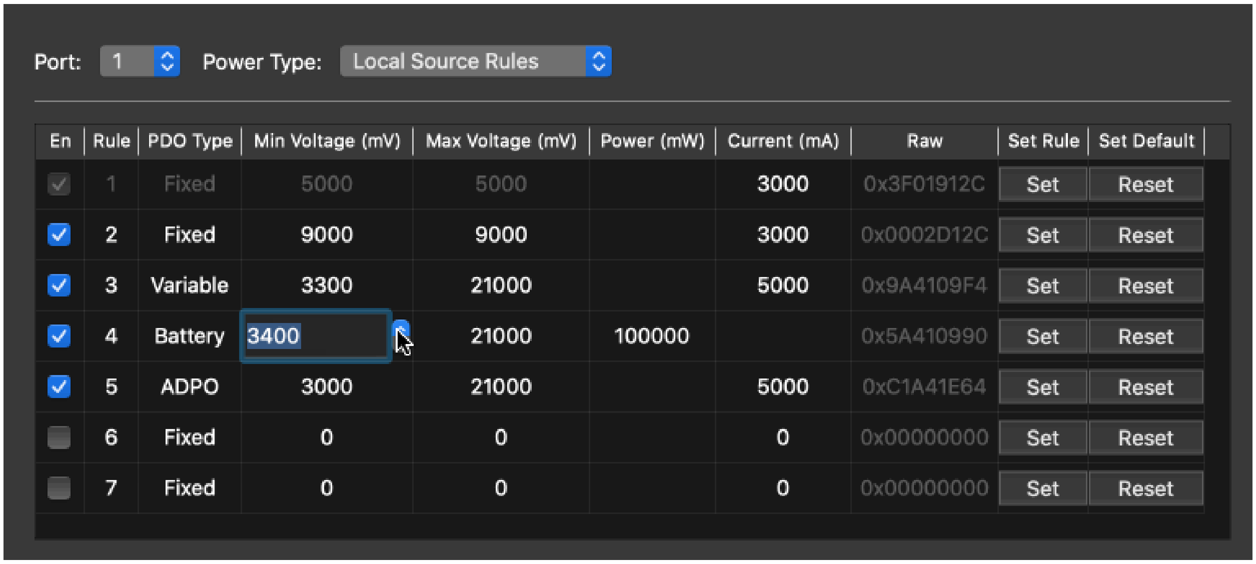
- Designing PDOs for you own device
- Exposing DUTs to PDOs
- Restricting DUTs from PDOs

Editable Items

PDO Types	PDO Flags	PDO Limits
Fixed	Unchuncked message support	Voltage
Variable	Dual role data	Current
Battery	USB communications possiable	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.



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)
```

(continues on next page)

(continued from previous page)

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

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

#Do Stuff

stem.disconnect()
```

Relevant API's

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

`stem.hub.pd[x].setPowerDataObjectEnabled()` *[cpp] [python] [NET] [LabVIEW]*
`stem.hub.pd[x].getPowerDataObjectEnabled()` *[cpp] [python] [NET] [LabVIEW]*

PD Logging

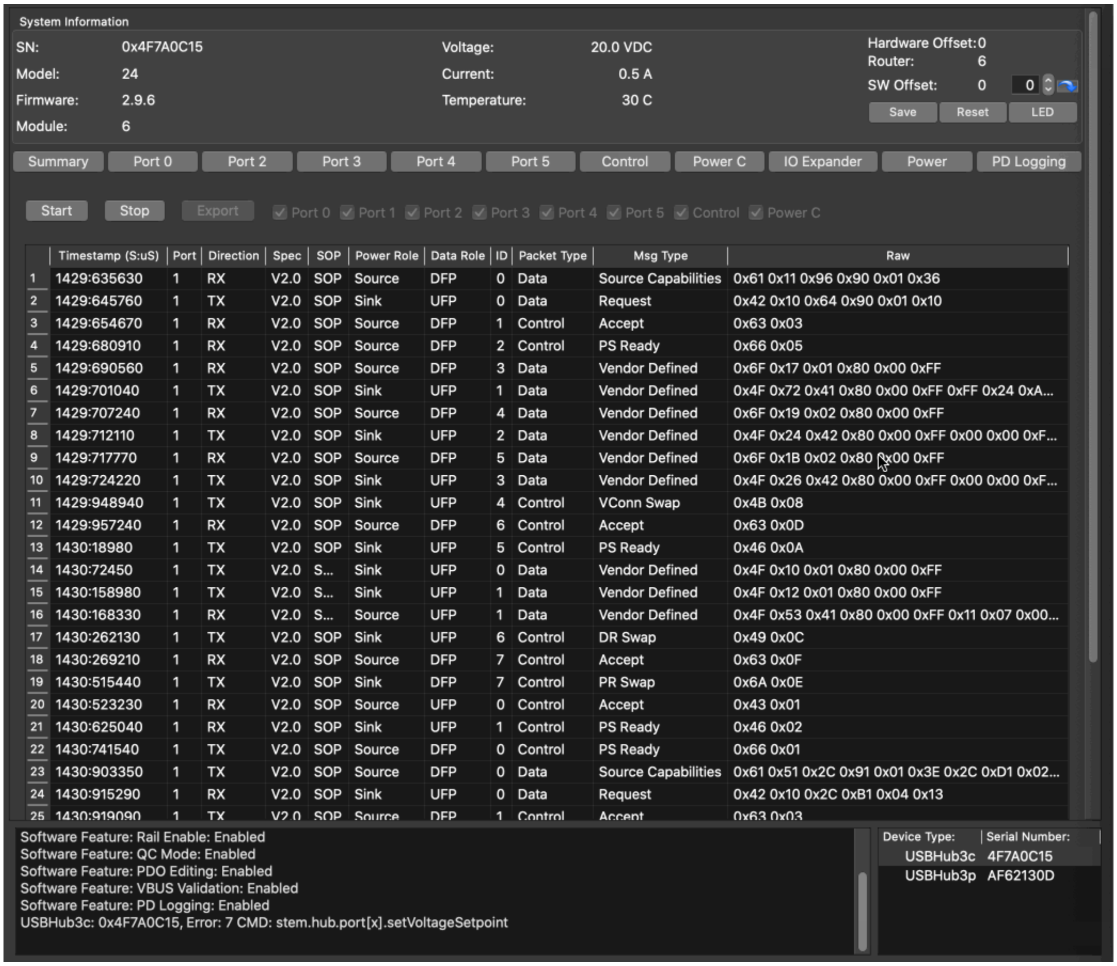
The PD Logging feature allows you to monitor Power Delivery communication on all 8 ports of the USBHub3c.

Use Cases

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

HubTool

With HubTool you can easily visualize the Power Delivery log.



Relevant API's

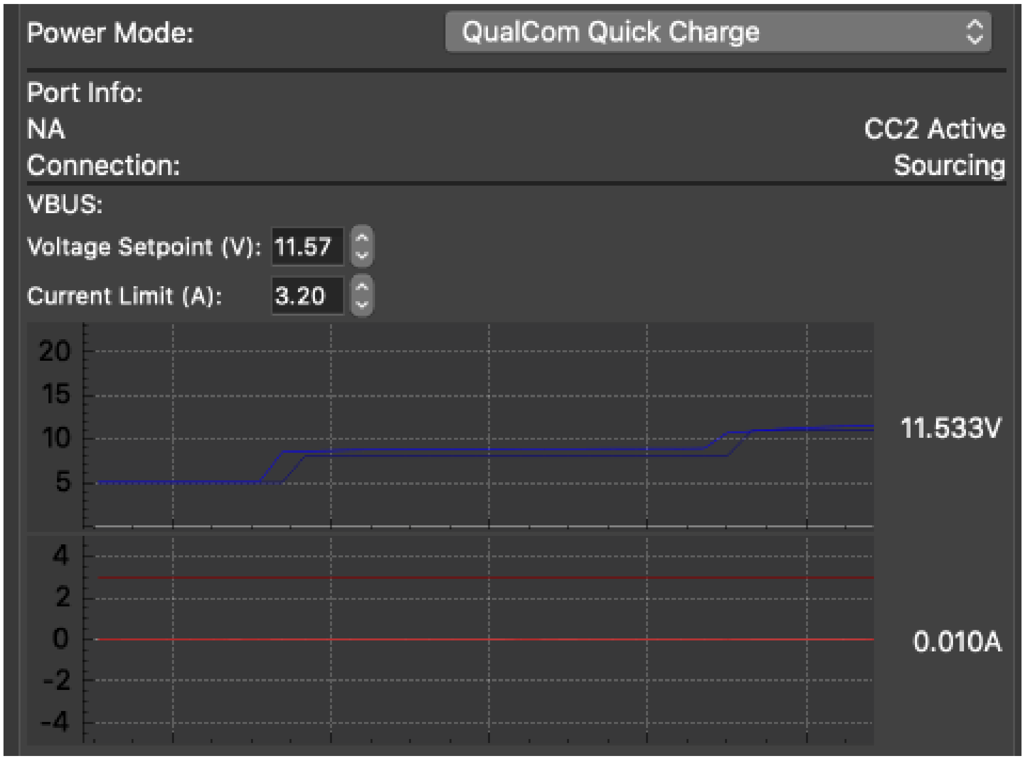
`stem.pd[x].setUEIBytes()` *[cpp]* [python]

Quick Charge™

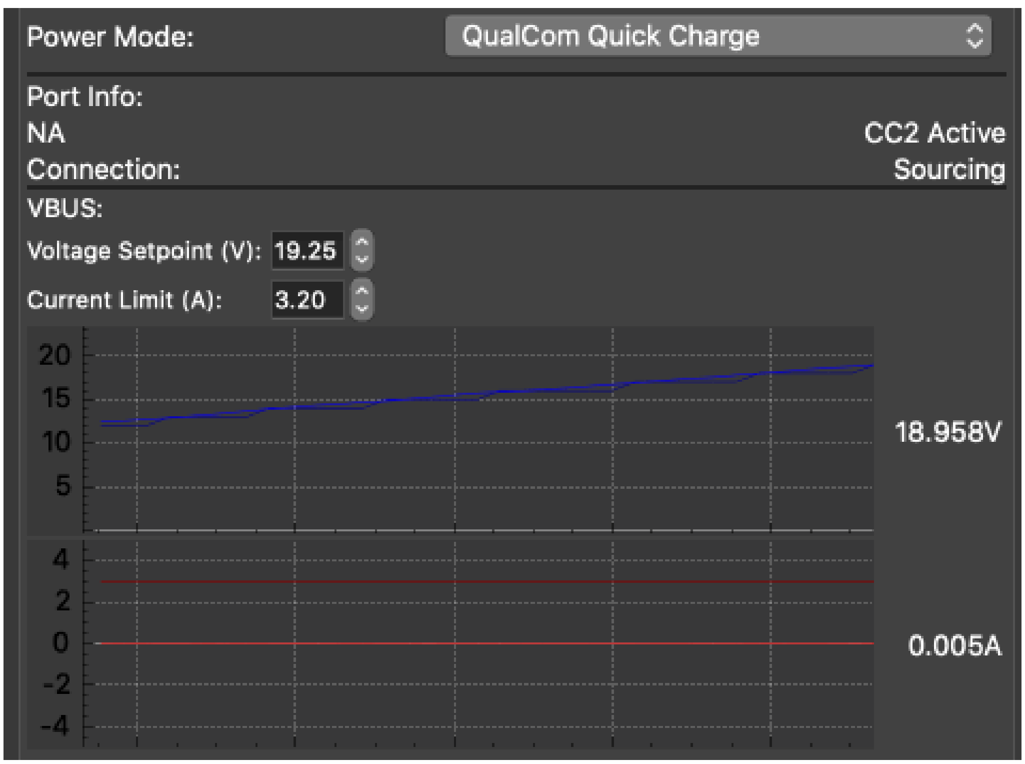
The USBHub3c Quick Charge™ feature adds the ability to enable the Quick Charge™ port power mode on one or more ports. This enables fast charging on devices that support Quick Charge™.

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

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

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.

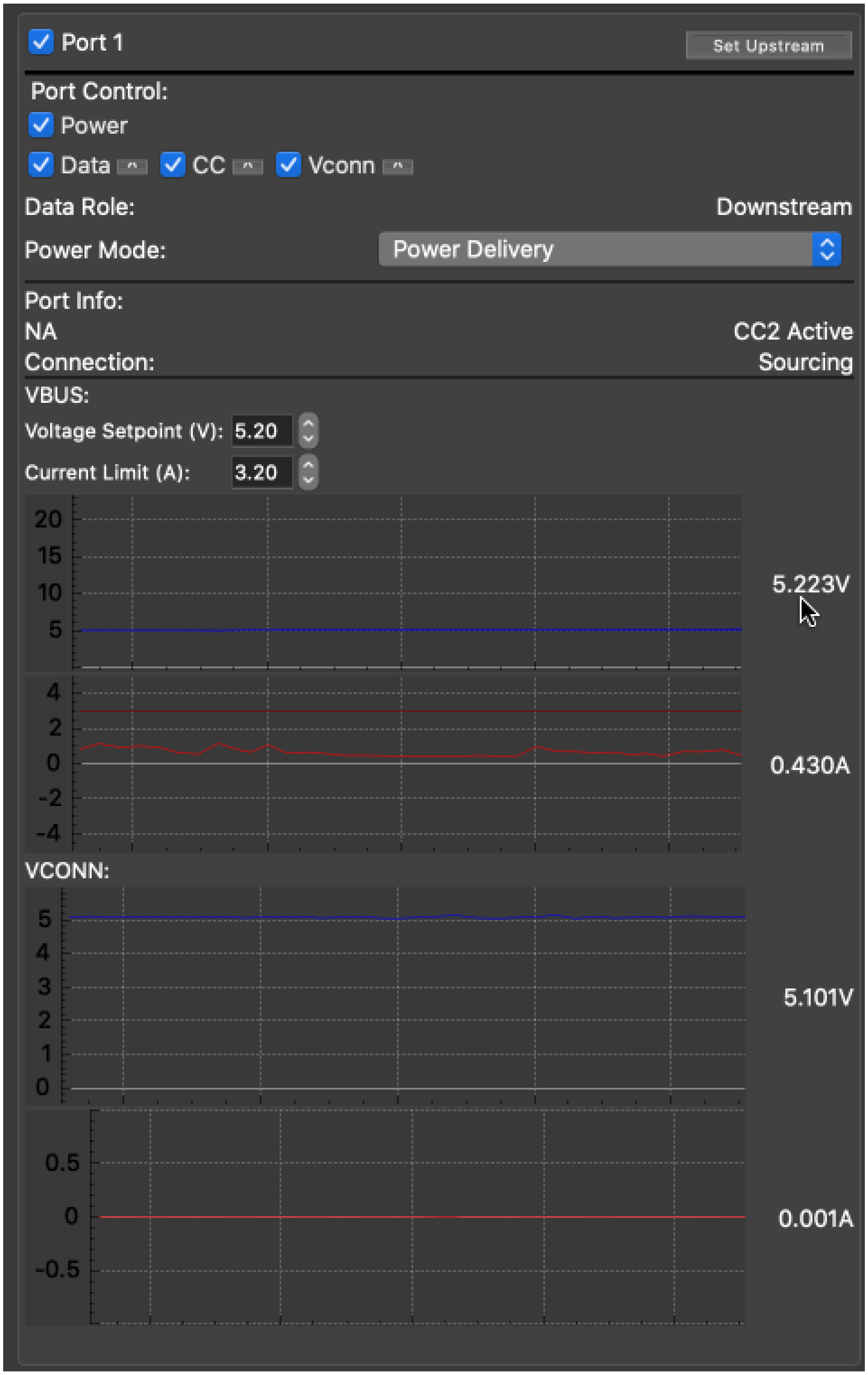
This software feature can be exploited through the *USBHub3c Port Entity*

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)

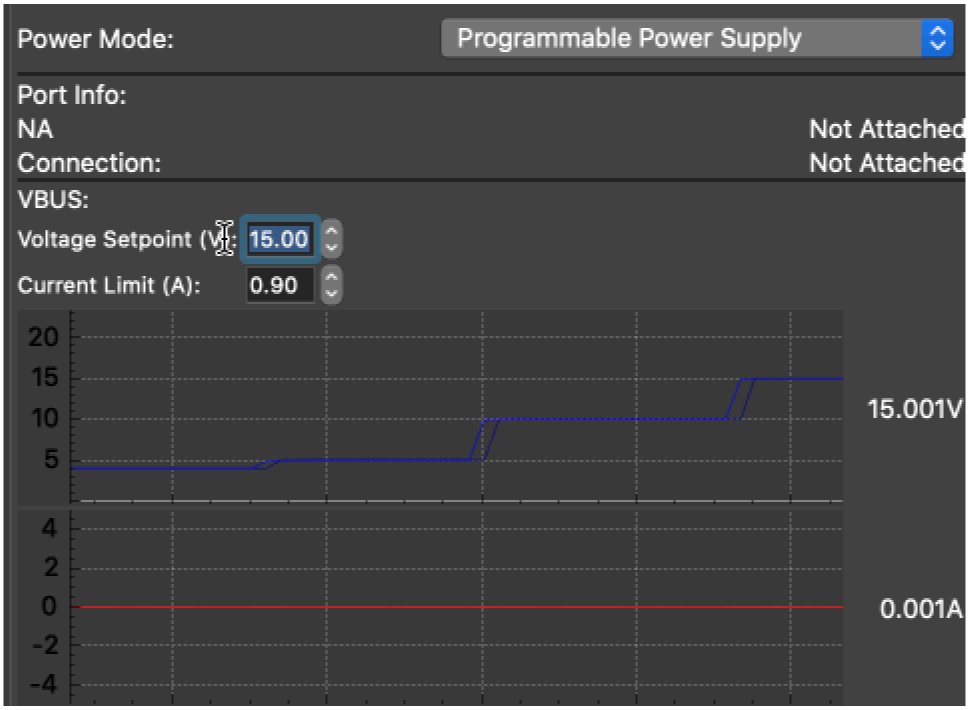
```
aUSBHub3c stem;
stem.discoverAndConnect(USB);

//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].setEnable(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].setEnable(true);

//Do Stuff

//Return port to safe state.
stem.hub.port[TEST_PORT].setEnable(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].setEnable(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].setEnable(true)

#Do Stuff

#Return port to safe state.
stem.hub.port[TEST_PORT].setEnable(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].setEnable(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].setEnable(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].setEnable()` *[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]*

1.3 USBHub2x4

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

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

1.3.1 Quick Start Guide

1. Download The Development Kit

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

[6] https://acroname.com/api
[7] https://acroname.com/hubtool

2. Connect Power

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

3. Connect Data

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

4. Run System

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

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

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

1.3.2 Basic Example

C++

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

    //Disconnect
    hub.disconnect();
}
```

Python

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

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

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

############
```

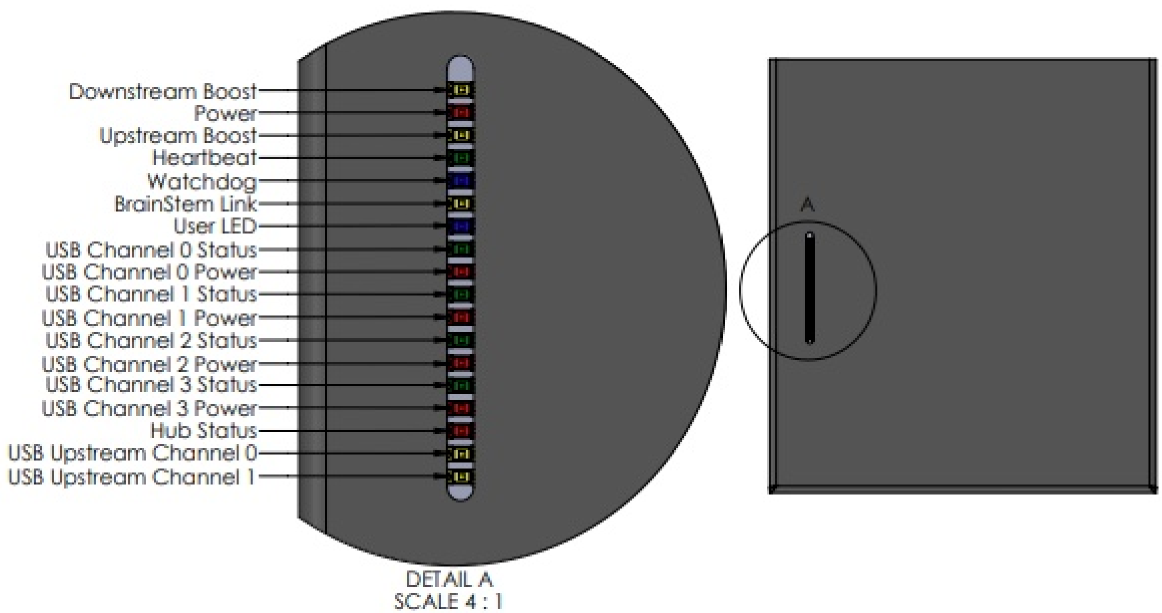
(continues on next page)

(continued from previous page)

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

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

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

1.3.3 Indicators and Connections

LEDs and Connections



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

1.3.4 Programming Interface

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

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

Software Control

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

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

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

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

Using Multiple Hosts with USBHub2x4

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

Device Drivers

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

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

1.3.5 USBHub2x4 Module Entities

System

API Documentation: *[cpp] [python] [.NET] [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-degress 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 folloing methods provide equivalent functionality; the two methods are offered for compatability 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 apprears 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, µV)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.getPortCurrent(channel, µA)` *[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, µA)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.setPortCurrentLimit(channel, µA)` *[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.getEnumberationDelay(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

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

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

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

`stem.hub.port[x].setEnabled()` *[cpp] [python] [NET] [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 0-3.

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

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

`stem.hub.port[x].setDataEnabled()` *[cpp] [python] [NET]* [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 0-3.

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

Voltage and Current Measurements

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

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

Power Modes

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

`stem.hub.port[x].setPowerMode()` *[cpp] [python] [NET] [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

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

Port Mode

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

`stem.hub.port[x].setMode()` *[cpp] [python] [NET] [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
Power Mode: Offset		16	portPortMode_portPowerMode_Offset
Power Mode: Mask		0x7	portPortMode_portPowerMode_Mask
Power Mode: None	16-18	0	portPortMode_portPowerMode_none_Value
Power Mode: SDP	16-18	1	portPortMode_portPowerMode_sdp_Value
Power Mode: CDP/DCP	16-18	2	portPortMode_cdp_dcp_Value

Data Role

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

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

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

Downstream Data Speed

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

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

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

USB System

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

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

Upstream Control

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

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

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

Enumeration Delay

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

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

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

Data Behavior

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

List of Available Data Behaviors for USBHub2x4

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

Hard Coded (Default Configuration)

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

Port Priority

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

Relevant API's

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

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

Complete list of Supported Entities and Functions

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

continues on next page

Table 4 – continued from previous page

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

continues on next page

Table 4 – continued from previous page

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

1.4 USB-C-Switch

1.4.1 Quick Start Guide

1. Download The Development Kit

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

2. Connect Device(s)

- Using the Acroname Unversal 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

- Open HubTool
- On the bottom right side of the application select the USB-C-Switch device.

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

Congratulations! You are now ready to start exploring the capabilities of the USB-C-Switch. For more information please take a look at our *Getting Started Guide*

1.4.2 Basic Example

C++

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

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

    // Connect to the hardware.
    aUSBCSwitch cswitch;
    auto err = cswitch.discoverAndConnect(USB);
    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);
```

(continues on next page)

[8] https://acroname.com/api
[9] https://acroname.com/hubtool

(continued from previous page)

```
    cswitch.mux.setEnable(false);
    cswitch.mux.setChannel(0);

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

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

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

    cswitch.mux.setChannel(1);

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

    cswitch.mux.setChannel(2);

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

    cswitch.mux.setChannel(3);

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

    //Finished with testing.
    //De-initialize.
    cswitch.usb.setPortDisable(0);
    cswitch.mux.setEnable(false);

    //Disconnect
    cswitch.disconnect();
}
```

Python

```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 USBCSwitch module.
cswitch = brainstem.stem.USBCSwitch()

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

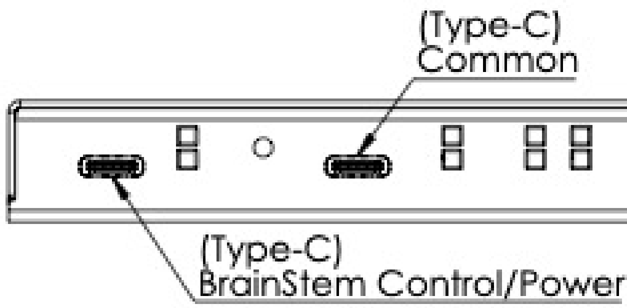
(continues on next page)

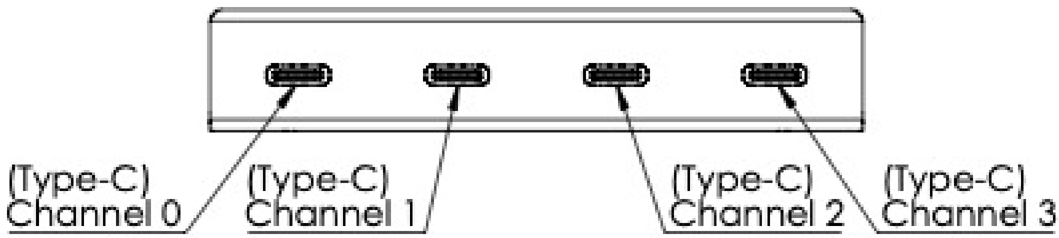
(continued from previous page)

```
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 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)


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

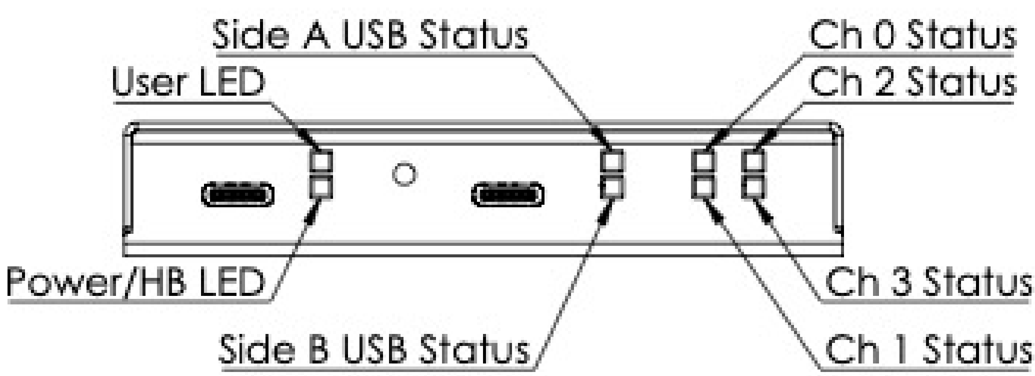
1.4.3 Indicators and Connections

USB Channels





LED Indicators



LED Name	Color	Description
User	Blue	Can be manipulated through the available APIs
Power/ Heartbeat	Red/Green	Red indicates system is powered. Flashing green is the heartbeat which indicates an active software connection. Pulses at a rate determined by the system heartbeat rate to indicate an active BrainStem link.
Side A USB Status	Green/Yellow/Grey	Un-flipped/Flipped/CC1 Disabled
Side B USB Status	Green/Yellow/Grey	Un-flipped/Flipped/CC2 Disabled
Channel 0 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.
Channel 1 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.
Channel 2 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.
Channel 3 Status	Blue	Indicates Mux Channel selection. Disabled when Split mode is enabled.

1.4.4 Programming Interface

Generally, the Passive model is best for emulating off-the-shelf cables and for eye-diagram validation.

The Redriver model is optimal for general connectiviity or longer connections. It includes a programmable, linear, equalizing redriver which allows USB signal tuning to compensate for insertion and cabling losses.

The USB-C-Switch contains many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the USB-C-Switch.

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

Software Control

Software control of the features of the USB-C-Switch is done with the BrainStem API via a BrainStem link. BrainStem links are done over USB and can be established via the Control Port. After this port is connected to a host machine, a user can connect to it via software API:

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

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

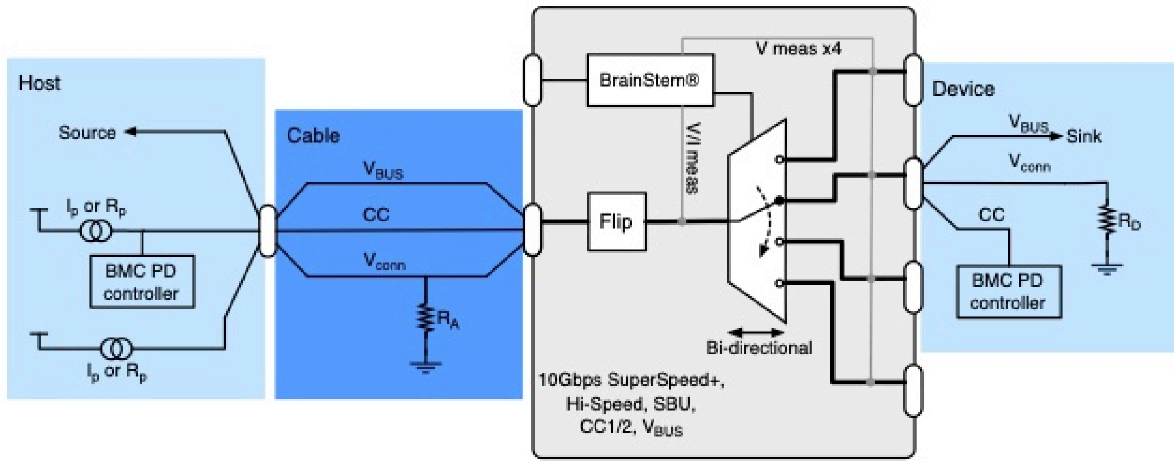
```
brainstem.discover.findAllModules(USB) (Python)
Acroname::BrainStem::Link::sDiscover() (C++)
```

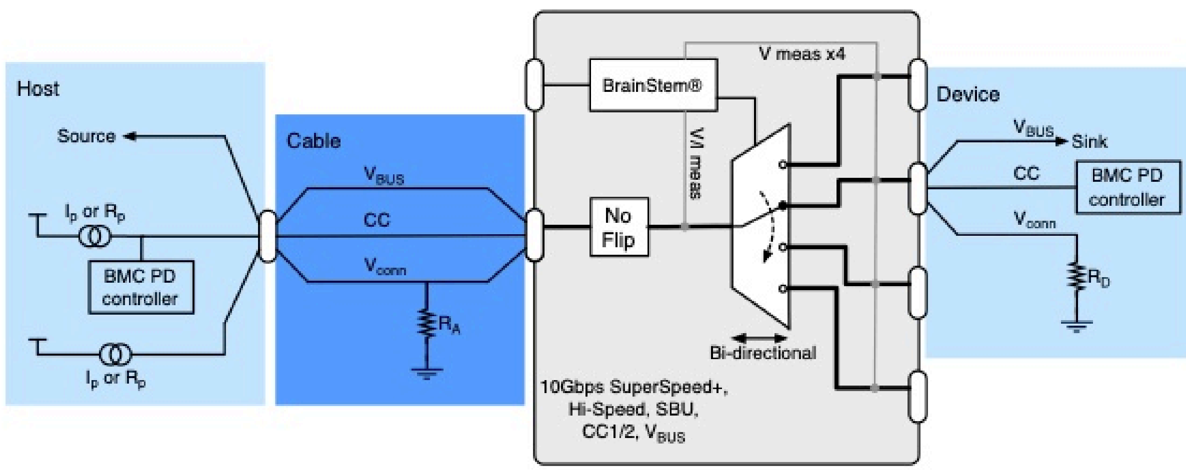
Cable Flip

A key feature of the USB-C connector is its symmetric design allowing for insertion in either orientation. This makes the USB-C connector user-friendly yet complicates the development of devices using the USB-C standard. The orientation is defined by the cable or downstream device in the system; more specifically, by components inside of the USB-C male plug of a connection. The USB-C specification makes determining connector orientation a responsibility of the active devices in the system.

With an Acroname UOC cable, the USB-C-Switch enables the unique ability to affect a cable orientation flip. When this orientation flip occurs, it will appear to connected devices that the orientation of their connection has reversed. Most USB-C devices with a female socket will include at least one set of muxes in order to route signal to the correct side of the socket based on the orientation of the cable. When testing such a system it is import to test both orientations to ensure that these internal muxes are functioning. The USB-C-Switch allows flipping of USB-C cable connections to be programmatically automated.

Depicted below are the flip and no-flip setting for full-featured cable and device





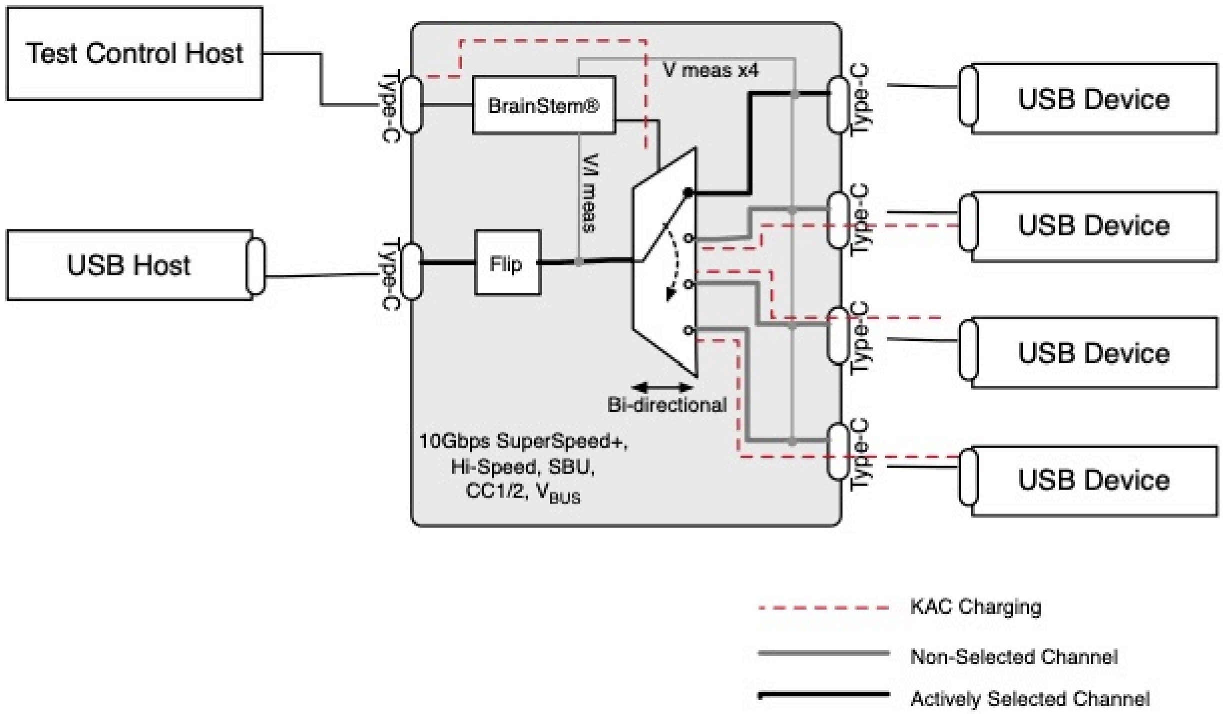
A UOC should be used on either the common port or mux port to enable automated cable flips. The UOC should be connected to the device under test.

When not using the cable flip feature, any standard USB-C cable can be used on both sides of the USB-C-Switch. The orientation of the cables need to be matched in order to facilitate a connection through the switch.

Keep-Alive Charging (KAC)

It is common to use battery powered devices on either side of the USB-C-Switch. When these devices are not in the active path, either on the common or mux side, the device battery may discharge. The USB-C-Switch has the unique feature of Keep-Alive Charging (KAC) for the mux channel connections.

Below is a diagram for the USB-C-Switch KAC capability:



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

Mux Modes

Default Mode

The default behavior of the USB-C-Switch is to act as a port selector, where all USB-C lines are connected between the common port and one selected mux channel.

Channel Priority Mode

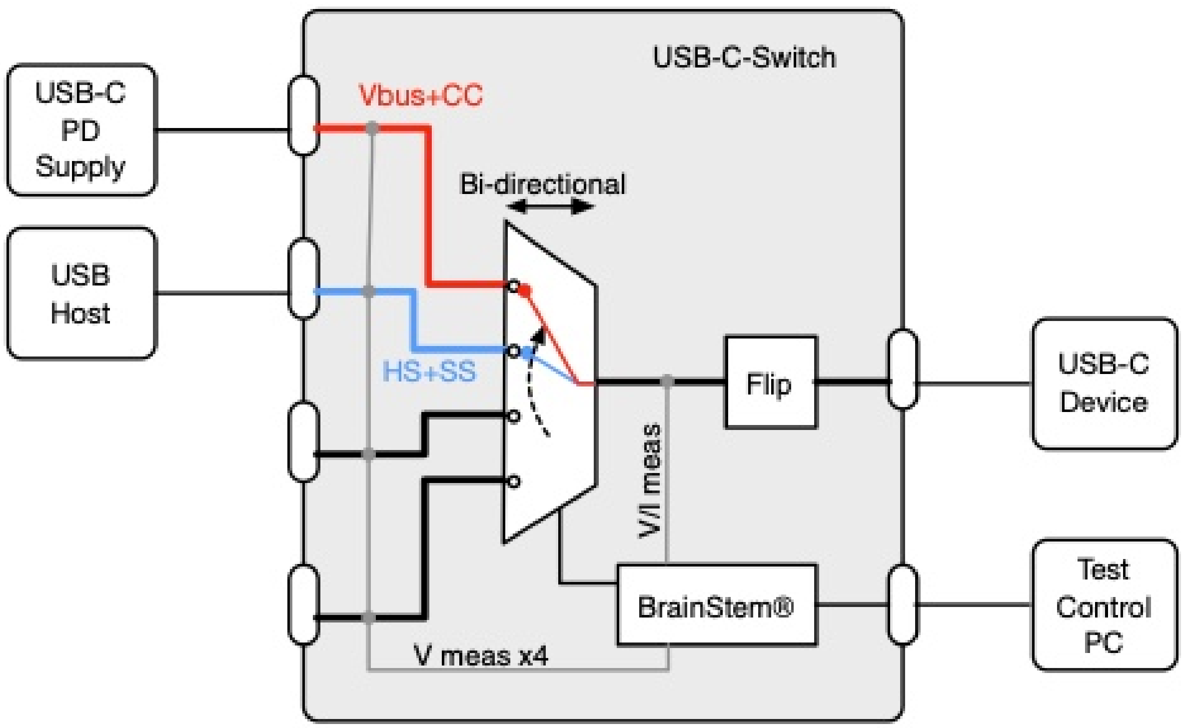
In channel priority mode, the USB-C-Switch automatically selects the lowest-numbered mux port where VBUS is detected, enabling simple automatic host selection. Note that this will only work with USB A-to-C cables, since VBUS is not immediately available on USB-C to USB-C connections.

Split Mode

In some cases, it is desirable to split the connections in a USB-C cable and route them to different mux paths. A common application is to be able connect a USB device to a host machine for USB data while connecting VBUS charging from a device specific charger.

Split mode gives control over individual signal groups, allowing each group to be connect to a mux channel. VBUS can be connected to any combination of mux channels or disabled on the mux channels. Signal groups under Split control assignment are: VBUS, SSA (TX1+/-, RX1+/-), SSB (TX2+/-, RX2+/-), HSA (D+/-, Side A), HSB (D+/-, Side B), CC1, CC2, SBU1, and SBU2.

A basic example of the USB-C-Switch Mux split mode is depicted.



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

Device Drivers

The USB-C-Switch leverages operating system user space interfaces that do not require custom drivers for operation on all modern operating systems including Windows, Linux and MacOS X. With a connection between a host PC and the USB-C control port, the host PC will recognize a USB full-speed device named "USBCSwitch".

Legacy operating systems like Windows 7 may require the installation of a BrainStem USB driver. Installation details on installing USB drivers can be found within the BrainStem Development Kit under the "drivers" folder.

1.4.5 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 progammatic 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 intergrity loss due to cable quality, length, and insertoion 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) paramters are defined below:

Value	Receiver Channel
0	Applies setting to both common and mux sides
1	Applies settings to mux side
2	Applies settings to common side

High Speed Redriver Configuration

Due to the half-duplex nature of the USB2 data lines, there is only one receiver and transmitter setting for both the common and mux ports. In addition, since the transmitter and receiver are tightly coupled, the linear gain achieved by transmitter setting varies with the equalizer receiver configuration. Approximate gains for example configurations are shown in the specifications table.

`stem.equalizer[0].setTransmitterConfig(config)` *[cpp] [python] [NET] [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 equalier receiver option can only be 0 because the HS data lines are half-duplex. All other values will result in an error return.

The HS equalizer receiver option configurations control the sensitivity of the redriver to incoming HS signals. The effect of this change in sensitivity can be considered a variable AC boost turned to the specific HS signal applied. Setting the HS equalizer to Level 0 will disable the HS redriver regardless of the HS entity transmitter configuration. The available options are shown below:

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

`stem.equalizer[0].getReceiverConfig(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.

`stem.equalizer[1].setTransmitterConfig(config)` *[cpp] [python] [NET] [LabVIEW]*

`stem.equalizer[1].getTransmitterConfig(config)` *[cpp] [python] [NET] [LabVIEW]*

Value	Mux Side	Com Side	Range
0	1db	0db	900mVpp
1	0db	1db	900mVpp
2	1db	1db	900mVpp
3	0db	0db	900mVpp
4	0db	0db	1100mVpp
5	1db	0db	1100mVpp
6	0db	1db	1100mVpp
7	2db	2db	1100mVpp
8	0db	0db	1300mVpp

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

`stem.equalizer[1].setReceiverConfig(0,config)` *[cpp] [python] [NET] [LabVIEW]*

`stem.equalizer[1].getReceiverConfig(0,config)` *[cpp] [python] [NET] [LabVIEW]*

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 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 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 swith can split the USB-C functional groupd and route them to selected mux ports. This feature is referred to as "split mode". The switch can also auto select the lowest Mux channel that currently has VBUS present which is called "port priority". Default, split mode, or port priority can be enabled with:

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

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

where (config) is 0 for default, 1 for Split Mode, and 2 for Port Priority Mode.

Split Mode

After enabling split mode the USB-C functional groups can be individually assigned to separate mux channels with:

`stem.mux.getSplitMode(splitMode)` *[cpp] [python] [NET] [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 funtional signal group. Vbus and CC use 4-bits to define which mux ports are connected to the common port Vbus/CC lines.

Bit	Mux Split Mode Bit Map
0:1	SBU1
2:3	SBU2
4	CC1 enable CH0
5	CC1 enable CH1
6	CC1 enable CH2
7	CC1 enable CH3
8	CC2 enable CH0
9	CC2 enable CH1
10	CC2 enable CH2
11	CC2 enable CH3
12:13	HS Side A Data
14:15	HS Side B Data
16:17	SS Lane 1 Data
18:19	SS Lane 2 Data
20	Vbus enable CH0
21	Vbus enable CH1
22	Vbus enable CH2
23	Vbus enable CH3
24:31	Reserved

Note: The split mode interface changed in 2.10.0, ensure your firmware and software are up to date.

System

API Documentation: *[cpp] [python] [.NET] [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 functionality 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, µV)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.setCC2Voltage(0, µV) [cpp] [python] [NET] [LabVIEW]`

`stem.usb.getCC1Current(0, µA)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.getCC2Current(0, µA)` *[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, μV)` *[cpp] [python] [NET] [LabVIEW]*

`stem.usb.getPortCurrent(0, μA)` *[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
App[0-3]	execute	
	return	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	
	getTransferStore	
	setTransferStore	
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	
	setInt	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	getModel	
	getHardwareVersion	
	getModule	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getVersion	
	getSerialNumber	
	save	
	reset	
	getInputVoltage	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	getUptime	
	getName	
	setName	
	resetDeviceToFactoryDefaults	
timer[0-8]	getExpiration	
	setExpiration	
usb[0]	setPortEnable	

continues on next page

Table 5 – continued from previous page

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

1.5 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.

1.5.1 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)[10] for your particular operating system and architecture.
- Download HubTool[11] for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM Relay by either connecting to the:
  - Onboard USB connection
  - Card edge USB input
  - Through other MTM modules on the local BrainStem bus.

3. Run System

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

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

Congratulations! You are now ready to start exploring the capabilities of the MTM Relay. For more information please take a look at our *Getting Started Guide*

Basic Example

C++

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

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

    //Create an instance of a MTM-RELAY module.
    aMTMRelay mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
```

(continues on next page)

[10] https://acroname.com/api
[11] https://acroname.com/hubtool

(continued from previous page)

```
        return 1;

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

    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

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

Python

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

# Create an instance of a MTM-RELAY module.
mtm = brainstem.stem.MTMRelay();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

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

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

Indicators and Connections

LEDs

The MTM Relay board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below.

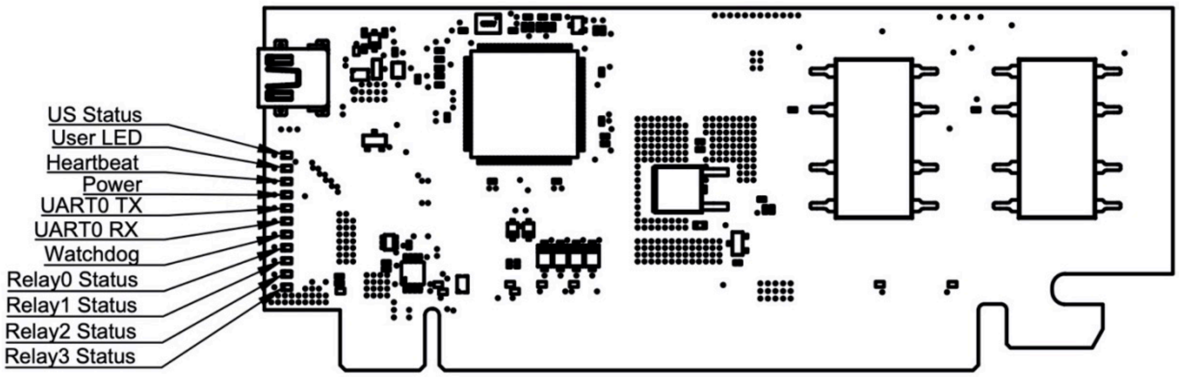


Figure 3: MTM-Relay LED Indicators

Programming Interface

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

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

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-Relay. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-Relay can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStemprotocol (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].setEnable()` *[cpp]* [python] *[NET]* *[LabVIEW]*

`stem.relay[index].getEnable()` *[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

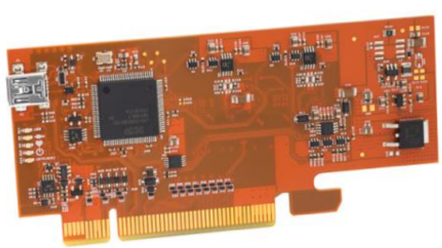
Entity Class	Entity Option	Variable(s) Notes
digital[0-3]	setConfiguration	
	getConfiguration	
	setState	
	getState	
i2c[0]	write	
	read	
relay[0-3]	setEnable	
	getEnable	

continues on next page

Table 6 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getVoltage	
store[0-1]	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	

1.5.2 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)[12] for your particular operating system and architecture.
- Download HubTool[13] for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM DAQ-2 by either connecting to the:
  - Onboard USB connection
  - Card edge USB input
  - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM DAQ-2 device.

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

Congratulations! You are now ready to start exploring the capabilities of the MTM Relay. For more information please take a look at our *Getting Started Guide*

Basic Example

C++

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

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

    //Create an instance of a MTM-DAQ-2 module.
    aMTMDAQ2 mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
```

(continues on next page)

[12] https://acroname.com/api
[13] https://acroname.com/hubtool

(continued from previous page)

```
        printf("Error %d encountered connecting to BrainStem module\n", err);
       return 1;

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

    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

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

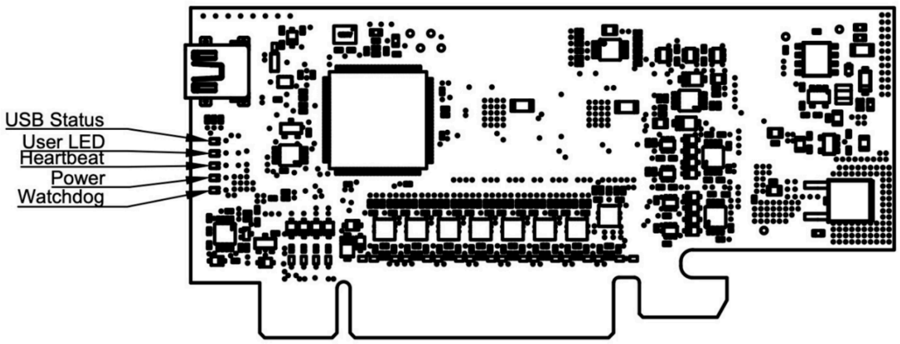
Python

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

# Create an instance of a MTM-DAQ-2 module.
mtm = brainstem.stem.MTMDAQ2();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

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

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

Indicators and Connections

LEDs

The MTM-DAQ-2 board has five LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below



Programming Interface

The MTM-DAQ-2 is capable of many features. These features are organized into groups called *entities*. Through these entities we can access the vast features of the MTM-DAQ-2.

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

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-DAQ-2. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-DAQ-2 can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStemprotocol (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 Index	Storage Type	Number of Slots
0	Internal	12
1	RAM	1

Get Slot State

For slots which hold reflexes, this read only command returns whether the slot is currently enabled or not. 1 is enabled 0 is disabled. This command can be called from a reflex.

`stem.store[index].getSlotState(slot)` *[cpp] [python] [NET] [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 setEnable(1) must be called before their voltage will be present on their respective pins.

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

`stem.analog[index].setEnable(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 =: 0x0FFF << 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 applicapble 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	
	setEnable	
	getEnable	
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	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	

1.5.3 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)[14] for your particular operating system and architecture.
- Download HubTool[15] for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM Power Module by either connecting to the:
  - Onboard USB connection
  - Card edge USB input
  - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM Power Module device.

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

Congratulations! You are now ready to start exploring the capabilities of the MTM Relay. For more information please take a look at our *Getting Started Guide*

Basic Example

C++

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

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

    //Create an instance of a MTM-PM-1 module.
    aMTMPM1 mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
       return 1;

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

```

(continues on next page)

[14] https://acroname.com/api
[15] https://acroname.com/hubtool

(continued from previous page)

```
    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

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

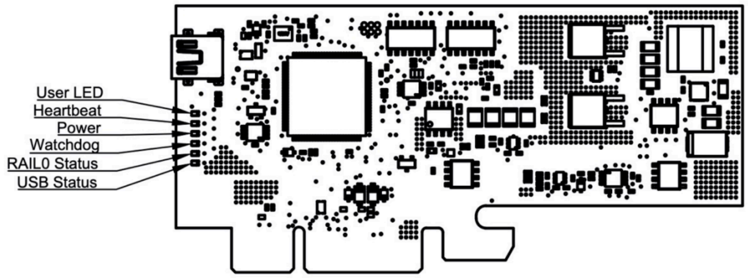
Python

```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 BrainStemprotocol (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].setEnable(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" (railOperationStateOverCurrentFault)	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-degress 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]	setEnable	
	setCurrentLimit	
	getCurrent	
	getCurrentLimit	
	getVoltage	
rail[0]	setVoltage	
	setOperationalMode	
	getOperationalMode	
	getOperationalState	
	getTemperature	
	getKelvinSensingEnable	
	setKelvinSensingEnable	
	getKelvinSensingState	
store[0-1]	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		

1.5.4 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 it's 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)[16] for your particular operating system and architecture.
- Download HubTool[17] for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM Load by either connecting to the:
  - Onboard USB connection
  - Card edge USB input
  - Through other MTM modules on the local BrainStem bus.

3. Run System

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

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

Congratulations! You are now ready to start exploring the capabilities of the MTM Load. For more information please take a look at our *Getting Started Guide*

Basic Example

C++

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

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

    //Create an instance of a MTM-LOAD module.
    aMTMLoad mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
       return 1;

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

(continues on next page)

16 https://acroname.com/api
17 https://acroname.com/hubtool

(continued from previous page)

```
    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

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

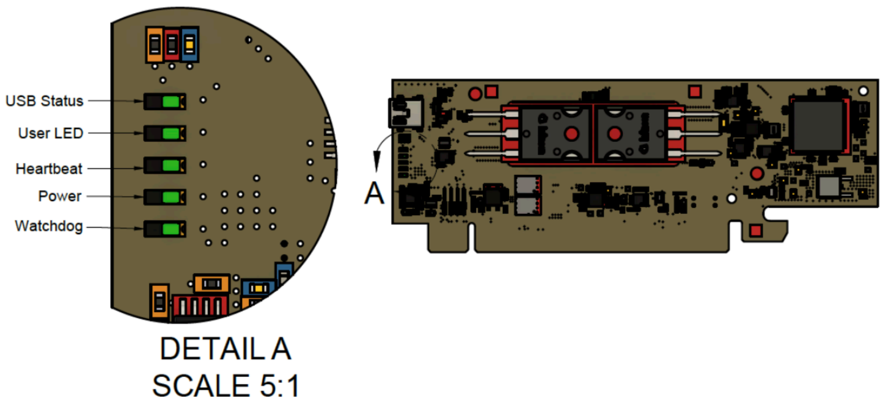
Python

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

# Create an instance of a MTM-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 BrainStemprotocol (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].setEnable(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

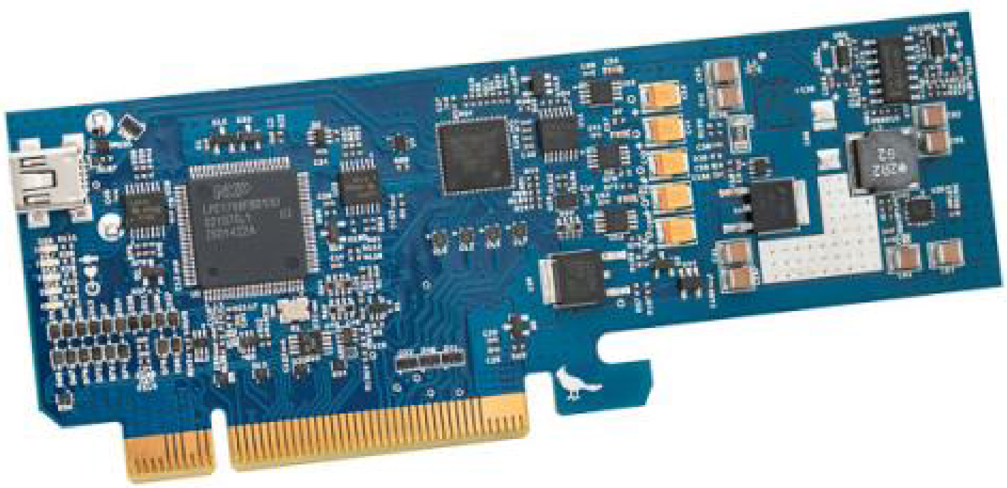
Entity Class	Entity Option	Variable(s) Notes
digital[0-3]	setConfiguration	
	getConfiguration	
	setState	
	getState	
i2c[0]	write	
	read	
	setPullup	Disabled by default.
rail[0]	setCurrent	
	getCurrent	
	getCurrentSetpoint	
	setCurrentLimit	
	getCurrentLimit	
	getTemperature	
	setEnable	
	getEnable	
	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	
	slotSize	
system[0]	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	

1.5.5 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)[18] for your particular operating system and architecture.
- Download HubTool[19] for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM IO Serial by either connecting to the:
  - Onboard USB connection
  - Card edge USB input
  - Through other MTM modules on the local BrainStem bus.

3. Run System

- Open HubTool
- On the bottom right side of the application select the MTM IO Serial device.

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

Congratulations! You are now ready to start exploring the capabilities of the MTM IO Serial. For more information please take a look at our *Getting Started Guide*

Basic Example

C++

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

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

    //Create an instance of a MTM-IO-SERIAL module.
    aMTMIOSerial mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
       return 1;

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

(continues on next page)

18 https://acroname.com/api
19 https://acroname.com/hubtool

(continued from previous page)

```
    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

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

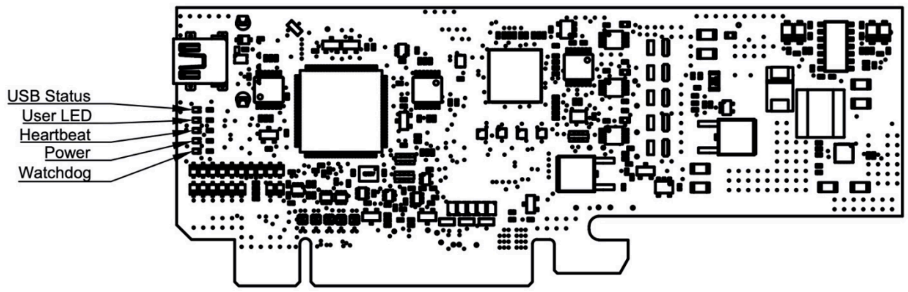
Python

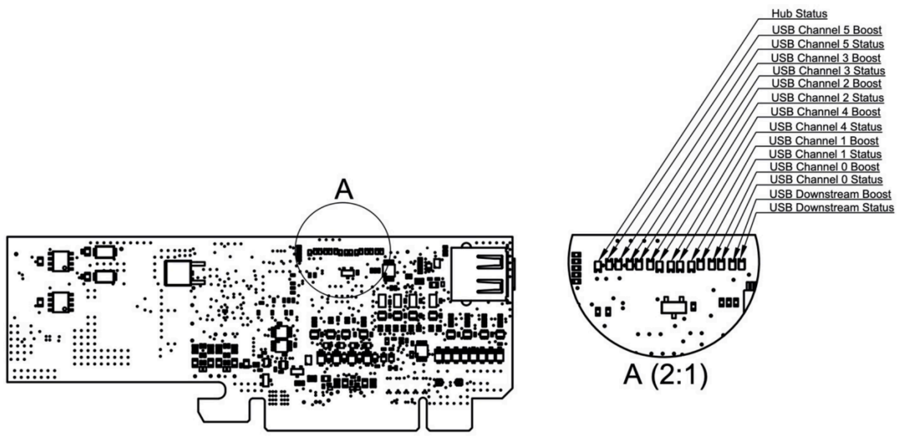
```
import brainstem
#for easy access to error constants
from brainstem.result import Result
import time
import sys

# Create an instance of a MTM-IO-SERIAL module.
mtm = brainstem.stem.MTMIOSerial();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

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

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

Indicators and Connections

LEDs

The MTM-IO-Serial board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below.





Programming Interface

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

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

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-IO-Serial. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-IO-Serial can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStemprotocol (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].setEnable(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-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

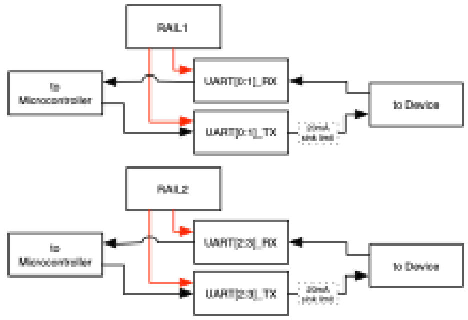
UART entities provide a mechanism to enable and disable UART data lines. All the UARTs that are passed down from the MTM-IO-Serial module can be turned on/off through software control. If a voltage is applied that is higher than the current rail voltage setpoint, each UART transmit line is current limited 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].setEnable(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].setEnable(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].setEnable(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]	setEnable	
	getEnable	
	setPosition	Index 4-7 only
	getPosition	

continues on next page

Table 10 – continued from previous page

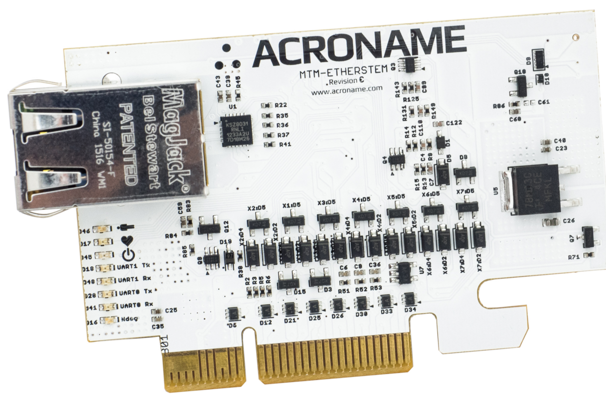
Entity Class	Entity Option	Variable(s) Notes
	setReverse	Index 4-7 only
	getReverse	
i2c[0]	write	
	read	
usb[0]	setPortEnable	
	setPortDisable	
	setDataEnable	
	setDataDisable	
	setHiSpeedDataEnable	
	setHiSpeedDataDisable	
	setPowerEnable	
	setPowerDisable	
	getPortError	
	clearPortErrorStatus	
	getSystemTemperature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

continues on next page

Table 10 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	slotCapacity	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	
port[0-3]	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getPowerEnabled	
	setPowerEnabled	
	getName	
	setName	
	getDataRole	
	getErrors	
port[4]	getDataRole	
USBSystem [0]	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	

1.5.6 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)[20] for your particular operating system and architecture.
- Download HubTool[21] for your particular operating system and architecture.

[20] https://acroname.com/api
[21] https://acroname.com/hubtool

2. Connect to Device

- Utilize the MTM EtherStem by either connecting to the:
  - Onboard USB connection
  - Card edge USB input
  - Through other MTM modules on the local BrainStem bus.

3. Run System

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

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

Congratulations! You are now ready to start exploring the capabilities of the MTM EtherStem. For more information please take a look at our *Getting Started Guide*

Basic Example

C++

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

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

    //Create an instance of a MTM-ETHERSTEM module.
    aMTMEtherStem mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
       return 1;

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

    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

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

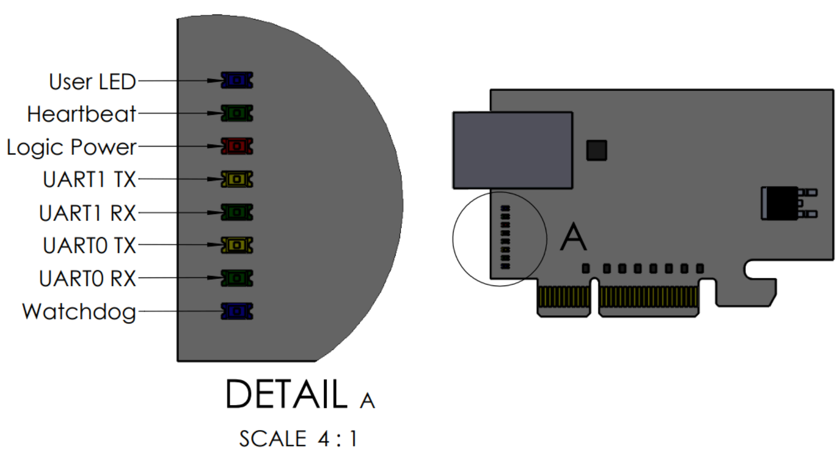
Python

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

# Create an instance of a MTM-ETHERSTEM module.
mtm = brainstem.stem.MTMEtherStem();

# Locate and connect to the first object you find on MTM
result = hub.mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % result)
else:
    print ("Connected to BrainStem module.\n")

#Basic initialization (Get everything turned off).
mtm.system.setLED(0)

##Ready for testing
##Enable LED
mtm.system.setLED(1)

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

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

Indicators and Connections

LEDs

The MTM-EtherStem board has a number of LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below



Programming Interface

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

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

Software Control

A BrainStem link can be established that will give the user access to the resources available on the MTM-USBEtherStem. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS. A BrainStem link to the MTM-USBEtherStem can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStemprotocol (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].setEnable(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 =: 0x0FFF << 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 applicapble 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].setEnable(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

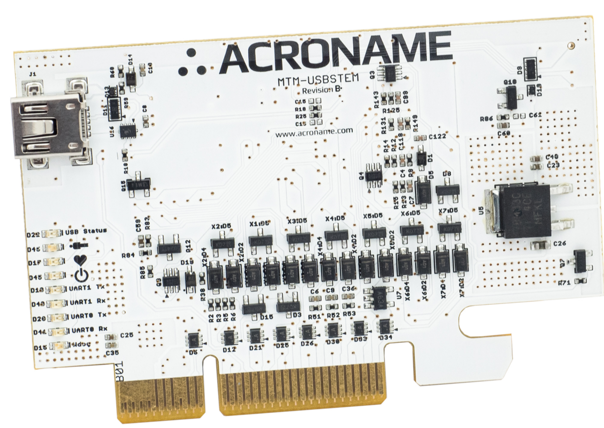
Entity Class	Entity Option	Variable(s) Notes
digital[0-14]	setConfiguration	
	getConfiguration	
	setState	
	getState	
rcservo[0-7]	setEnable	
	getEnable	
	setPosition	Index 4-7 only
	getPosition	
	setReverse	Index 4-7 only
	getReverse	
i2c[0-1]	write	
	read	
analog[0-2]	getValue	
	getVoltage	
	setBulkCaptureSampleRate	
	getBulkCaptureSampleRate	
	setBulkCaptureNumberOfSamples	
	getBulkCaptureNumberOfSamples	
	initiateBulkCapture	
	getBulkCaptureState	
analog[3]	setValue	
	setVoltage	
signal[0-5]	setEnable	
	getEnable	
	setInvert	
	getInvert	
	setT3Time	
	getT3Time	
	setT2Time	
	getT2Time	
store[0-2]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	

continues on next page

Table 11 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	

1.5.7 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)[22] for your particular operating system and architecture.
- Download HubTool[23] for your particular operating system and architecture.

2. Connect to Device

- Utilize the MTM USBStem by either connecting to the:
  - Onboard USB connection
  - Card edge USB input
  - Through other MTM modules on the local BrainStem bus.

3. Run System

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

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

Congratulations! You are now ready to start exploring the capabilities of the MTM USBStem. For more information please take a look at our *Getting Started Guide*

Basic Example

C++

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

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

    //Create an instance of a MTM-USBSTEM module.
    aMTMUSBStem mtm;

    //Connect to the hardware.
    err = mtm.discoverAndConnect(linkType, serialNumber, modelNumber)
    if (err != aErrNone) {
        printf("Error %d encountered connecting to BrainStem module\n", err);
       return 1;

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

(continues on next page)

[22] https://acroname.com/api
[23] https://acroname.com/hubtool

(continued from previous page)

```
    //Basic initialization (Get LEDs turned off).
    mtm.system.setLED(0);

    //Ready for testing
    //Enable LED
    mtm.system.setLED(1);

    //Turn LED off
    mtm.system.setLED(0);

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

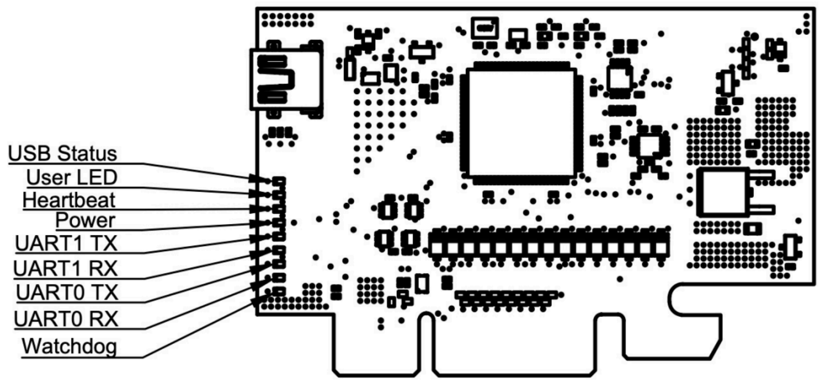
Python

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

# Create an instance of a MTM-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 BrainStemprotocol (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].setEnable(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 =: 0x0FFF << 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 applicapble 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].setEnable(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]	setEnable	
	getEnable	
	setPosition	Index 4-7 only
	getPosition	
	setReverse	Index 4-7 only
	getReverse	
i2c[0-1]	write	
	read	
analog[0-2]	getValue	
	getVoltage	
	setBulkCaptureSampleRate	
	getBulkCaptureSampleRate	
	setBulkCaptureNumberOfSamples	

continues on next page

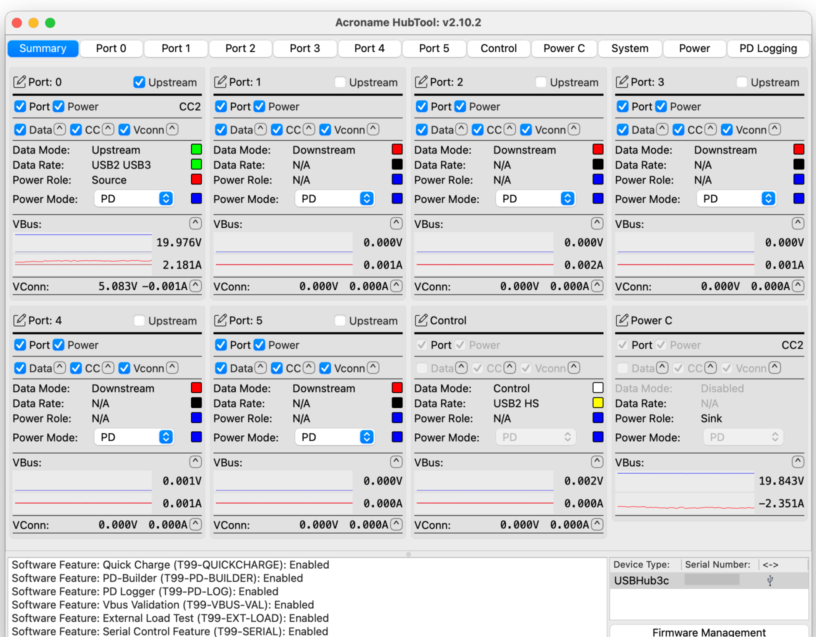
Table 12 – continued from previous page

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

2 Software

Here you can find software developed by Acroname.

2.1 HubTool



HubTool is a utility that lets users view detailed information and control settings of Acroname devices. Users can operate HubTool either on the USB host computer or a computer linked to the Acroname device's Control port.

2.1.1 What can it do?

HubTool provides a simple GUI to:

- Enable and disable ports (virtually plug and unplug)
  - Selectively toggle port data and power connections such as USB 3, USB 2, and VBus independently
- View connection speed
- Visualize per-port current and voltage
- Select automatic and manual upstream host port switching
- Set PD power rules (USBHub3c) and port power modes
- Control Acroname devices over the local network

2.1.2 Which Acroname devices work with HubTool?

USB hubs and switches[24]	USBHub3c[25]
	USBHub3p[26]
	USBHub2x4[27]
	USB-C-Switch[28]
MTM manufacturing test modules[29]	MTM-Load-1[30]
	MTM-DAQ-2[31]
	MTM-PM-1[32]
	MTM-IO-Serial[33]
	MTM-Relay[34]
	MTM-EtherStem[35]
	MTM-USBStem[36]

2.1.3 Host system requirements

HubTool binaries are available for these platforms:

- Windows 10 and higher
- Intel and Apple Silicon Macs running macOS 10.15 or higher
- Linux:
  - x86_64 Ubuntu LTS 16.04, 18.04, 20.04, 22.04, 24.04 - Required dependencies: *apt install xcb**

[24] https://acroname.com/store-grid/field_category/Programmable-USB
[25] https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro
[26] https://acroname.com/store/programmable-industrial-hub-s79-usbhub-3p
[27] https://acroname.com/store/industrial-intelligent-4-port-hub-s77-usbhub-2x4
[28] https://acroname.com/store/programmable-industrial-switch-s85-rdvr-usbcsw
[29] https://acroname.com/manufacturing-test-modules-mtm
[30] https://acroname.com/store/s96-mtm-load-1
[31] https://acroname.com/store/s92-mtm-daq-2
[32] https://acroname.com/products/ACRONAME-MTM-1-CHANNEL-POWER-MODULE
[33] https://acroname.com/products/ACRONAME-MTM-IO-SERIAL-SOFTWARE-CONTROLLED-USB-HUB
[34] https://acroname.com/store/s78-mtm-relay
[35] https://acroname.com/store/s67-mtm-etherstem
[36] https://acroname.com/store/s69-mtm-usbstem

- x86_64 Red Hat 8 - Required dependencies: *dnf install qt5-qtbase-gui*
- x86_64 Red Hat 9 - Required dependencies: *dnf install qt6-qtbase-gui*
- arm64v8 Ubuntu LTS 16.04, 18.04, 20.04, 22.04, 24.04 - Required dependencies: *apt install xcb**
- i686 Ubuntu LTS 16.04 - Required dependencies: *apt install xcb**

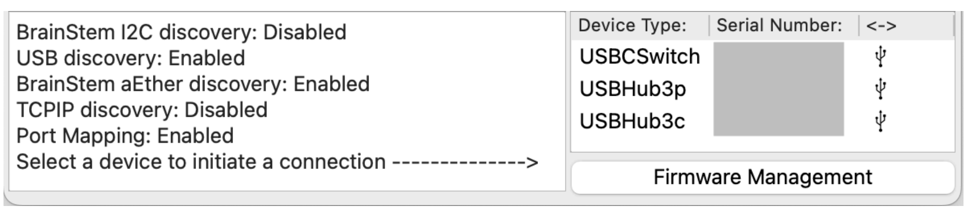
2.1.4 Installation

1. Download HubTool[37] and BrainStem Development Kit (BDK)[38]
2. Open the .dmg / .zip / .tgz
3. Move the contents to a folder, or move just the HubTool application to your preferred location.

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

2.1.5 Usage

Click on the HubTool app to launch it. The lower right panel shows any Acroname devices that are connected. The lower left panel is the console. On launch, HubTool writes default settings to the console.



Startup messages in HubTool Console

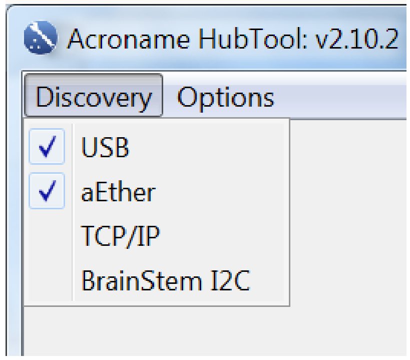
Table 1: *Startup messages*

Console message	Meaning
BrainStem I2C discovery: Disabled	Discover Acroname devices connected to a USB-connected MTM module's I2C bus
USB discovery: Enabled	Discover local USB-connected Acroname devices
BrainStem aEther discovery: Enabled	Discover Acroname devices connected through other applications (e.g. BrainD), local only by default
TCPIP discovery: Disabled	Discover MTM-EtherStem modules connected to the host by ethernet
Port Mapping: Enabled	Get USB descriptors for each downstream device, E.G. vendor ID, serial number

Discovery methods (USB, aEther, TCP/IP, and Brainstem I2C) are enabled and disabled using the Discovery menubar dropdown. Port mapping is enabled in the options menu. Both settings reset to default values with every launch of HubTool.

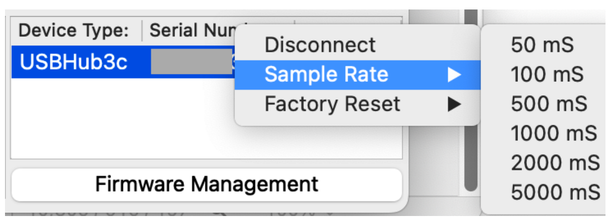
Discovery menu

[37] https://acroname.com/hubtool
[38] https://acroname.com/api



2.1.6 Connecting devices

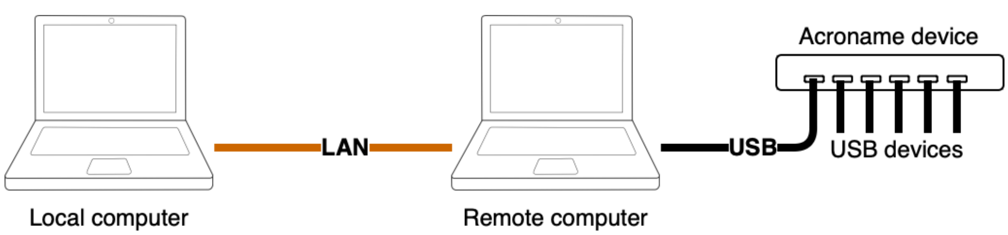
When an Acroname device is connected, the device type, serial number, and an icon indicating the connection type will appear in the lower right pane. If the device is connected by USB to the local host, the USB icon will appear (). If the device is connected to a remote host, the aEther icon will be shown instead (). Click on the device serial number to view the *device interface*. Right-click to bring up a contextual menu to *disconnect*, *set sample rate*, and perform a *factory reset*.



Device context menu

2.1.7 Control devices on remote hosts

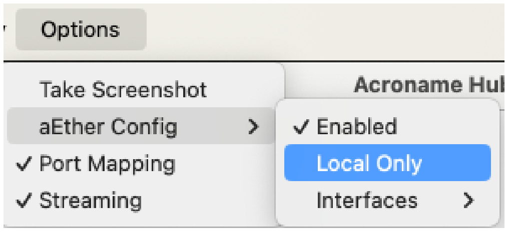
HubTool can also view and control shared Acroname devices connected to other hosts on the same network (WiFi or ethernet). To enable remote control of devices, follow these steps:



Controlling remote devices over the local network

On the remote host:

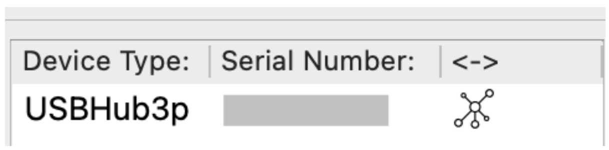
- Connect the device via USB
- Launch HubTool
- Click the device serial number to connect
- Enable aEther and deselect "Local only" in Options > aEther Config



On the local host:

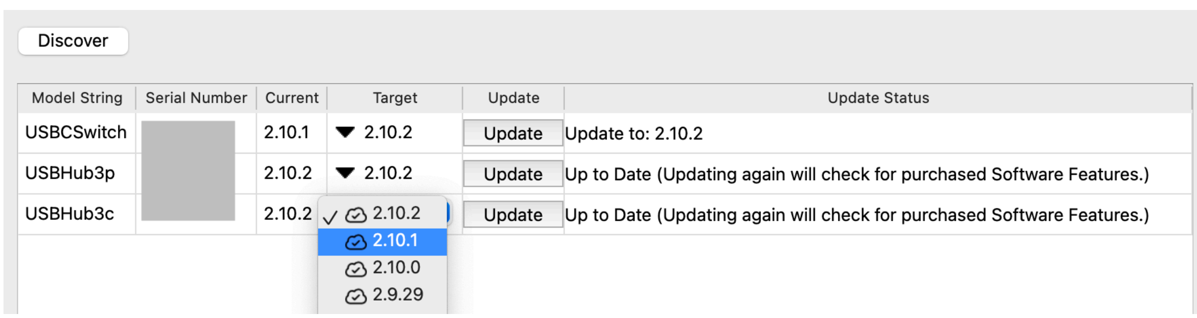
- Launch HubTool
- Enable aEther and deselect "Local only" in Options > aEther Config

The serial number of the remote device and the aEther icon should appear in the local host's device list. Click to view the remote device's interface.



aEther-connected remote device

2.1.8 Updating device firmware



Firmware update panel

To update the firmware of a connected Acroname device, in the lower-right panel, click "Firmware Management" to bring up the firmware update panel. A list of connected devices with serial number, currently installed firmware version, target version, and an update button should appear. Select target firmware version, and click *update* to update the firmware. If the device isn't visible, check that it is connected and click *discover*.

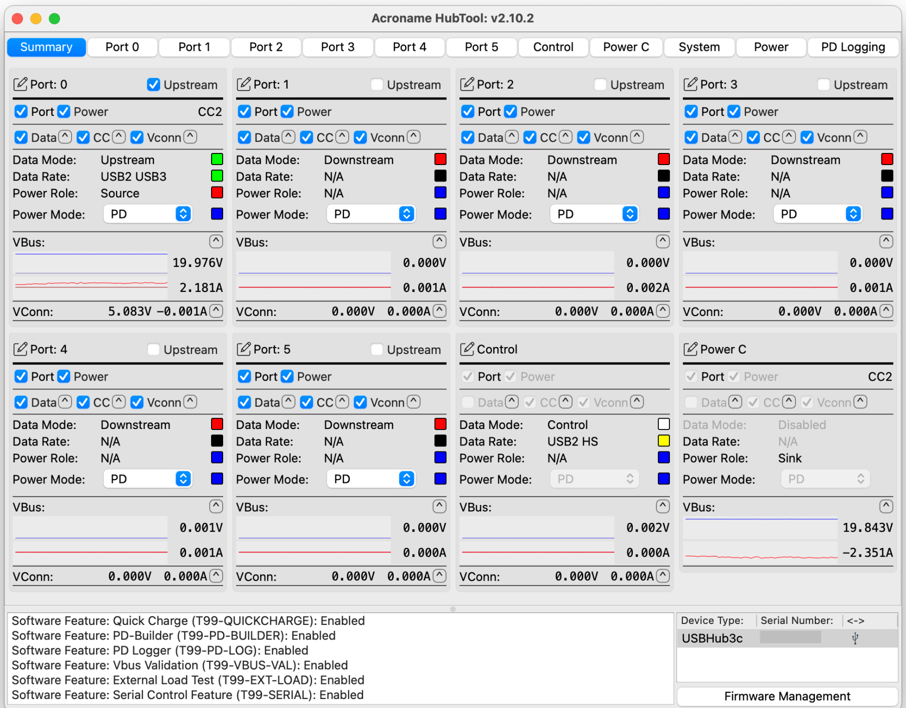
Firmware update is required to enable newly purchased software add-on features, even if the installed version is current.

See the *Firmware Management* documentation for more information on updating device firmware.

2.1.9 Device-specific interfaces

Each Acroname product has a distinct interface in HubTool. Select your product below:

USBHub3c

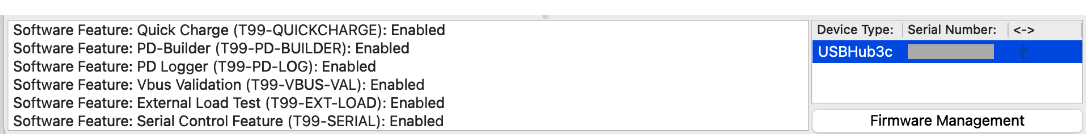


[39] *HubTool interface for USBHub3c*

USBHub3c[40] is an industrial USB-C hub featuring 6 full-featured 10 Gbps USB 3.2 data ports, one USB 2.0 control port, and a dedicated PD power input port. Each data port can provide up to 100 W of power and can act as a downstream-facing or upstream-facing port.

HubTool presents a unified dashboard to control and view the state of USBHub3c.

Add-on software features



Add-on features listed in HubTool console after selecting the USBHub3c on the right

[39] https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro
[40] https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro

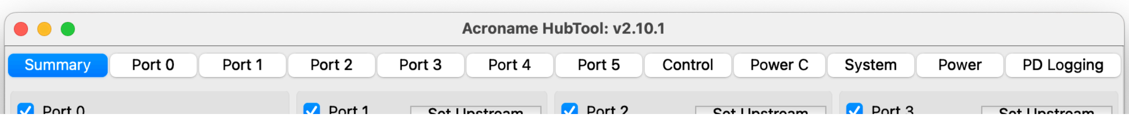
USBHub3c supports add-on software features, which can be purchased to enable new capabilities. When a USBHub3c is selected in HubTool, the console will list the available add-on software features and show their enabled / disabled status on the device.

Table 2: *Add-on licensed software features*

Feature	Capability
Quick charge[41]	Support Qualcomm Quick Charge® 2 and 3 on ports 0 - 5 (Pro model only)
PD builder[42]	Edit Power Data Objects (PDOs) on each port to emulate any USB-PD configuration
PD logger[43]	Log USB-PD communications on all ports
VBus validation[44]	Override normal VBus voltage set points and current limits. Used in testing a device's response to incorrect VBus voltages after USB-PD negotiation.
External load test[45]	Test VBus sinking with external programmable or resistive electronic loads. Use with External Load Expansion Accessory[46]
Serial control[47]	Enable RS-232 serial control of the hub via the serial expansion accessory[48]

Licenses for additional features are available for purchase[49]. After purchase, update the hub firmware to enable the new features.

Dashboard tabs



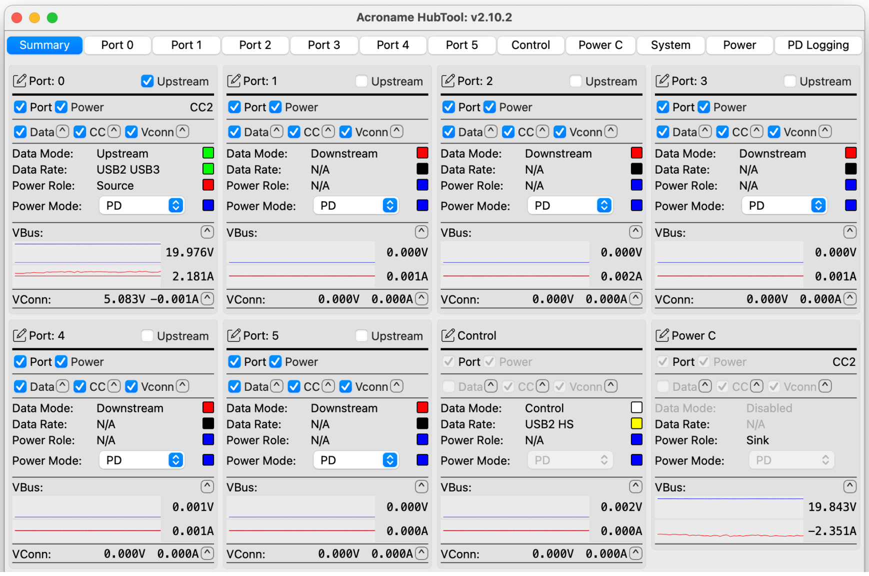
HubTool USBHub3c dashboard tab view

After selecting USBHub3c, HubTool will launch the device dashboard in summary view. Buttons along the top of the window represent tabs for:

41 https://acroname.com/store/t99-quickcharge
42 https://acroname.com/store/t99-pd-builder
43 https://acroname.com/store/t99-pd-log
44 https://acroname.com/store/t99-vbus-val
45 https://acroname.com/store/t99-ext-load
46 https://acroname.com/store/s101-load-exp
47 https://acroname.com/store/t99-ext-load
48 https://acroname.com/store/s103-serial-exp
49 https://acroname.com/store-grid/field_category/Software-Licenses

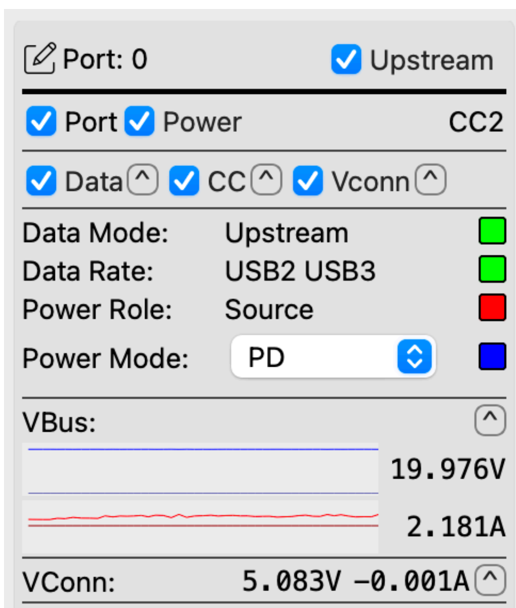
Summary tab

The summary tab displays simplified interface panels for Ports 0-5, Control port, and Power C port (power input on rear panel).



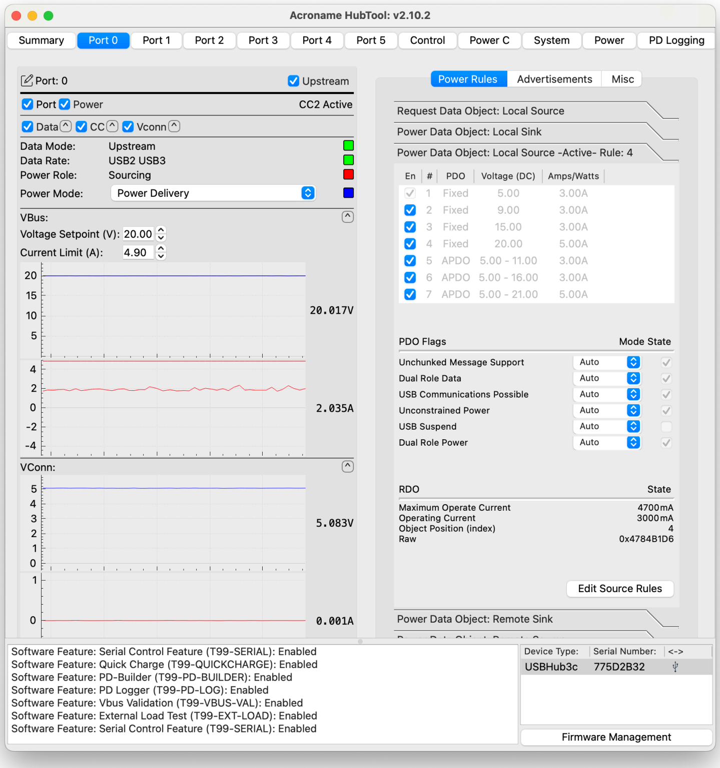
Summary tab view

Each port interface panel shows a consolidated view of the corresponding *port tab*.



Port panel in summary view

Port tabs



The port tabs show a more detailed view of each port. Control and Power C ports have specialized functions and some options are unavailable. The left side of the port tab contains *Power and data settings*. The right side of the port tab views contains sub tabs for USB-PD *Power rules*, *Advertisements*, and *Miscellaneous messages and settings*.

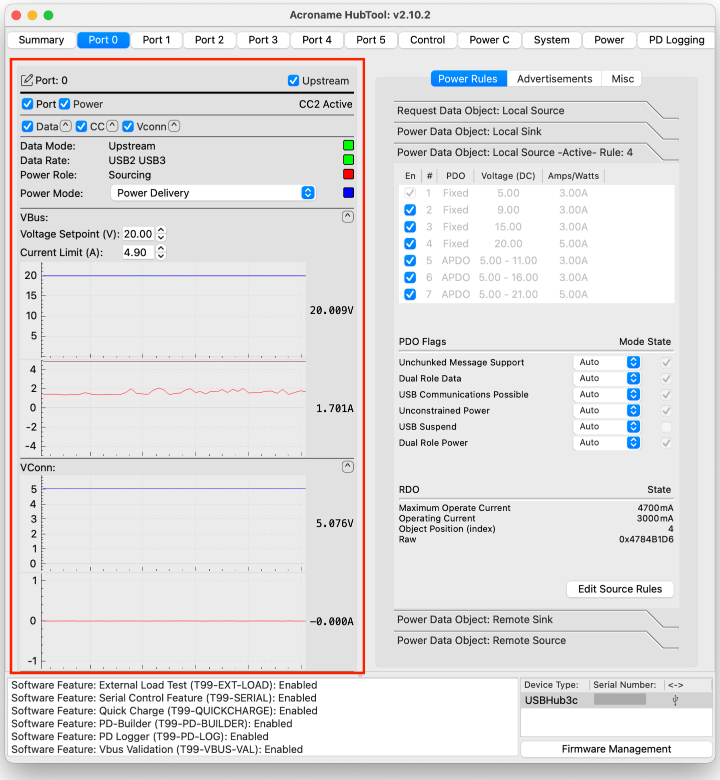
Power and data settings

Port tab view - power and data settings

The left side of each port tab view contains power and data settings for the port. At the top are the *port name* and *upstream toggle*.

Port name (editable ) - friendly port name used by HubTool and ControlRoom)

Upstream toggle - indicates and selects which port is in upstream mode and able to connect to a host



Power and data toggles

Below the port name are the power and data toggles. Click the carats (^) to expand the view.

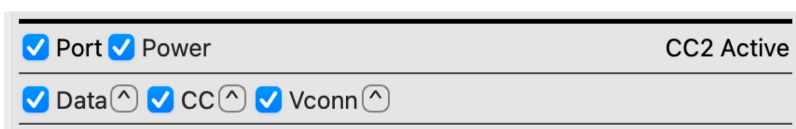


Table 3: Power and data toggles

Toggle name	Enables and disables
Port	Port data and power
Power	VBus
Data ^	All data
HS	USB 2 (High Speed) pins
SS	USB 3 (Super Speed) pins
CC ^	Both CC pins
CC1, CC2	CC pins individually
Vconn ^	Vconn pins
Vconn1, Vconn2	Vconn pins individually

Data and power status, power mode

Below the toggles are *data mode*, *data rate*, and *power role* status indicators, followed by the *power mode* menu. The virtual LED color corresponds to the color of the real LED indicators on the USBHub3c front panel.

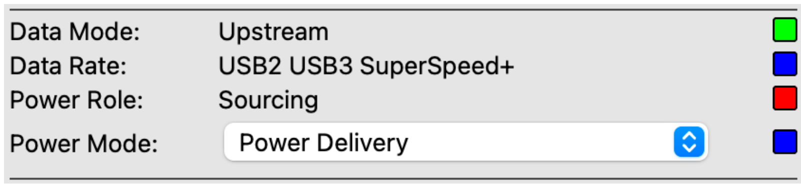
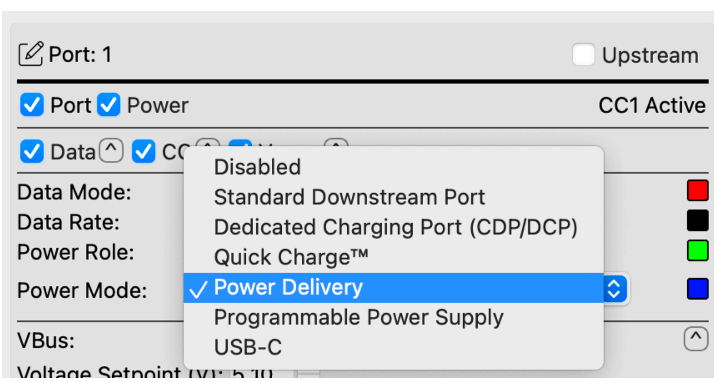
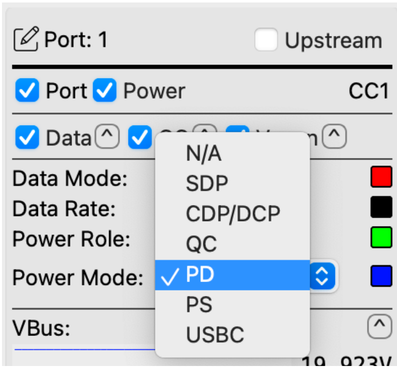


Table 4: Data and power status

Name	Options	Virtual LED color
Data mode	Upstream (to host)	Green
	Downstream	Red
	Control port	White
Data rate (highest speed)	USB 2	Yellow
	USB 3 (5 Gb/s)	Green
	USB 3 SuperSpeed+ (10 Gb/s)	Blue
Power role	Sinking (receives power)	Green
	Sourcing (provides power)	Red

Port power mode is selected from a dropdown menu. In the summary view, the modes are abbreviated.





Power mode selection in port view (left) and summary view (right)

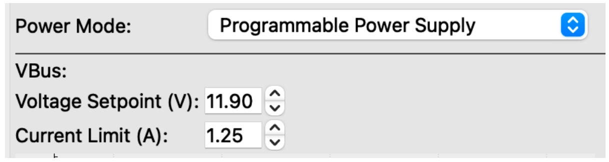
Table 5: Power modes

Summary abbreviation	Power mode	Definition	Virtual LED
N/A	Disabled	Power off	Black
SDP	Standard Downstream Port	5 V, 900 mA	Red
CDP/D(	Dedicated Charging Port (CDP/DCP)	5 V, 3 A	Red
QC	Quick Charge™	Qualcomm® Quick Charge™ 2.0 and 3.0 fast charging, requires Quick Charge feature license 3 A	Green
PD	Power Delivery	USB-PD revision 2.0 and 3.2 compliant PD fixed voltage 5 - 20 V up to 5 A, USB PD-PPS and QC 4+ variable voltage up to 5 - 21 V, 5 A	Blue
PS	Programmable Power Supply	Manually set 2.8 V – 21 V and 0 A – 5 A, requires VBus validation feature license	White
USBC	USB-C	5 V, 3 A	Purple

In USB-C and PD modes, the VBus is only enabled after strapping resistors are detected on the CC lines, while SDP, CDP/DCP, PS, and QC modes provide VBus continuously. In CDP and SDP modes, the port acts as a source if downward facing, and a sink if upward facing.

VBus voltage and current control

(Requires VBus validation add-on[50])



Manually set VBus voltage and current

When **Programmable Power Supply (PS)** mode is enabled, VBus voltage setpoint and current limit can be set directly, transforming the port into a programmable power supply capable of supplying up to 100 Watts. These are saveable settings (System tab / Save) that persist through reset.

In **Power Delivery (PD)** mode, the user-set values override the negotiated values. However, PD negotiations are still active and PD events or errors can trigger re-negotiation, which will replace the user-set values. These values should only be changed when the power role is set to *sourcing*.

In **USB-C, SDP, CDP/DCP** modes, VBus settings revert to 5 V and default current on port connect or disconnect.

Setting	Value
Voltage setpoint	2.8 V – 21 V
Current limit	0 A – 5 A

Warning: Can damage attached equipment!

Voltage and current display

Shows a graph of the VBus and VConn voltage and current of the port.

VBus and VConn panel view

Clicking on the port graph pops up the Port Plots window with a larger rolling strip chart showing all ports. Click the carat ()to expand each port chart. If *PD logging* is enabled, PD messages will be marked in yellow.

Port Plots window

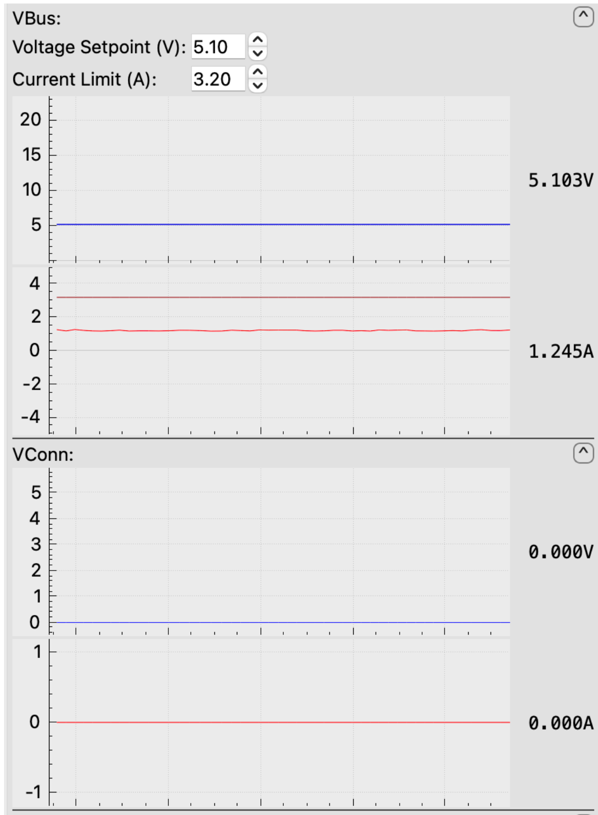
Left vertical axis – voltage (V)

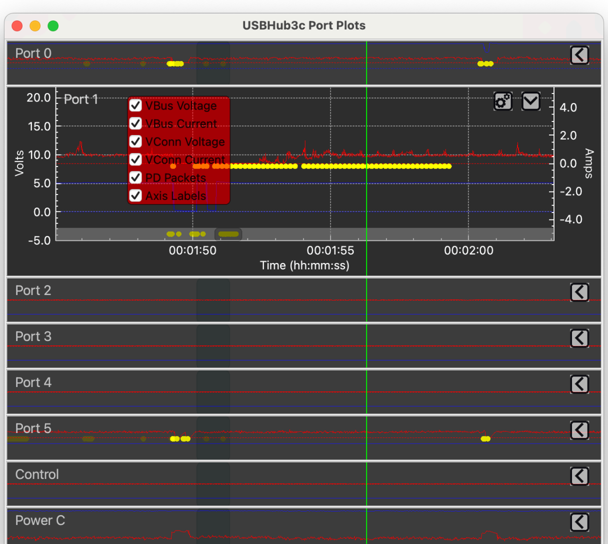
Right vertical axis – current (A)

Horizontal axis – time (s)

By default, the plot will be scrolling. To stop scrolling, drag the dark gray scroll bar to the left. To enable scroll-to-zoom, click the desired axis.

[50] https://acroname.com/reference/devices/usbhub3c/software_features/vbus_validation.html


VBus:
Voltage Setpoint (V): 5.10
Current Limit (A): 3.20
5.103V
1.245A
VConn:
0.000V
0.000A


USBHub3c Port Plots
Port 0
Port 1
VBus Voltage
VBus Current
VConn Voltage
VConn Current
PD Packets
Axis Labels
Volts
Amps
00:01:50
00:01:55
00:02:00
Time (hh:mm:ss)
Port 2
Port 3
Port 4
Port 5
Control
Power C

Device descriptors

If Options > Port Mapping is selected, when a device is attached to a downstream port, its descriptors will scroll at the bottom of the port panel. Click the carat ( ) to expand:

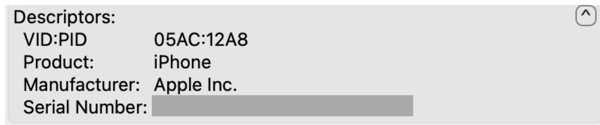


Table 6: *Descriptor table*

Descriptor	Content
VID:PID	16-bit vendor ID and 16-bit product ID
Product	Product name string
Manufacturer	Manufacturer name string
Product serial number	Product serial number
ATT	Indicates device is attached

Power rules

The right side of the port tab views contains sub-tabs for USB-PD power rules, advertisements, and miscellaneous messages and settings. The power rules section presents a set of vertical tabs on the right side of the port view:

Power rules vertical tabs

The vertical tabs are:

- *Request Data Object*
- *Power Data Object: Local Sink*
- *Power Data Object: Local Source*
- *Power Data Object: Remote Sink*
- *Power Data Object: Remote Source*

Request Data Object: Local [Source or Sink]

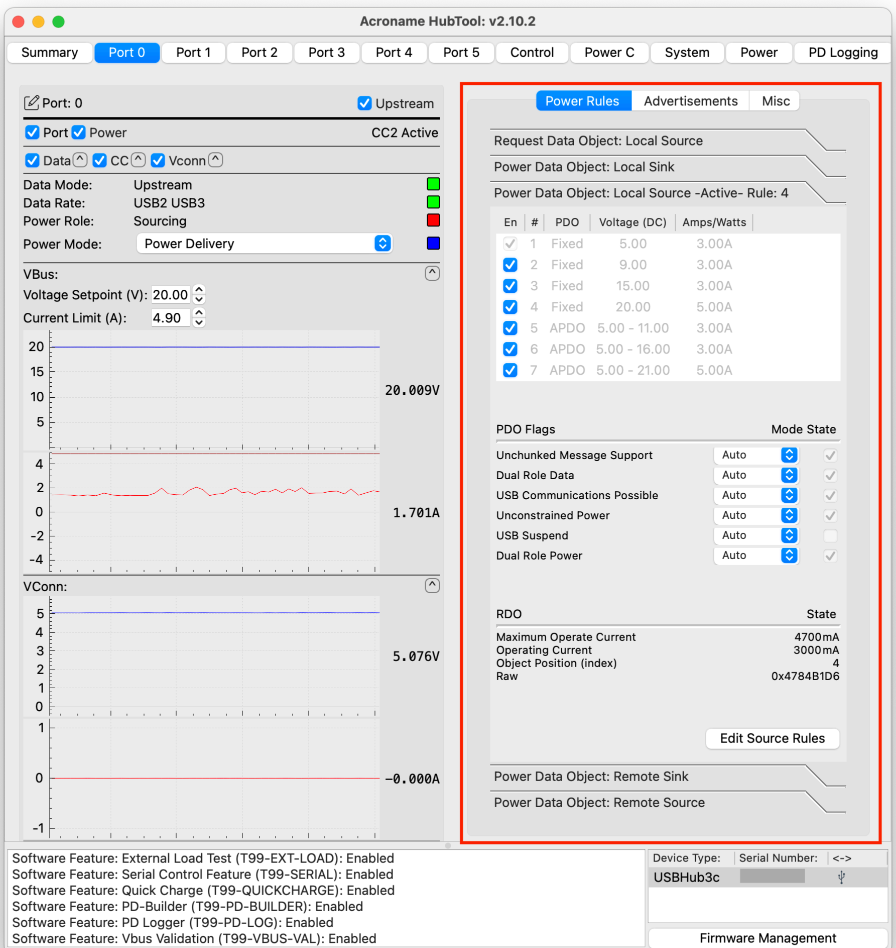
A Request Data Object (RDO) tells the host about the sink's capabilities and selects one of the Power Data Objects (PDOs) offered by the source by specifying the PDO's index (1-7).

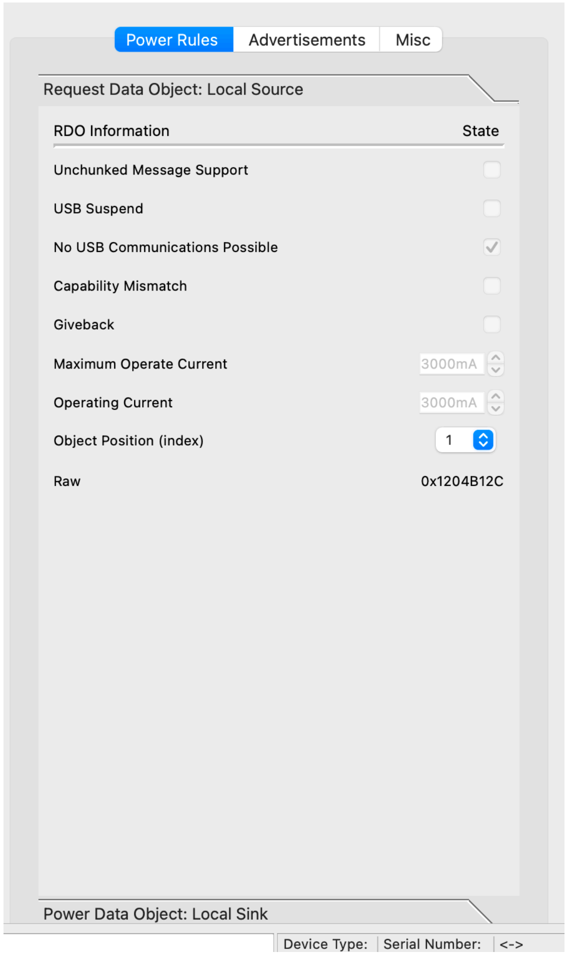
When the port is configured as a sink:

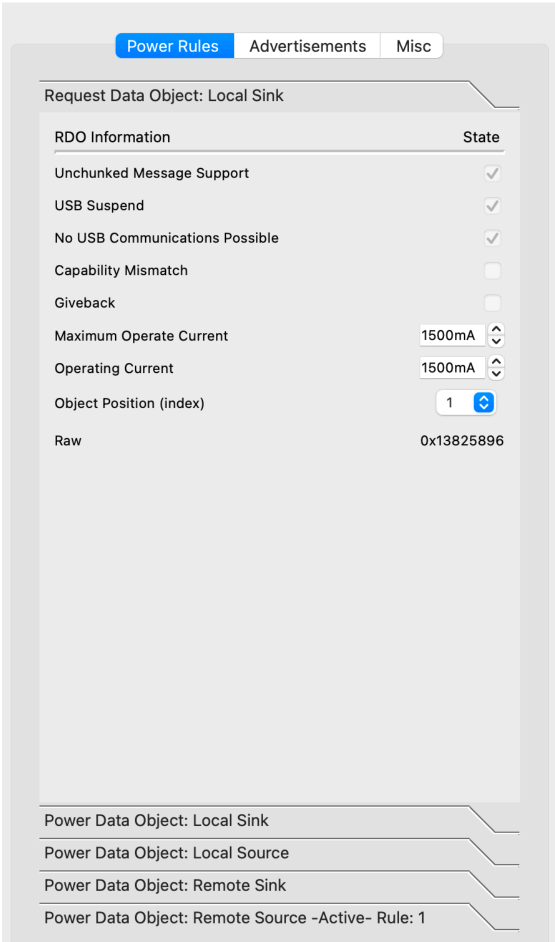
- Tab label shows "Local Sink"
- Panel displays the RDO sent from the hub (sink) to the source device

When the port is configured as a source:

- Tab label shows "Local Source"
- Panel displays the RDO received by the hub (source) from the sinking device (informational only)


Acroname HubTool: v2.10.2
Summary
Port 0
Port 1
Port 2
Port 3
Port 4
Port 5
Control
Power C
System
Power
PD Logging
Port: 0
Upstream
Port
Power
CC2 Active
Data
CC
Vconn
Data Mode: Upstream
Data Rate: USB2 USB3
Power Role: Sourcing
Power Mode: Power Delivery
VBus:
Voltage Setpoint (V): 20.00
Current Limit (A): 4.90
20.009V
1.701A
VConn:
5.076V
-0.000A
Power Rules
Advertisements
Misc
Request Data Object: Local Source
Power Data Object: Local Sink
Power Data Object: Local Source -Active- Rule: 4
En # PDO Voltage (DC) Amps/Watts
1 Fixed 5.00 3.00A
2 Fixed 9.00 3.00A
3 Fixed 15.00 3.00A
4 Fixed 20.00 5.00A
5 APDO 5.00 - 11.00 3.00A
6 APDO 5.00 - 16.00 3.00A
7 APDO 5.00 - 21.00 5.00A
PDO Flags
Mode State
Unchunked Message Support Auto
Dual Role Data Auto
USB Communications Possible Auto
Unconstrained Power Auto
USB Suspend Auto
Dual Role Power Auto
RDO
State
Maximum Operate Current 4700mA
Operating Current 3000mA
Object Position (index) 4
Raw 0x4784B1D6
Edit Source Rules
Power Data Object: Remote Sink
Power Data Object: Remote Source
Software Feature: External Load Test (T99-EXT-LOAD): Enabled
Software Feature: Serial Control Feature (T99-SERIAL): Enabled
Software Feature: Quick Charge (T99-QUICKCHARGE): Enabled
Software Feature: PD-Builder (T99-PD-BUILDER): Enabled
Software Feature: PD Logger (T99-PD-LOG): Enabled
Software Feature: Vbus Validation (T99-VBUS-VAL): Enabled
Device Type:
Serial Number:
USBHub3c
Firmware Management


Power Rules
Advertisements
Misc
Request Data Object: Local Source
RDO Information
State
Unchunked Message Support
USB Suspend
No USB Communications Possible
Capability Mismatch
Giveback
Maximum Operate Current
3000mA
Operating Current
3000mA
Object Position (index)
1
Raw
0x1204B12C
Power Data Object: Local Sink
Device Type:
Serial Number:
<->



RDO local source (left) and local sink (right) views

Table 7: *Request Data Object tab options*

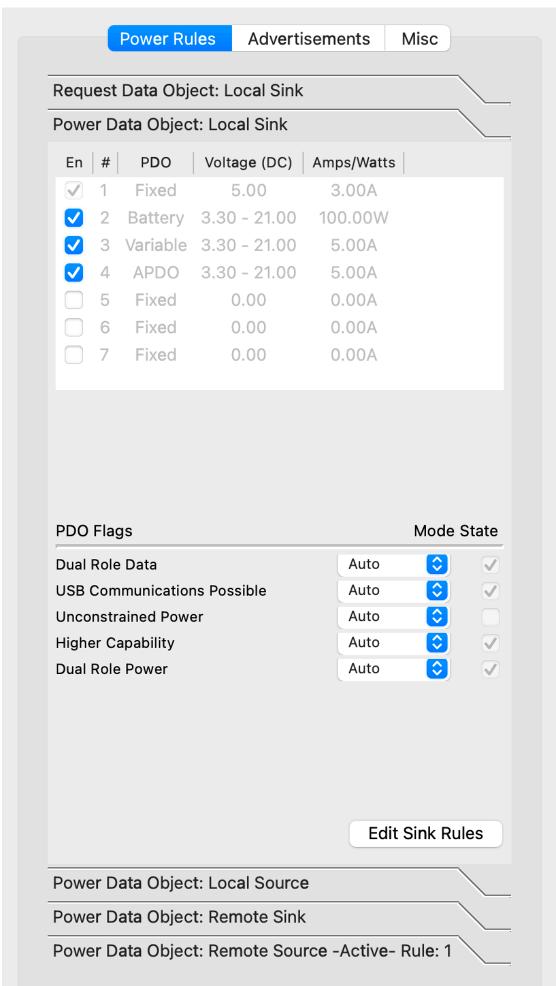
RDO Information	Meaning
Unchunked message support	Checked = unchunked mode; supports larger data payloads Unchecked = legacy "chunked" mode; leave unchecked for maximum compatibility
USB suspend	Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
No USB communications possible	Device can not communicate over USB
Capability mismatch	Source PDO modes are not sufficient for device to operate at full capability
Giveback	Device can go to a lower power state if needed
Maximum operate current	Maximum current the sinking device is capable of drawing
Operating current	Nominal operating current of the sinking device
Object position (index)	Index of PDO requested, range 1-7
Raw	The RDO message expressed as 32-bit hexadecimal value

Power Data Object: Local Sink

This tab shows the settings for the port's local sink Power Data Object (PDO). This is how the port tells a remote source about its capabilities and how much power it can accept. If the port is not operating as a sink, these settings have no effect. At the top of the panel is a table of PDO modes that the port can accept as a sink.

Table 8: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage



Local Sink PDO list and flags

Table 9: *PDO flags*

PDO flag	Mode	State	Meaning
Dual Role Data	Enable, Disable, Auto	Checked if flag is enabled	Can be host or peripheral
USB Communications Possible			Supports USB data
Unconstrained Power			AC powered or large battery
Higher Capability			Sink needs more than 5 V for full functionality
Dual Role Power			Can source or sink power

Edit sink rules button (requires PD Builder license)

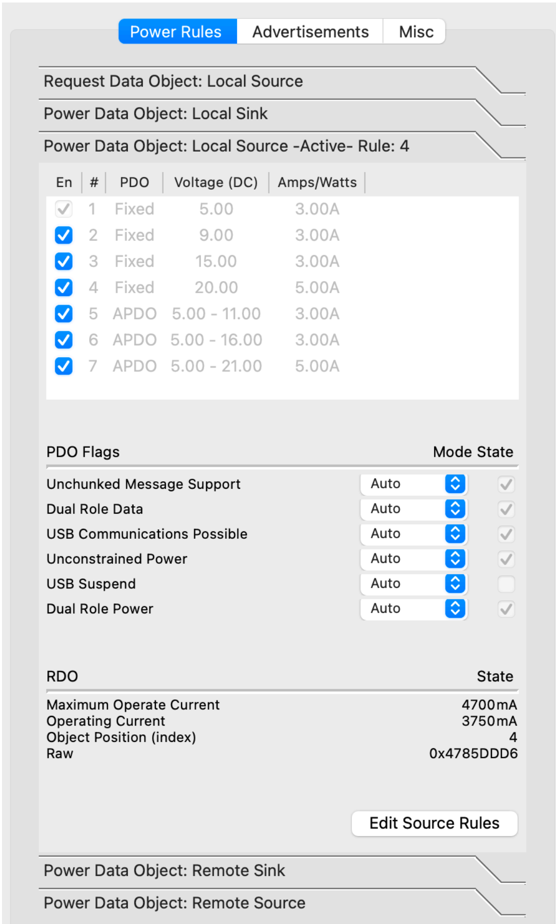
Pops up *Power Rule Editor*.

Power Data Object: Local Source

This tab shows the settings for the port's local source Power Data Object (PDO). If the port is not operating as a source, these settings have no effect. At the top is a table of PDO modes that the port can provide as a source. If active, the PDO rule index is listed in the tab title.

Table 10: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage



Local source PDO list and flags

Table 11: *PDO flags*

PDO flag	Mode	State	Meaning
Unchunked Message Support	Enable, Disable, Auto	Checked if flag is enabled	Checked = unchunked mode; supports larger data payloads Unchecked = legacy "chunked" mode; leave unchecked for maximum compatibility
Dual Role Data			Can be host or peripheral
USB Communications Possible			Supports USB data
Unconstrained Power			AC powered or large battery
USB Suspend			Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
Dual Role Power			Can source or sink power

Edit source rule button (requires PD Builder license)

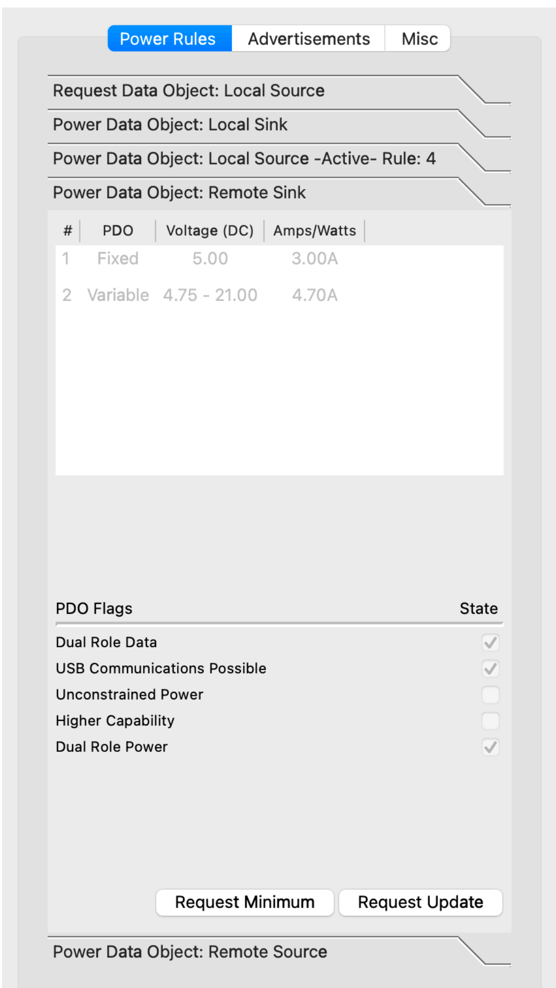
Pops up *Power Rule Editor*.

Power Data Object: Remote Sink

This tab shows the power sink settings for the remote device's Power Data Object (PDO). Since these settings are controlled by the remote device, this tab is informational only. At the top is a list of PDO modes that the remote device can accept as a sink.

Table 12: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage



Remote sink PDO list and flags

Table 13: *PDO flags (informational only)*

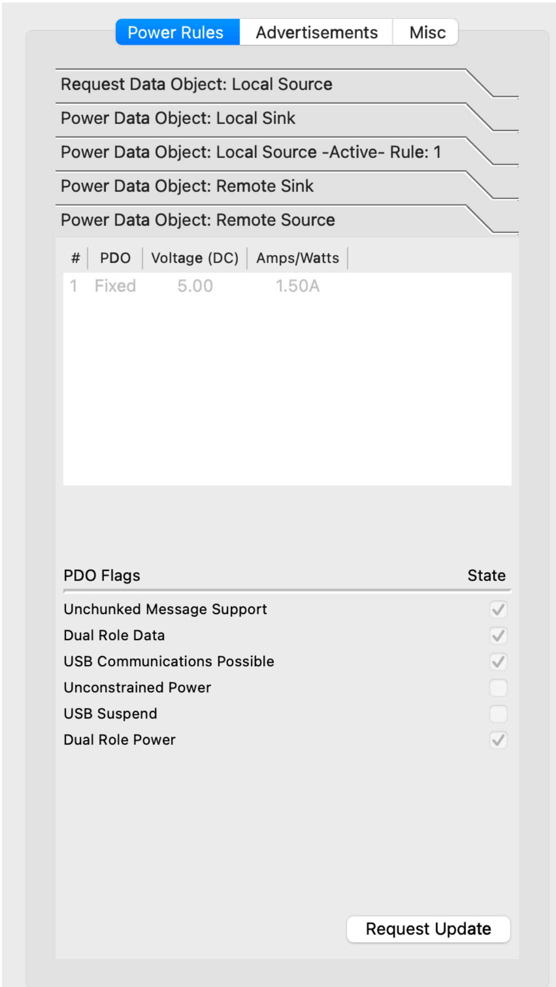
PDO flag	State	Meaning
Dual Role Data	Checked if flag is enabled	Can be host or peripheral
USB Communications Possible		Supports USB data
Unconstrained Power		AC powered or large battery
Higher Capability		Sink needs more than 5 V for full functionality
Dual Role Power		Can source or sink power

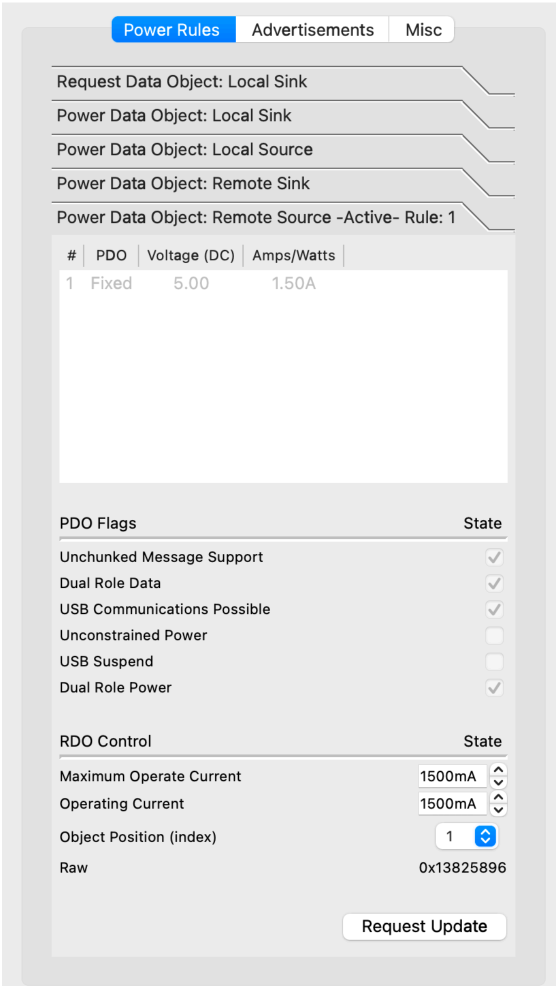
Request minimum, Request update buttons

[Not implemented]

Power Data Object: Remote Source

This tab shows settings for the remote device's Power Data Object (PDO) and its Request Data Object if active. Since the PDO settings are controlled by the remote device, the PDO mode list and flags are informational only. At the top is a list of PDO modes that the remote device can provide as a source. If the remote device is acting as a source, the active PDO rule index is listed in the tab title.


Power Rules
Advertisements
Misc
Request Data Object: Local Source
Power Data Object: Local Sink
Power Data Object: Local Source -Active- Rule: 1
Power Data Object: Remote Sink
Power Data Object: Remote Source
PDO Voltage (DC) Amps/Watts
1 Fixed 5.00 1.50A
PDO Flags
State
Unchunked Message Support
Dual Role Data
USB Communications Possible
Unconstrained Power
USB Suspend
Dual Role Power
Request Update



Remote source PDO list, flags and RDO control

Table 14: *PDO mode list column headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
#	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Voltage (DC)	PDO voltage (range if variable)
Amps / Watts	PDO maximum amperage or wattage

Table 15: *Remote source PDO flags (informational only)*

PDO flag	State	Meaning
Unchunked Message Support	Checked if flag is enabled	Checked = unchunked mode; supports larger data payloads Unchecked = legacy "chunked" mode; leave unchecked for maximum compatibility
Dual Role Data		Can be host or peripheral
USB Communications Possible		Supports USB data
Unconstrained Power		AC powered or large battery
USB Suspend		Checked = OK for host to suspend device when in USB suspend mode Unchecked = host should not suspend device in USB suspend mode
Dual Role Power		Can source or sink power

RDO control

When the hub port is acting as a sink, some sink RDO settings can be controlled directly from the bottom of the Remote Source tab. These settings mirror the fields in the *Request Data Object: Local Sink* tab.

Table 16: *Remote source RDO control settings*

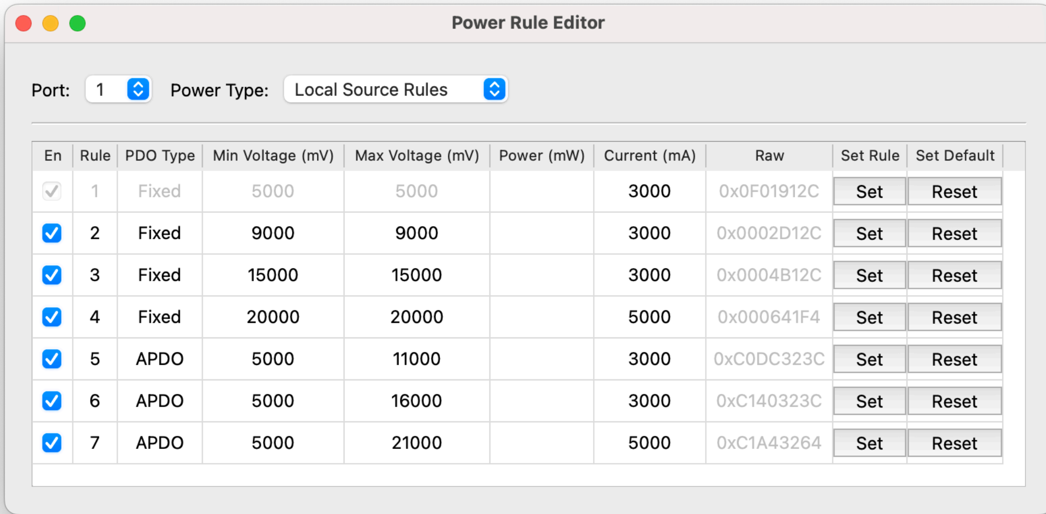
RDO Information	Meaning
Maximum operate current	Maximum current the sinking device is capable of drawing
Operating current	Nominal operating current of the sinking device
Object position (index)	Index of PDO requested, range 1-7
Raw	The RDO message expressed as 32-bit hexadecimal value

Request update button

[Not implemented]

Power rule editor (requires PD builder add-on feature[51])

[51] https://acroname.com/store/t99-pd-builder



The power rule editor sets sink and source power rules for each port.

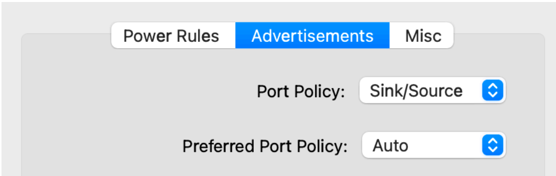
Table 17: *Power rule editor headings*

Heading	Meaning
En	Enabled; index 1 is always enabled
Rule	Index of PDO mode (1 – 7)
PDO	PDO type (fixed, battery, variable, APDO)
Min Voltage (mV)	Min PDO voltage
Max Voltage (mV)	Max PDO voltage
Power (mW)	Max PDO power
Current (mA)	Max PDO current
Raw	Raw bytes of PDO message
Set rule	Set rule after changes
Set default	Return rule to default

Table 18: *PDO Types*

PDO type	Definition
Fixed	Fixed supply voltage of 5 V, 9 V, 15 V, 20 V, current limit 0 A – 5 A. Source must have at least one fixed PDO at 5 V
Variable	Represents a poorly regulated supply. The voltage must stay between the minimum and maximum voltage; minimum voltage must be >= 80% of maximum voltage
Battery	Batteries can be connected to VBus directly as a source. Specifies maximum and minimum voltage (in 50 mV steps) and maximum power (in 250 mW steps)
APDO	Programmatically controllable source. Voltage range set in 20 mV steps, up to 3.3V – 21V. Current limit 0 A – 5 A

Advertisements



Power role advertisements subtab

USB-PD advertisements tell connected devices the power policy of the port and its preferred policy.

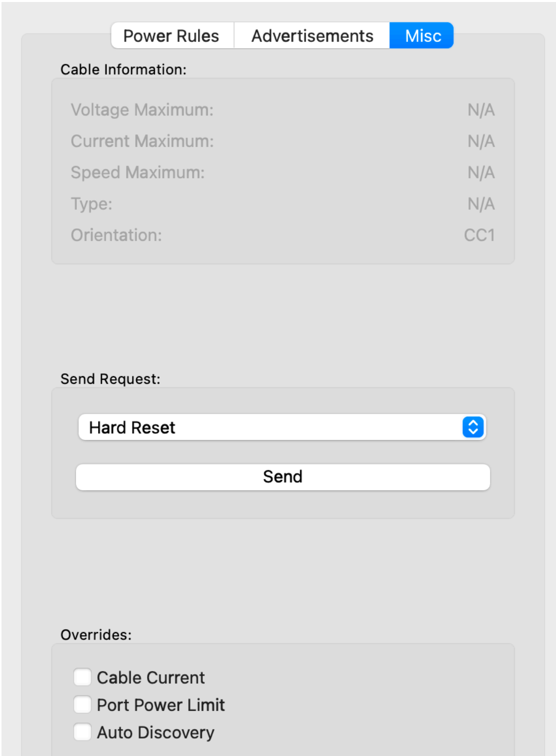
Port policy

- **None** - no power
- **Source** - provides source
- **Sink** - receives power
- **Sink** / **Source** - can provide or receive power according to the *Preferred port policy*

Preferred port policy (Only has an effect if port policy is set to *Sink / Source*)

- **None** - no power
- **Source** - provides power
- **Sink** - receives power
- **Follow data** - provides power if downstream-facing, receives power if upstream-facing
- **Auto** - negotiate with the connected device

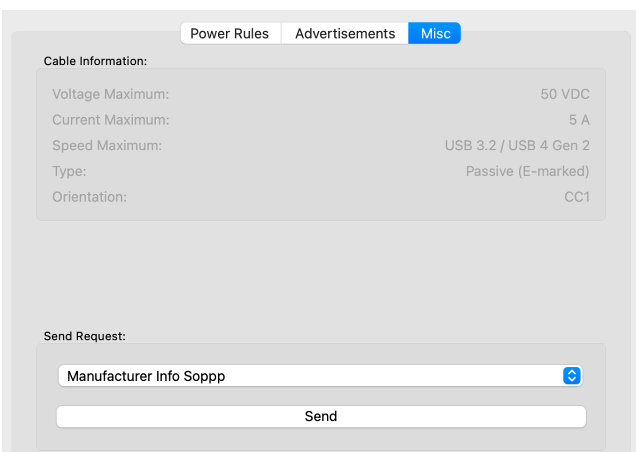
Misc



Misc subtab

The Misc subtab contains *Cable information*, *Send request*, and *Overrides*

Cable information



Cable information panel with e-mark data

Displays power and data capacity for e-marked cables connected to the hub and a device.

- Hub port must be providing VConn (VConn strip chart showing 5 V)

- If VConn strip chart shows 0 V, send "VConn Swap" message to swap whether the hub or device is providing VConn

To read e-mark without a device, connect both ends of the cable to ports of USBHub3c, with one port set as *sink* or *auto* and the other set to *source* or *auto*. View the cable info panel on the source port.

Send Request

This control sends USB PD requests to the connected device to initiate a connection sequence. There is no guarantee that the request will succeed, and many of these request messages will be unsupported by the connected device.

Send request	Definition
Hard Reset	Reinitialize PD communications and cycle VBus
Soft Reset	Reinitialize PD communications, but maintains power connection
Data Reset	Cycle USB data connection [unimplemented]
Power Role Swap	Exchange source and sink power roles
Power Fast Role Swap	Fast exchange source and sink power roles
Data Role Swap	Swap upstream and downstream ask roles
VCONN Swap	Swap whether the hub or device is supplying VConn
Sink Go To Min	Sink goes to minimum power draw
Request Remote Source PDOs	Get PDO from remote source
Request Remote Sink PDOs	Get PDO from remote sink
Request Remote Extended Source Caps	Get extended source capabilities
Request Remote Extended Sink Caps	Get extended sink capabilities
Status	Get status
PPS Status	Get PPS status
Battery Capabilities	Get battery capabilities (Design cap, last full cap)
Battery Status	Get Battery status (state of charge, charging status)
Manufacturer Info Sop	Get device manufacture info
Manufacturer Info Sopp	Get cable manufacturer info (nearest VConn sourcing port)
Manufacturer Info Soppp	Get cable manufacturer info (opposite from VConn sourcing port)
Discover Identity Sop	Device type and capabilities
Discover Identity Sopp	Cable type and capabilities (Vconn end)
Discover Identity Soppp	Cable type annd capabilities (not VConn end)
Revision	USB PD revision and version numbers for connected device
Source Info	Source port power capability
Country Codes	Country of origin
Country Info	Country of origin

Overrides

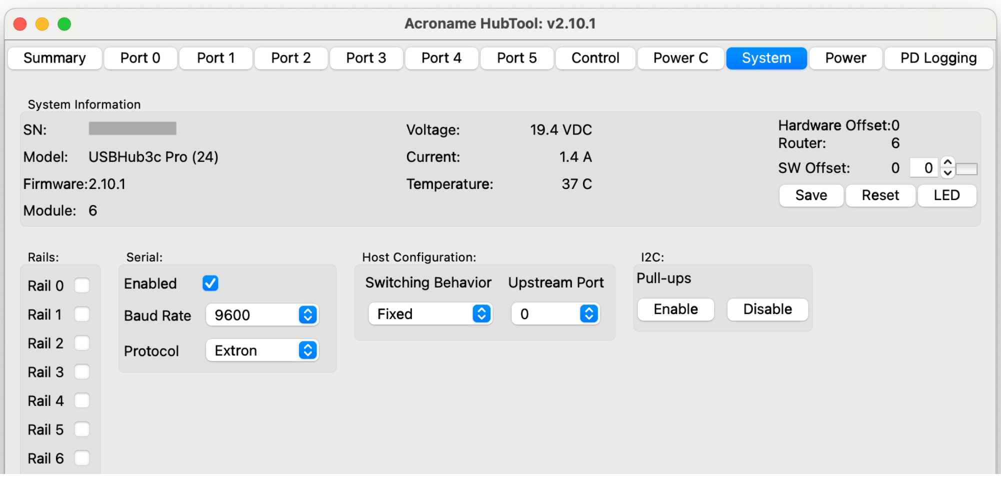
Cable current – overrides the 3 A current limit for cables that don't specify 5 A via e-mark

Port power limit – overrides port power budgeting and allows full power output up to 105 W per port

> **Warning:** Can exceed power supply current limit and brown out the hub, triggering a reset

Auto discovery – overrides the auto discovery feature. When enabled, the hub will only establish a basic power connection and not request vendor information. Used for legacy compatibility and debugging.

System tab



HubTool USBHub3c system tab

The system tab contains information and settings for the hub.

System information

Heading	Meaning
SN	Hub serial number
Model	Hub model name and number, e.g. USBHub3c Pro (24)
Firmware	Currently installed firmware version
Module	Address of the module on the Brainstem network
Voltage	Voltage at the input power source port
Current	Input current at the input power source port
Temperature	System temperature
Hardware offset	Increments Brainstem module address by a hardware setting - fixed to 0 in USBHub3c
Router	Address of the routing Brainstem module
Software offset	Increment the module address. Requires Save and Reset
Save	Store select changed settings
Reset	Soft reset of the hub
LED	Toggles blue user LED on front panel, used debugging and to identify the hub

Rails

Requires External Load[52] add-on feature. Array of toggles to enable VBus rails 0 – 6 on the 20-pin expansion connector. Rails 0 - 5 represent ports 0 – 5. Rail 6 is an additional 5 V rail for triggering or powering external devices. Default = off

Serial

Requires RS232 Serial Control Feature[53] feature. Use Serial RS232 Expansion Connector Accessory[54] to break out the serial pins.

- Enabled (toggle)
- Baud rate (1200 – 115200 bps)
- Protocol (*Undefined*, *Extron*, *BrainStem*)

52 https://acroname.com/store/t99-ext-load
53 https://acroname.com/store/t99-serial
54 https://acroname.com/store/s103-serial-exp

Host configuration

Switching behavior

- Fixed (default)
- Port priority - sets lowest-numbered port that is connected to a host-capable device to upstream mode

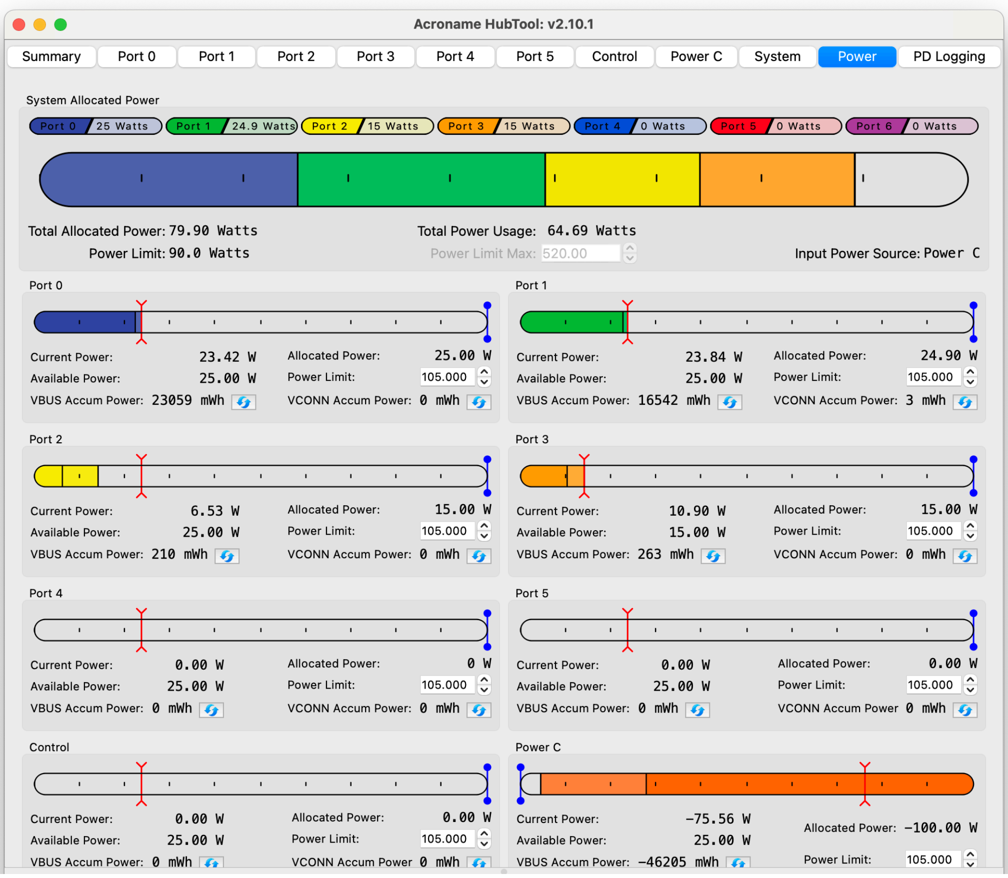
Upstream port

Shows current upstream port. Select Port 0 – 5 to connect to host - default = 0

I2C

Enables and disables pullups on the I2C bus - IO3 (SDA) and IO4 (SCL) on the expansion connector

Power tab

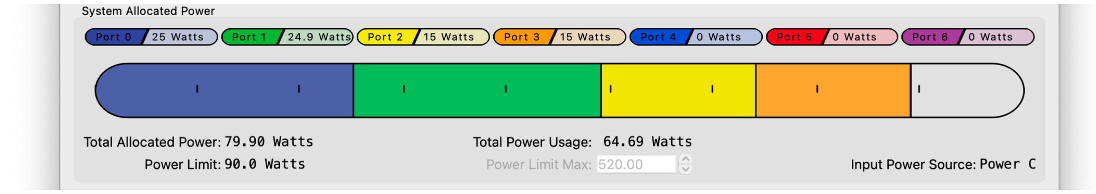


HubTool USBHub3c Power tab

The power tab shows the allocation and actual flow of power through the hub.

System allocated power panel

Across the top of this panel are the allocated currents of each port. Positive currents represent that the port is a power source, negative currents are for sinks.



System allocated power panel

Below the row of ports is a visualization of the stackup of allocated currents.

Total allocated power - sum of the allocated source power

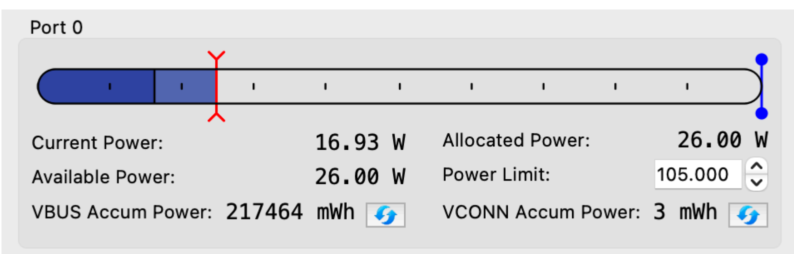
Power limit - maximum power that can be allocated across all ports. Determined by input power minus nominal losses, e.g. 90 W for 100 W supply

Total power usage - sum of actual power being sourced

Power limit max - set power limit for DC input when unregulated power is provided to the DC port

Input power source - input port with the highest power capability

Port power panel



Port power panel

Current power - actual port power output (positive) or input (negative)

Available power - maximum power that could be allocated to the port, not to exceed *Power limit*

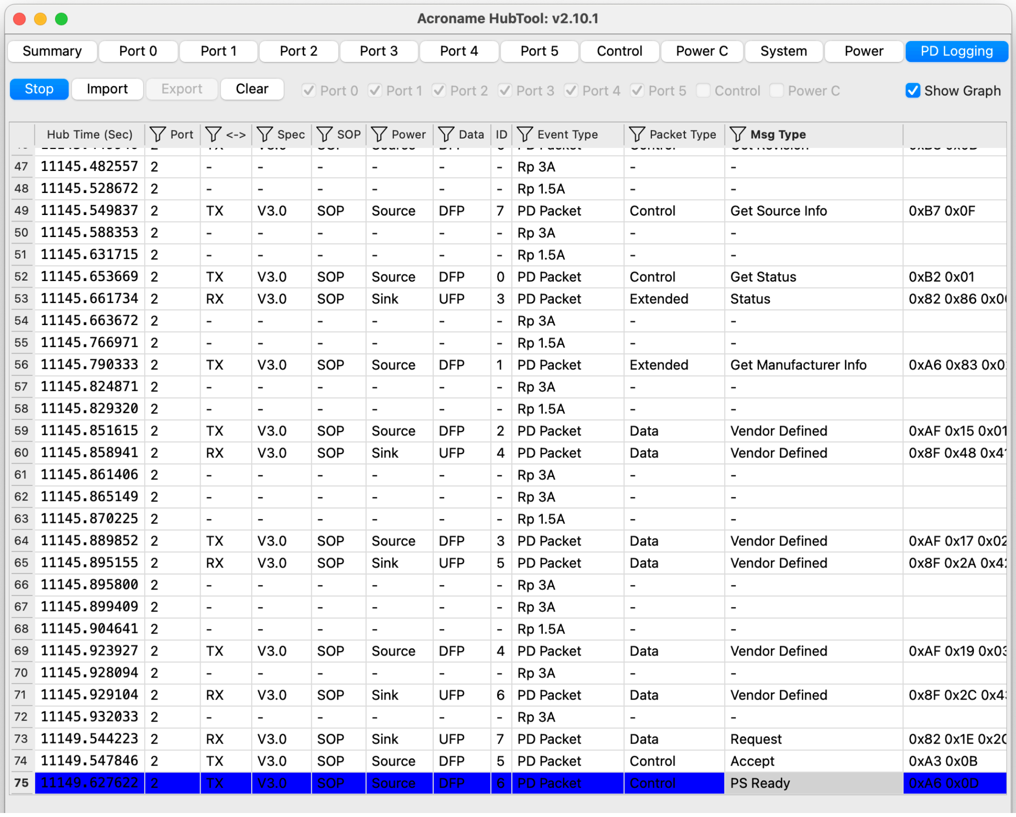
VBus accum power - total energy sunk or sourced on VBus since reset

Allocated power - power allocated to the port, not to exceed *Power limit*

Power limit - power limit for the port, max 105 W

VConn accum power - total energy sunk or sourced on VConn since reset

PD Logging tab (add-on feature)



PD logging tab view

PD logging[55] is a software add-on feature enabling capture, logging, and decoding of USB PD communications on all USB-C ports of the hub, including power negotiations and timing. At the top of the panel are logging controls and port selection toggles.

Stop - stops logging

Import - imports a CSV log file

Export - exports a CSV log file (stop logging before export)

Clear - clears the PD log

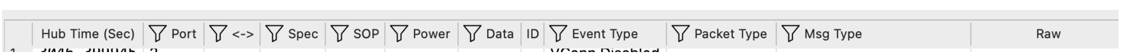
Port toggles - select which ports to monitor (can only change toggles when logging is stopped)

Show graph - when checked, clicking on an event pops up the *Port Plot* graph window and highlights the corresponding PD packet. Clicking on a yellow PD message in the port plot highlights the corresponding message in the log

[55] https://acroname.com/store/t99-pd-log

Graph view with highlighted PD packet

Below these controls is the PD packet log. The left axis is the row number, which resets when the log is cleared. Columns can be filtered by clicking the filter icon ( ) on the column headings:



PD packet log headings

Time - clicking on this heading cycles among time references:

- Hub time (s) - time since the hub powered on
- App time (hh:mm:ss) - time since the HubTool App was launched
- System time (yyyy.MM.dd hh:mm:ss:zzzz) - date and time

Port - ports 0-5, Control, or Power C

<-> - message direction - *RX*, *TX*, or '-' (none)

Spec - USB PD version

SOP* - "Start Of Packet"

- SOP - for messages between source and sink
- SOP' - for messages to the cable connector closest to the downstream-facing port (DFP)
- SOP" - for messages to the cable connector closest to the upstream-facing port (UFP)

Power - *sink*, *source*, *none*

Data - *UFP*, *DFP*, '-'

ID - message ID, (0 – 7), increments with each new message. Acknowledgements should match message ID

Event type - description of the event

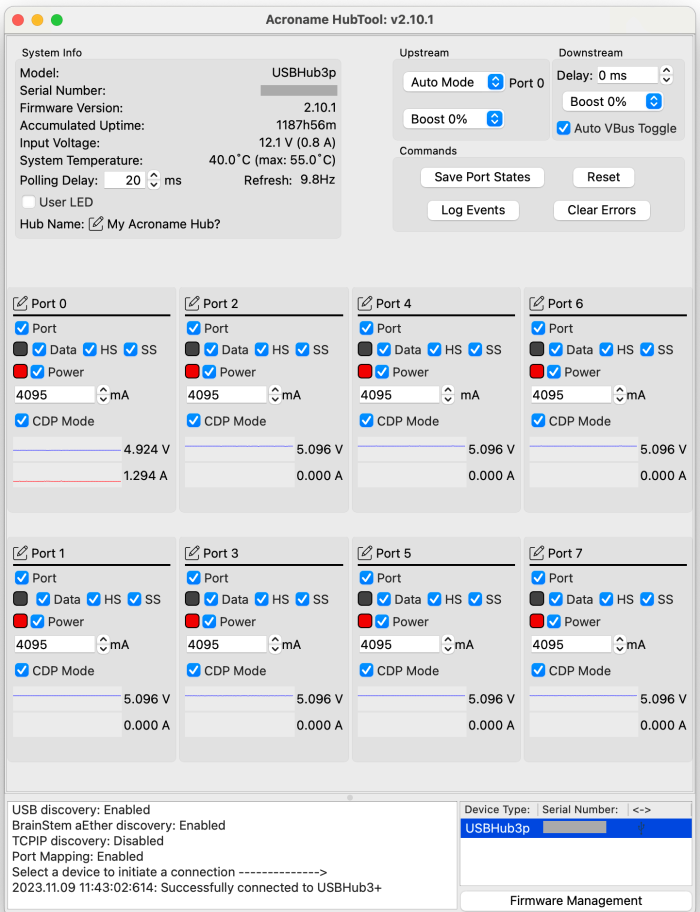
Packet type

- Control - short messages that typically require no data exchange
- Data - messages contain data objects that are transmitted between devices
- Extended - data messages with larger data payloads

Message type - description of message

Raw - raw PD message (hexadecimal)

USBHub3p

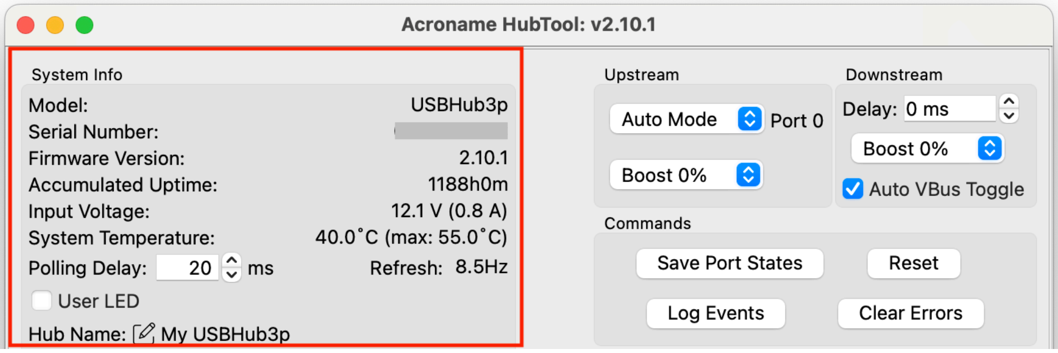


The USBHub3p[56] is a programmable USB hub with 8 full-featured 5 Gbps USB-A ports and 1 downlink port for daisy chaining. Two upstream ports allow automatic or manual host switching.

HubTool presents a unified dashboard to control and view state of USBHub3p.

[56] https://acroname.com/store/programmable-industrial-hub-s79-usbhub-3p

General system information



USBHub3p general system information

The upper-left panel shows **general system information** for the hub:

Serial number

Firmware version

Accumulated uptime

The total time the system has been powered on since leaving the factory

Input voltage (V)

System temperature (°C)

Polling delay (ms)

Sets how long to wait after receiving all information from the hub to poll again. A setting of 0 will start a new set of requests immediately. Polling takes 50-150 ms.

Refresh (Hz)

The measured refresh rate, which is the inverse of the polling delay plus the time to receive the data

User LED toggle

Toggles a blue LED located on the USBHub3p back panel - used for debugging and to identify the hub

Hub name

Editable friendly name for the hub - used by HubTool and ControlRoom

Port settings and commands

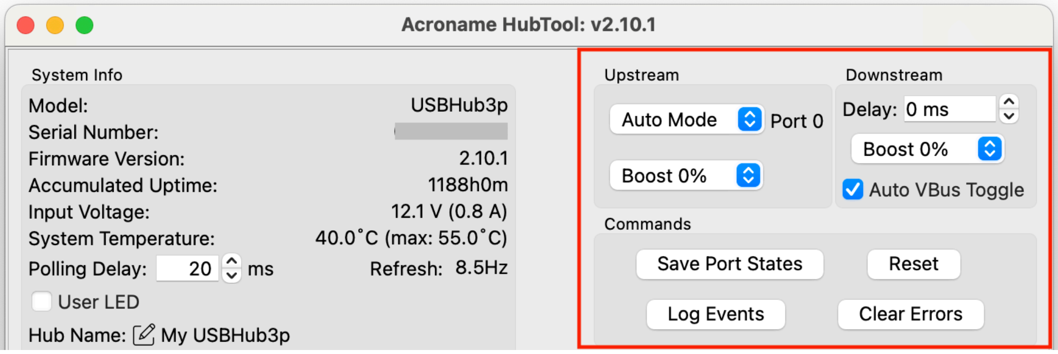
USBHub3p port settings and commands

The upper-right panel shows **port settings** and **commands**:

Upstream (host) port settings:

Host select:

- Auto mode (Port 0 if both ports connected)



- Port 0 only
- Port 1 only
- None (do not connect to either host port)

Note that the Control port (mini USB, located on USBHub3p back panel) is always accessible

Boost

Amplifies the upstream USB signal up to 12 percent to improve marginal connections - default = 0

Downstream port settings:

Delay (ms)

Delays the enumeration of all downstream ports when power is enabled. Useful if devices are slow to power on and don't respond to enumeration in time - default = 0

Boost

Amplifies the downstream USB signals up to 12 percent to improve marginal connections - default = 0

Auto VBus toggle

Toggles downstream port VBus when the upstream connection changes - default = on

Commands

Save port states

Saves the settings of all ports to the hub's internal memory so that they will persist through power cycling and reset. Saved states: **Host select**, **Boost**, **Delay**, **Power and data toggles**, **Current limit**, **CDP mode toggle**

Reset

Resets the hub - VBus toggles and hub returns to previously saved state

Log Events

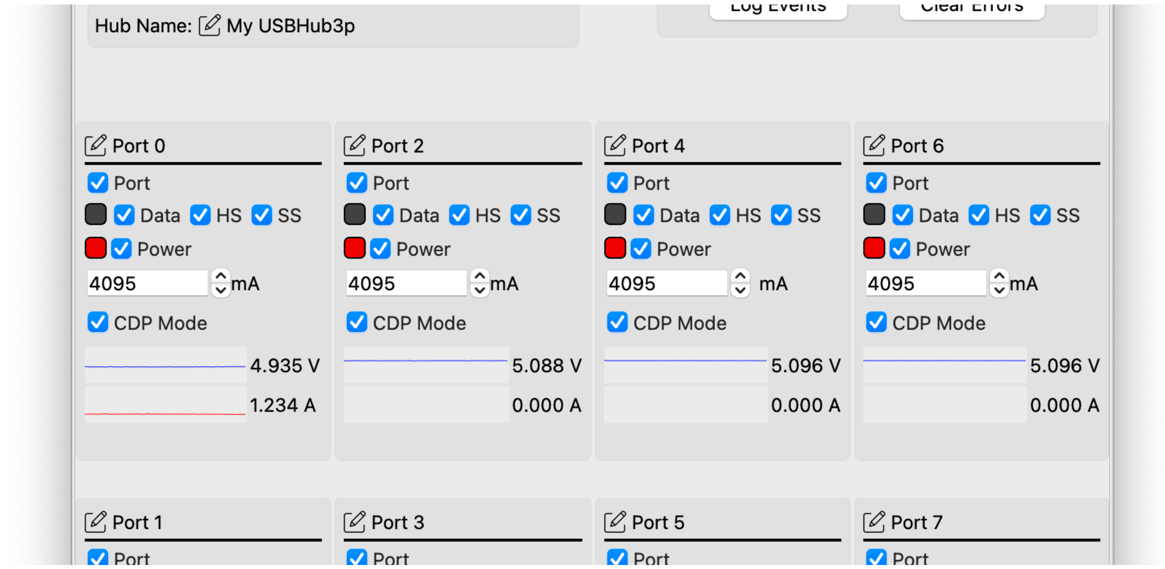
Displays the last 150 log events in the console

Clear errors

Clears internal error bits

Port view

Each of 8 full-featured downstream ports on the front panel of USBHub3p has its own interface panel in HubTool.



Downstream port view

Port name

Editable “friendly” name for the port that is displayed in HubTool and ControlRoom interfaces

Power and data toggles

Toggle name	Enables and disables
Port	Entire port
Data	All Data
HS	USB 2 (High Speed) pins
SS	USB 3 (Super Speed) pins
Power	VBus

Virtual LEDs

These indicate the data and power status of the port and match the real LEDs on the front panel of the hub.

Attribute	Status	LED color
Data rate	USB 2	Yellow
	USB 3	Green
	USB 2, USB 3	Green
VBus	Powered	Red
	Off	Black

Current limit

Maximum current that the hub will supply - default = 4095 mA

CDP mode toggle

Enables Charging Downstream Port modes (default = on). The port will signal the attached device to draw up to a maximum current based on this table:

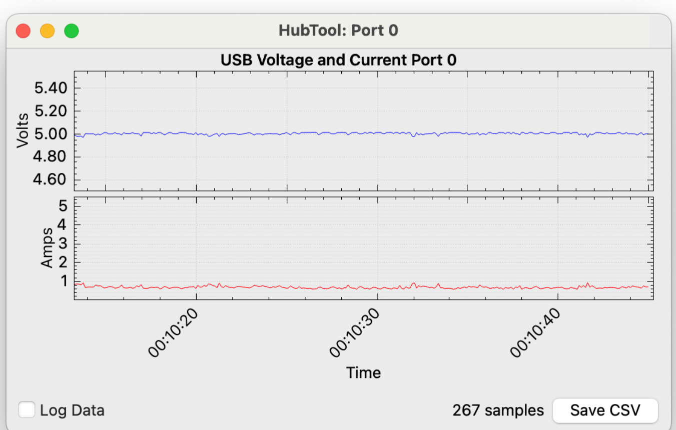
Table 19: *Port power modes*

CDP mode toggle	Condition	Port mode	Maximum current (device limited)*
On	Host present and USB 2 data lines enabled	CDP	1500 mA
	No host present and USB 2 data lines enabled	DCP (Device Charging Port) mode	5000 mA
Off	No host or no USB 2 data lines connected	SDP (Standard Downstream Port)	100 mA
	Host present and USB 2 data lines enabled		500 mA

* The hub limits current to **current limit**, up to a maximum of 4000 mA

Voltage and current display

Shows a graph of the port's bus voltage and current. Clicking on the port graph pops up a window with a larger rolling chart of the last 32 seconds.



Voltage and current strip chart and logging

Decreasing polling delay will increase the number of samples used in the chart. Selecting the "Log Data" toggle switches the chart from rolling to expanding mode. Clicking "Save CSV" saves the data of the graph view in a .CSV file.

Table 20: *Example .CSV file output*

Time (s)	Port 0 Voltage (V)	Port 0 Current (A)
6375.647	5.104	0.000
6375.838	5.104	0.000
6376.035	5.104	0.000
6376.238	5.104	0.000
6376.432	5.096	0.000
...		

Device descriptors

If Options > Port Mapping is selected, when a device is attached to a downstream port, its descriptors will scroll at the bottom of the port panel. Click the carat (^) to expand:

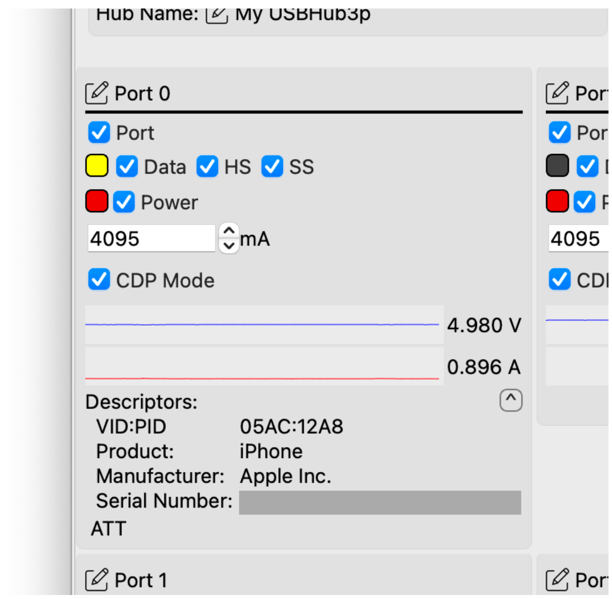


Table 21: *Descriptor table*

Descriptor	Content
VID:PID	16-bit vendor ID and 16-bit product ID
Product	Product name string
Manufacturer	Manufacturer name string
Product serial number	Product serial number
ATT	Indicates device is attached

USBHub2x4

The USBHub2x4[57] is a 4-port software-programmable 480Mbps (USB 2.0) hub designed for demanding environments where advanced control and monitoring of USB ports is required. Two upstream ports allow automatic or manual host switching.

HubTool presents a unified dashboard to control and view state of USBHub2x4.

General system information

USBHub2x4 general system information

The upper-left panel shows **general system information** for the hub:

Serial number

Firmware version

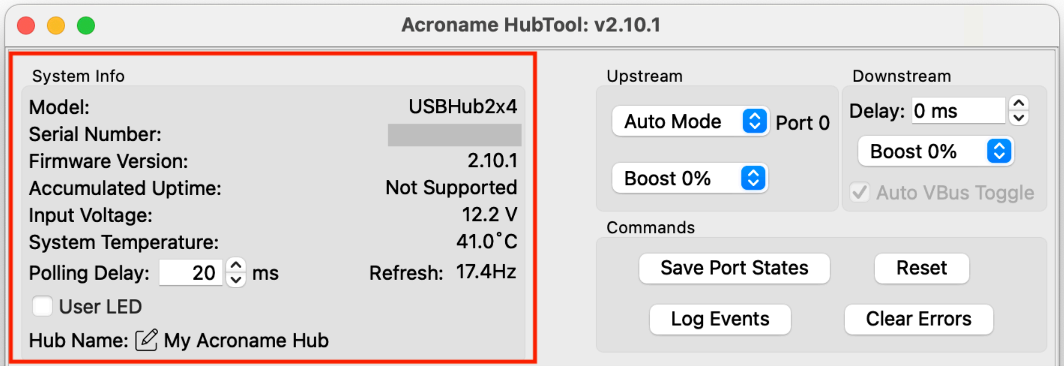
Accumulated uptime

> The total time the system has been powered on since leaving the factory - *unsupported in USBHub2x4*

Input voltage (V)

System temperature (°C)

[57] https://acroname.com/store/industrial-intelligent-4-port-hub-s77-usbhub-2x4



Polling delay (ms)

Sets how long to wait after receiving all information from the hub to poll again. A setting of 0 will start a new set of requests immediately. Polling takes 50-150 ms.

Refresh (Hz)

The measured refresh rate, which is the inverse of the polling delay plus the time to receive the data

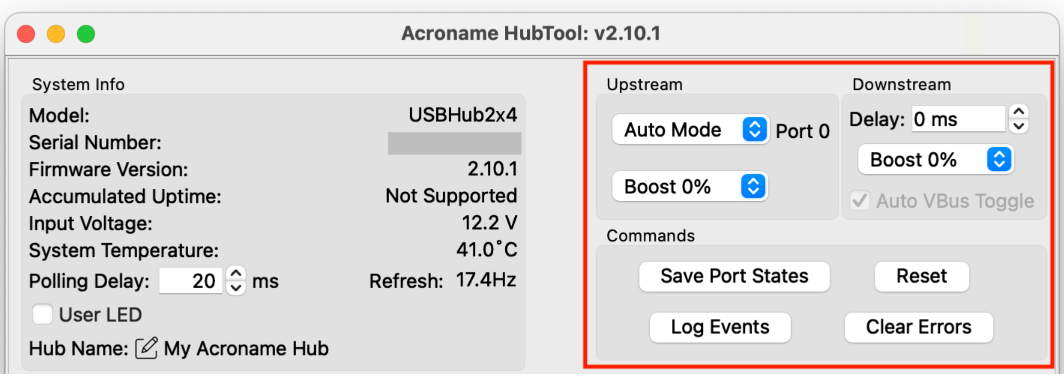
User LED toggle

Toggles a blue LED (7th from the top) visible through the slot in the cover of USBHub2x4 - used for debugging and to identify the hub.

Hub name

Editable friendly name for the hub - used by HubTool and ControlRoom

Port settings and commands



USBHub2x4 port settings and commands

The upper-right panel shows **port settings** and **commands**:

Upstream (host) port settings:

Host select:

- Auto mode (Port 0 if both ports connected, defaults to auto after power cycle)
- Port 0 only

- Port 1 only
- None (do not connect to either host port - *no effect in USBHub2x4*)

Boost

Amplifies the upstream USB signal up to 12 percent to improve marginal connections - default = 0

Downstream port settings:

Delay (ms)

Delays the enumeration of all downstream ports when power is enabled. Useful if devices are slow to power on and don't respond to enumeration in time - default = 0

Boost

Amplifies the downstream USB signals up to 12 percent to improve marginal connections - default = 0

Auto VBus toggle

Toggles downstream port VBus when the upstream connection changes - *always on for USBHub2x4*

Commands

Save port states

Saves the settings of all ports to the hub's internal memory so that they will persist through power cycling and reset. Saved states: **Boost**, **Delay**, **Power and data toggles**, **Current limit**, **CDP mode toggle**

Reset

Resets the hub - VBus toggles and hub returns to previously saved state

Log Events

Displays the last 150 log events in the console

Clear errors

Clears internal error bits

Port view

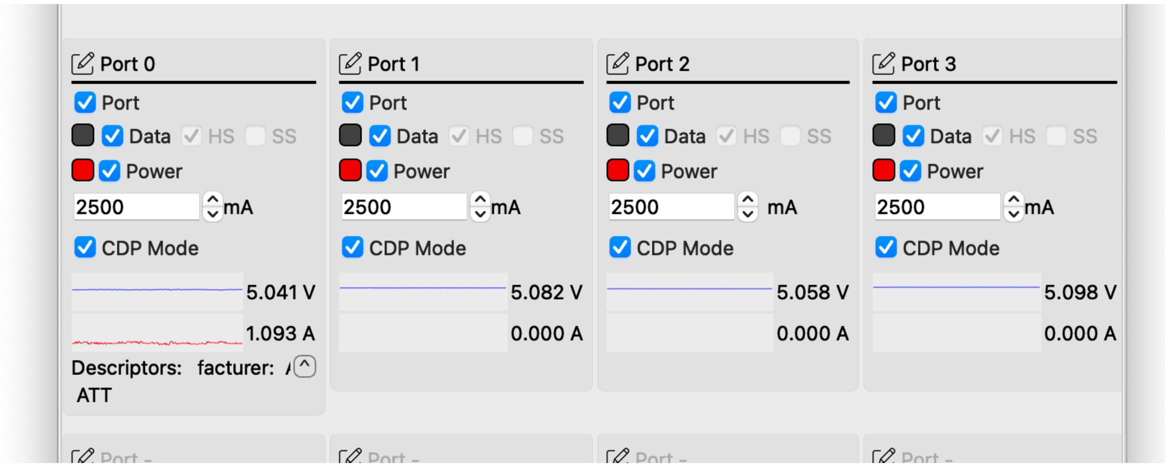
Each of 4 USB 2.0 downstream ports on the front panel of USBHub2x4 has its own interface panel in HubTool. A second row of 4 more ports are grayed out and unavailable with USBHub2x4.

Downstream port view

Port name

Editable "friendly" name for the port that is displayed in HubTool and ControlRoom interfaces

Power and data toggles



Toggle name	Enables and disables
Port	Entire port
Data	All Data
HS (grayed out, checked)	USB 2 (High Speed) pins - *always enabled*
SS (grayed out, unchecked)	USB 3 (Super Speed) pins - *not available on USBHub2x4*
Power	VBus

Current limit

Maximum current that the hub will supply - default = 2500 mA

CDP mode toggle

Enables Charging Downstream Port modes (default = on). The port will signal the attached device to draw up to a maximum current based on this table:

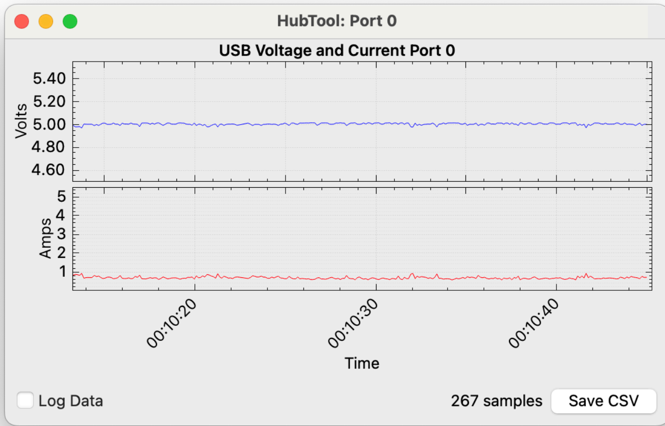
Table 22: *Port power modes*

CDP mode toggle	Condition	Port mode	Maximum current (device limited)*
On	Host present and USB 2 data lines enabled	CDP (Charging Downstream Port)	1500 mA
Off	No host or no USB 2 data lines connected	SDP (Standard Downstream Port)	100 mA
	Host present and USB 2 data lines enabled		500 mA

* The hub limits current to **current limit**, up to a maximum of 2500 mA

Voltage and current display

Shows a graph of the port's bus voltage and current. Clicking on the port graph pops up a window with a larger rolling chart of the last 32 seconds.



Voltage and current strip chart and logging

Decreasing polling delay will increase the number of samples used in the chart. Selecting the “Log Data” toggle switches the chart from rolling to expanding mode. Clicking “Save CSV” saves the data of the graph view in a .CSV file.

Table 23: *Example .CSV file output*

Time (s)	Port 0 Voltage (V)	Port 0 Current (A)
6375.647	5.104	0.000
6375.838	5.104	0.000
6376.035	5.104	0.000
6376.238	5.104	0.000
6376.432	5.096	0.000
...		

Device descriptors

If Options > Port Mapping is selected, when a device is attached to a downstream port, its descriptors will scroll at the bottom of the port panel. Click the carat (^) to expand:

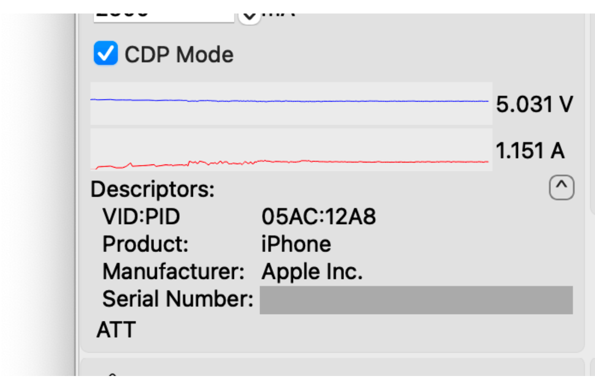
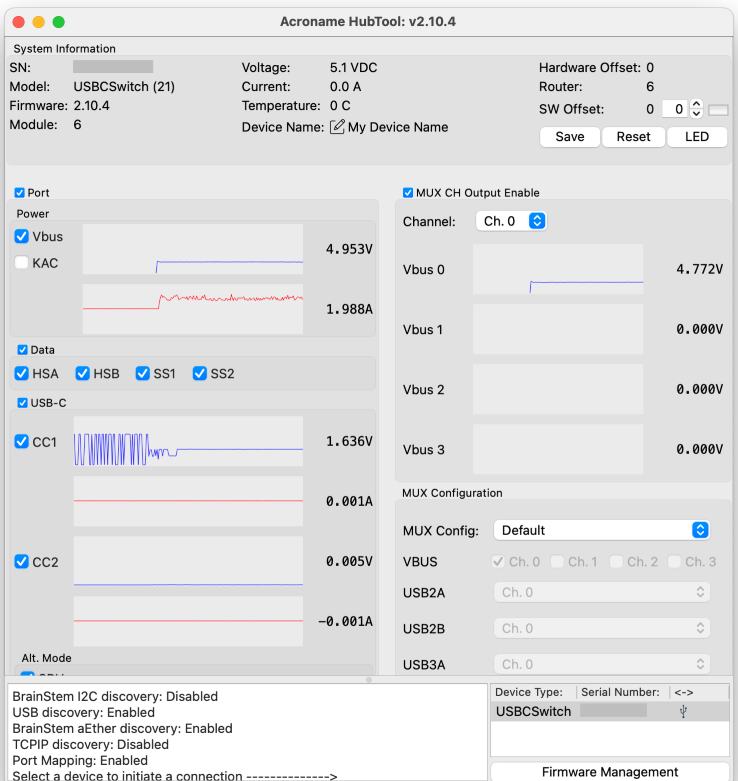


Table 24: *Descriptor table*

Descriptor	Content
VID:PID	16-bit vendor ID and 16-bit product ID
Product	Product name string
Manufacturer	Manufacturer name string
Product serial number	Product serial number
ATT	Indicates device is attached

USB-C-Switch



[58] *HubTool interface for USB-C-Switch*

Acroname's USB-C-Switch[59] is an industrial USB-C switch able to connect one of up to four devices to a host, or one device to one of up to four hosts. It is not a hub – the selected ports form a bidirectional connection and appear "like a cable" to connected devices, even supporting USB alt modes like DisplayPort.

When combined with an Acroname Universal Orientation Cable[60] , the USB-C-Switch can emulate a cable flip, allowing tests of all the connections on a port without the need to manually flip the cable.

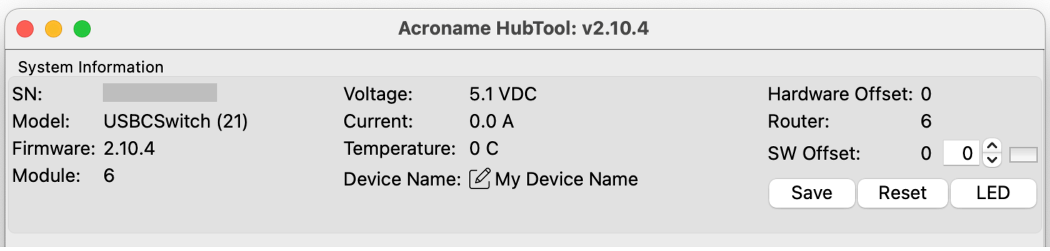
HubTool presents a unified dashboard to control and view the state of USB-C-Switch.

[58] https://acroname.com/store/programmable-industrial-switch-s85-rdvr-usbcsw
[59] https://acroname.com/store/programmable-industrial-switch-s85-rdvr-usbcsw
[60] https://acroname.com/store/c67-usbc-uoc

Connections

Port	Connects to		
Control	Control computer running HubTool		
Common	Common host	or	Common device
Ports 0 – 3	Switched devices		Switched hosts

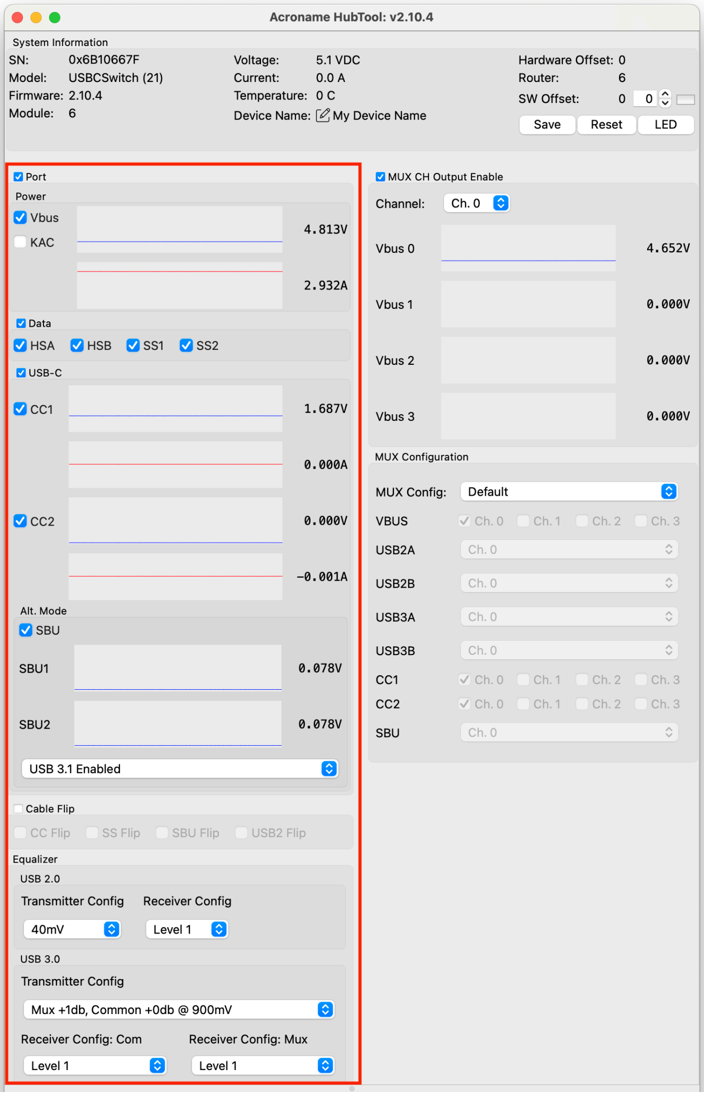
System information



The System Information panel contains information and settings for the switch.

Heading	Meaning
SN	Switch serial number
Model	Switch model name and number, e.g. USBCSwitch (21)
Firmware	Currently installed firmware version
Module	Address of the module on the Brainstem network
Voltage	Voltage at the input power source port
Current	Input current at the Control port
Temperature	System temperature
Hardware offset	Increments Brainstem module address by a hardware setting - fixed to 0 in USBC-Switch
Router	Address of the routing Brainstem module
Software offset	Increment the module address. Requires Save and Reset
Save	Store select changed settings
Reset	Soft reset of the switch
LED	Toggles blue user LED next to the control port, used for debugging and to identify the switch

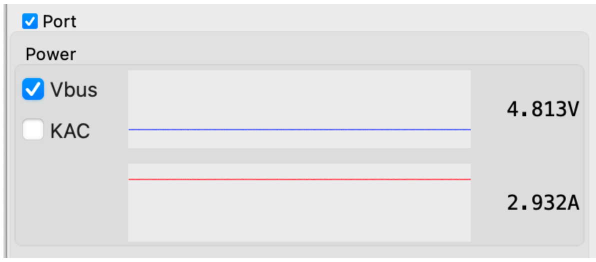
Common port controls



The left side of HubTool shows controls and attributes related to:

- Enabling and disabling individual lines
- Viewing voltage and current on VBus, CC1, CC2, SBU1, SBU2
- Keep-alive charging
- Alt mode
- Cable flip
- Equalization (redriver version only)

Power



- **Port toggle** – enables and disables all lines connecting the Common port to the selected mux channel
- **VBus toggle** – enables and disables VBus lines
- **KAC toggle** – enables Keep-alive charging (KAC)

  Keep-alive charging helps keep battery-powered devices on the non-selected mux ports charged. When enabled, the KAC circuit connects power from the control port VBus to all non-selected mux channel VBus lines. See the API reference[61] for more detail
- **VBus voltage and current plots** – shows voltage and current for VBus. Click to pop up the *Voltage and Current plot window*

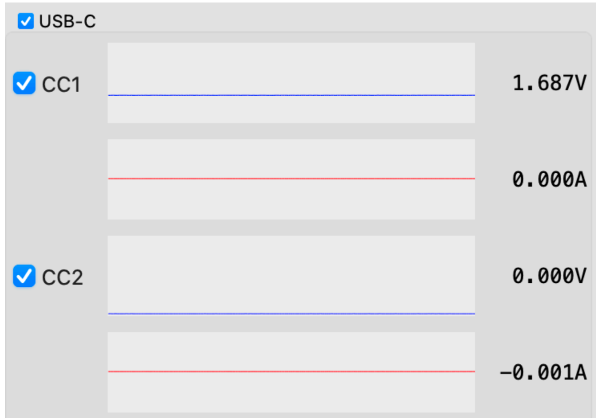
Data line toggles

- **Data toggle** – enables and disables all USB data lines
- **HSA, HSB, SSA, SSB toggles** – enable and disable USB 2 (HS) and USB 3 (SS) data lines on side A or side B independently

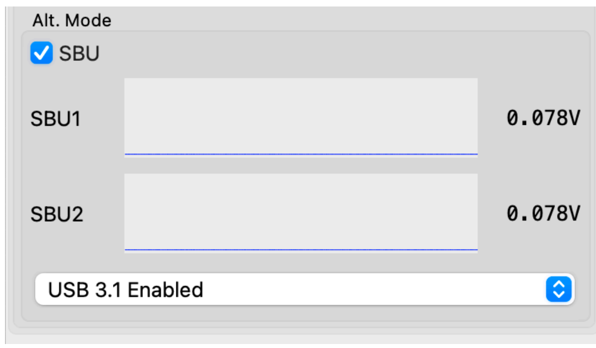
CC

- **USB-C toggle** – enables and disables CC and SBU lines
- **CC toggles** – enable and disables CC1 and CC2 lines independently
- **CC1 and CC2 voltage and current plots** – shows voltage and current for CC1 and CC2. Click to pop up the *Voltage and Current plot window*

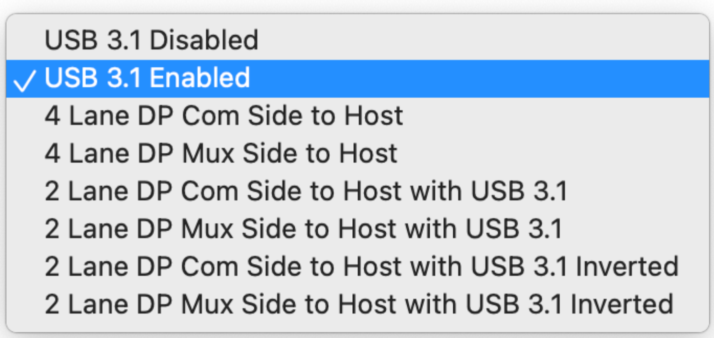
[61] https://acroname.com/reference/devices/usbcswitch/functionality.html#keep-alive-charging-kac



Alt mode



- **SBU toggle** – enables and disables SBU lines. Used for Alternate Mode discovery, negotiation, and configuration data exchange
- **SBU1 and SBU2 voltage and current plots (redriver model only)** – shows voltage and current for SBU1 and SBU2. Click to pop up the *Voltage and Current plot window*
- **DisplayPort alt mode configuration menu (redriver model only)**



Alt Mode Configuration
USB 3.1 Disabled – no SS lines connected
USB 3.1 Enabled – SS lines connected
4-Lane DisplayPort – no USB 3.1 – Host on Common Port
4-Lane DisplayPort – no USB 3.1 – Host on Mux Port
2-Lane DisplayPort + USB 3.1 –- Host on Common Port
2-Lane DisplayPort + USB 3.1 –- Host on Mux Port
2-Lane DisplayPort + USB 3.1 Inverted – Host on Common Port
2-Lane DisplayPort + USB 3.1 Inverted – Host on Mux Port

DisplayPort Alt modes use the SS lines for DisplayPort data, but change their direction. Since each redriver line can operate in only one direction at a time, the redriver direction needs to be set to match the mode being used by the host and display. See the API reference[62] for more detail.

Cable Flip

- **Cable flip toggle** – switches USB data, VConn, and SBU lines from side A to side B as if the cable had been flipped. When using standard USB-C cables with USB-C-Switch, one cable orientation will work, the other will need to be flipped physically, or by using the toggle.
- **CC, SS, SBU, USB2 flip toggles** – individually flip each connection type

To enable automated cable flips for testing, use one Universal Orientation Cable[63] for either the host or mux connection, and one standard cable for the other connection.

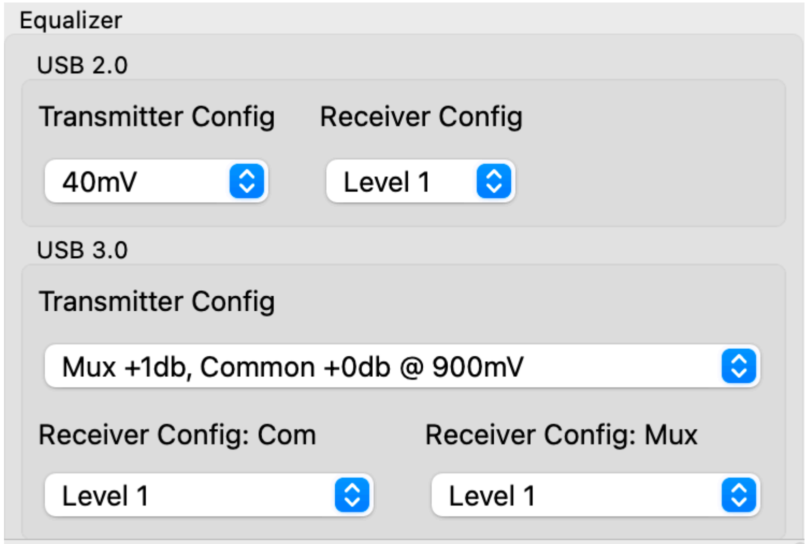
Equalizer (requires redriver model)

The equalizer section provides controls to set transmitter and receiver gains for USB 2 and USB 3 data lines in both directions.

USB 2.0

- **Transmitter config** – selects the amount of DC boost applied to USB 2.0 (HS) signals. USB Low Speed and Full Speed signals are unaffected. When boost is set to O mV, the HS redriver is disabled independent of receiver configuration

[62] https://acroname.com/reference/devices/usbcswitch/entities/usb_entity.html#alt-mode-configuration-redriver-only
[63] https://acroname.com/store/c44-usb-uoc



USB 2.0 transmitter DC boost
40 mV – default
60 mV
80 mV
0 mV (redriver disabled)

- **Receiver config** – controls the sensitivity of the redriver to incoming HS signals by boosting higher frequencies to make sharper edges. When receiver config is set to level 0, the redriver is disabled independent of transmitter configuration

USB 2.0 Receiver Equalization
Level 1 – moderate boost – default
Level 2 – higher boost
Level 0 – redriver disabled

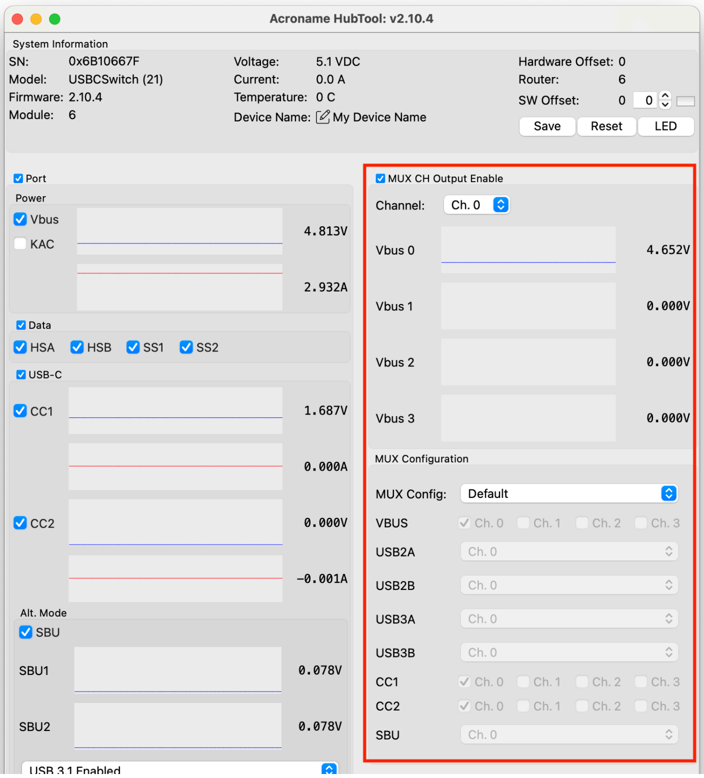
USB 3.0

- **Transmitter config** – selects preset combinations of transmitter gains for each direction of the full-duplex USB 3 (SS) data lines, and peak-to-peak voltage for both directions

Mux Side	Common Side	Range	
1 db	0 db	900 mVpp	default
0 db	1 db	900 mVpp	
1 db	1 db	900 mVpp	
0 db	0 db	900 mVpp	
0 db	0 db	1100 mVpp	
1 db	0 db	1100 mVpp	
0 db	1 db	1100 mVpp	
2 db	2 db	1100 mVpp	
0 db	0 db	1300 mVpp	

- **Receiver config: com** – sets the sensitivity of the redriver to incoming SS signals on the com side by boosting higher frequencies to make sharper edges. Level 1 (lowest boost, default) to Level 16 (highest)
- **Receiver config: mux** – sets the sensitivity of the redriver to incoming SS signals on the mux side by boosting higher frequencies to make sharper edges. Level 1 (lowest boost, default) to Level 16 (highest boost)

Mux port controls



The right side of HubTool shows controls and attributes related to the mux ports.

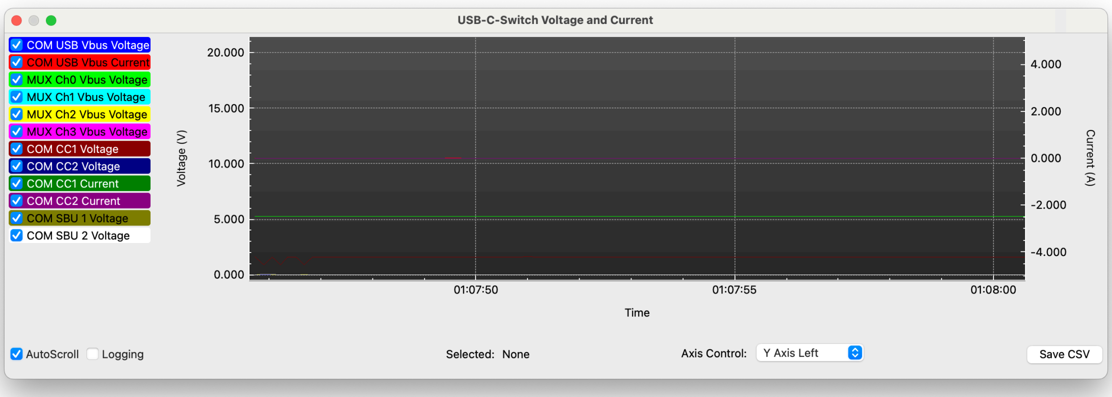
Mux channel control

- **Mux channel output enable** – toggles the output of the selected port
- **Mux channel selector** – designates one mux port to connect to the Common port. Unavailable in Channel priority and Split configuration
- **VBus voltage plots** – shows VBus voltage for each mux port. Click the graph to pop up the *Voltage and Current plot window*

Mux configuration

- **Default** – switches all enabled USB-C lines to the single mux port designated by the channel selector
- **Channel priority** – auto-selects the lowest-numbered mux port that has VBus present. Allows simple automatic host selection (requires USB A-to-C cables)
- **Split** – allows each signal type to be independently connected to a mux port. VBus and CC lines can be assigned to any combination of ports, while USB2, USB3, and SBU can be assigned to a single mux port. See the API reference[64] for more detail

Voltage and current plot window

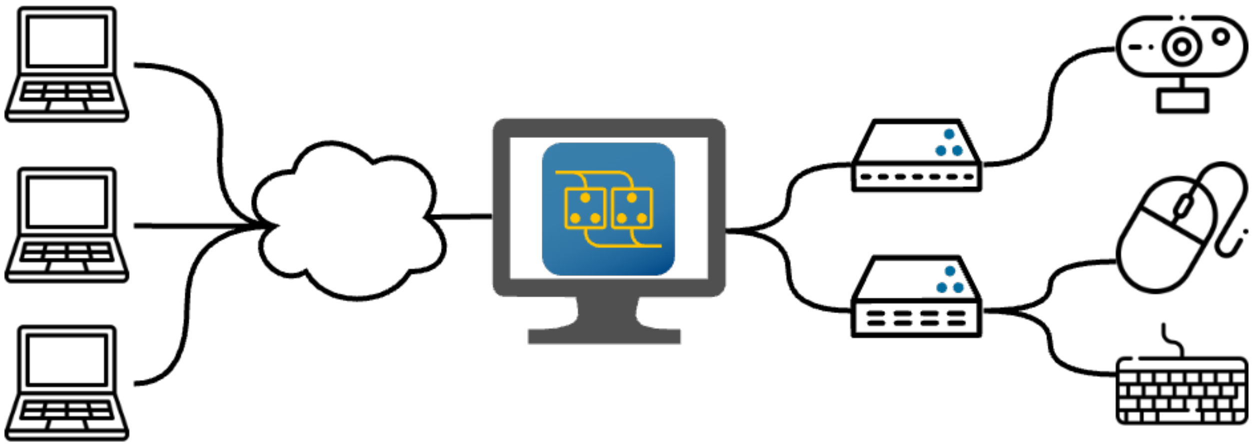


The voltage and current plot window pops up when any of the traces in the main window are clicked.

- **Trace select toggles** – show and hide each trace
- **Autoscroll** – scrolls automatically. When enabled, the right edge is "now" – default = on
- **Logging** – when enabled, data is stored beyond the current plot view
- **Axis control** – sets whether dragging and scrolling zoom affect the voltage, current, or time axis
- **Save CSV** – saves data for each non-hidden trace as an independent CSV file. If logging is not enabled, only the data visible in the graph is saved

[64] https://acroname.com/reference/devices/usbcswitch/functionality.html#mux-split-mode

2.2 BrainD



BrainD is a server application that provides simultaneous multi-client access and control to Acroname devices connected to a host computer. Connected devices can be accessed without drivers via any intranet or internet connection, including tunneling over VPNs. Using a standard HTTP request interface, BrainD serves a RESTFul API that provides telemetry and status information on the entire USB tree. BrainD builds upon decades of experience from successfully deploying complex test and measurement systems, conference room solutions, and artificially intelligent robotic applications.

2.2.1 Installation

Procedure

Please ensure you have appropriate privileges on your computer and back up any critical data to prevent data loss during the upgrade process.

1. Download latest software release.
1. Navigate to the Acroname Download Center[65].
2. Choose the appropriate software for your operating system and click the link to initiate the download.
3. If prompted to choose a download location, select a place that is easily accessible.
4. Wait for the download to complete.
2. Installation of BrainD
    1. Windows:
    1. The downloaded file will be a setup .exe file. Open the .exe file to run the program installer.
    2. If BrainD is already installed, the installer will prompt to uninstall the existing version.
    3. Follow the installer prompts to accept the license and select the application installation directory.
    4. Open BrainD from the Start Menu or Windows Explorer.
    2. macOS

[65] https://acroname.com/software/braind

1. The downloaded file will be a .dmg disk image. Open the .dmg file to mount the disk image.
2. Drag the BrainD application into the Applications folder inside the disk image
3. Open BrainD from the Applications folder.

3. Linux

   1. The downloaded file will be a .deb installer package.
   2. In a command line, run `sudo dpkg -i <image file>.deb`. This will install the BrainD application.

   Note: If dependencies are missing, the above command may fail. Run `sudo apt-get -f install` to install the dependencies, then re-run the previous install command.

   3. After installation, open BrainD from the applications selector or from the command line.

Note: On Linux, the `api/v1/devices` REST endpoint may not enumerate all devices on the system if the user does not have write permissions to `/dev/bus/usb`.

Software Requirements

Web Browsers

The following web browsers are supported for accessing and using the web-based application:

Table 25: Supported Web Browsers

Browser	Version	Windows	macOS	Linux	iOS	Android
Safari	Latest stable	Yes	Yes	Yes	Yes	No
Brave	Latest stable	Yes	Yes	Yes	Yes	Yes
Microsoft Edge	Latest stable	Yes	Yes	No	No	No
Google Chrome	Latest stable	Yes	Yes	Yes	Yes	Yes
Mozilla Firefox	Latest stable	Yes	Yes	Yes	Yes	Yes

Operating Systems

BrainD application is compatible with the following operating systems:

Table 26: Supported Operating Systems

Operating System	Version	BrainD	ControlRoom
Windows (x86/x86_64)	10	Yes	Yes
	11	Yes	Yes
macOS (Intel/Apple Silicon)	10.15	Yes	Yes
	11.x	Yes	Yes
	12.x	Yes	Yes
	13.x	Yes	Yes
Linux (x86_64)	Ubuntu 20.04 LTS	Yes	Yes
	Ubuntu 22.04 LTS	Yes	Yes
Linux (arm64)	Ubuntu 20.04 LTS	Yes	Yes
Linux (armhf)	Debian Bullseye	Yes	Yes

Please note that while our web-based application may function on earlier versions of the mentioned operating systems, we strongly recommend using the specified minimum versions for optimal performance, security, and compatibility.

2.2.2 Quick Start Guide

To get started with BrainD, follow these steps:

1. Install BrainD following the *Installation Guide*.
2. Connect an Acroname product with a valid Software License installed. For this guide, a *USBHub3c* will be used, with serial number `F7A9DFC6`.
3. Start BrainD by selecting it in the Start Menu under "Acroname" (Windows) or from the Applications directory (Mac). An icon will appear in the system taskbar.
4. The BrainD service is now running. The *RESTful API* is available at `http://127.0.0.1:9005`. See the following sections for example usage. Additional examples are in the documentation for each RESTful endpoint.

GET Example - Get Version

The version of software that is running can be read by issuing a GET request to the `version` endpoint.

Bash

```
curl -H 'Accept: application/json' http://127.0.0.1:9005/api/v1/version
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/version')
json_data = response.json()
print(json.dumps(json_data, indent=3))
```

The resulting return of the version will return the following JSON body.

```
{
    "braind": {
        "version": {
            "major": 1,
            "minor": 0,
            "patch": 0
        },
        "buildDate": "Thu Aug 24 13:58:24 2023",
        "buildHash": "d38f0a1dd5f8daa2c8cd0a0170fee8210d1356b4"
    },
    "brainstem": {
        "version": {
            "major": 2,
            "minor": 10,
            "patch": 0
        },
        "buildDate": "2023-09-08T19:03:10Z",
        "buildHash": "7af9c07e44601d6987fe3dab0ea201a13609683e"
    }
}
```


PUT Example - Set BrainStem Port Off

BrainD is designed to manage and manipulate Acroname products through the *RESTful API*

Bash

```
curl -X PUT http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/1/enabled -H 'Content-Type: application/json' -d '{"value": 0}'
```

Python

```python
import requests
import json
response = requests.put('http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/1/enabled', json={'value': 0})
json_data = response.json()
print(json.dumps(json_data, indent=3))
```

The resulting return of the version will return the following JSON body.


```
{
    "timestamp": "2023-09-18T04:43:09.813Z",
    "request": {
        "endpointName": "/api/v1/brainstem/F7A9DFC6/port/1/enabled",
        "parameters": {
            "value": 0
        }
    },
    "response": {}
}
```

GET Example - Get BrainStem Port 2 Vbus Voltage

Bash

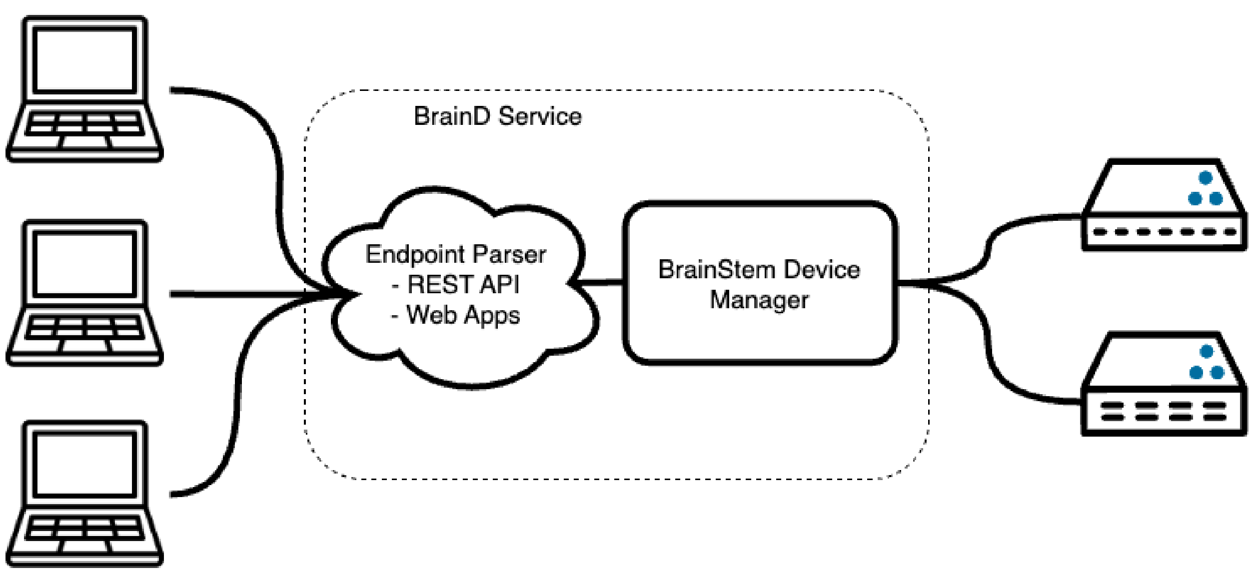
```
curl http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/2/VbusVoltage
```

Python

```
import requests
import json
response = requests.put('http://127.0.0.1:9005/api/v1/brainstem/F7A9DFC6/port/2/
↪VbusVoltage')
json_data = response.json()
print(json.dumps(json_data, indent=3))
```

```
{
    "timestamp": "2023-09-18T04:56:16.639Z",
    "request": {
        "endpointName": "/api/v1/brainstem/F7A9DFC6/port/2/VbusVoltage",
        "parameters": {}
    },
    "response": {
        "value": 5100097,
        "rawValue": 5100097
    }
}
```

2.2.3 Usage



BrainD has multiple methods of interface and interacting with a consumer or client. A host processor must host the BrainD service for endpoints and user applications to run. The BrainD service will inherently manage all links to BrainStem devices that are connected to the host processor with the use of *aEther*. Also, BrainD service creates specific *RESTful endpoints* for connected devices, bulk data endpoints, and web based application endpoints.

Background Service

BrainD user interface application enables simple configuration and manipulation of the application service.

Taskbar

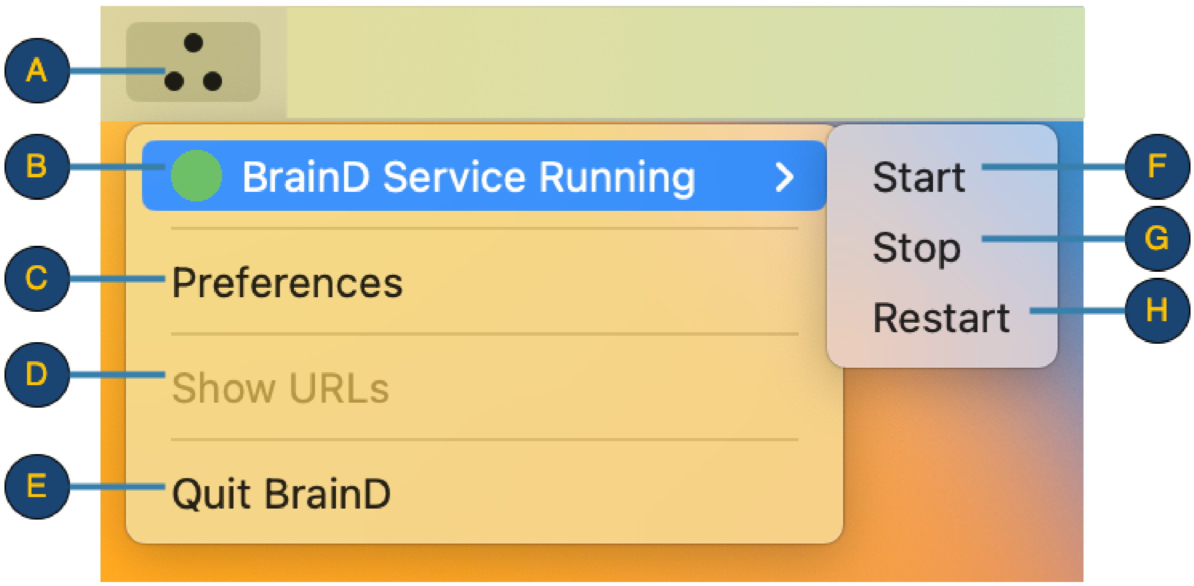


Table 27: Taskbar Details

Item	Description
A	Application icon.
B	BrainD operation status. A green indicator shows the service is running. A *red* indication shows the service is stopped and no application connections or updates will be available.
D	Opens the *settings and configuration dialog*.
E	Close and quit the BrainD service.
F	Start the BrainD service.
G	Stop the BrainD service.
H	Restart the BrainD service. All active connections will be terminated.

Command Line

For headless servers, starting as a system service, or as part of a larger solution, BrainD may be directly run from a command line, without the use of the taskbar GUI application.

The list of command line arguments may be found with the `--help` argument:

```
$ braind_service --help
```

```
BrainD Version 1.0.0
Service providing a REST interface to Acroname BrainStem devices

Application Usage:
```

(continues on next page)

(continued from previous page)

```
braind_service [-f <filename>]... [-d <json object>]

Parameters:
-h, --help                - Show full help message
-v, --version             - Show version information
-f, --config-file <file>  - Specifies any additional configuration files
                            that override the default setting. May be
                            specified multiple times for multiple files.
-d, --config-data <json>  - JSON Object containing configuration to
                            override the default settings
```

The `-f` flag is used to specify an additional configuration file that will be read, and this may be specified multiple times.

The `-d` flag allows a JSON configuration object to be passed directly to the service, which will take priority over all configuration files.

The details of the configuration is documented in *Configuration*.

Examples

For endpoint usage examples, please refer to the *RESTful API* section which will have additional information.

2.2.4 Security

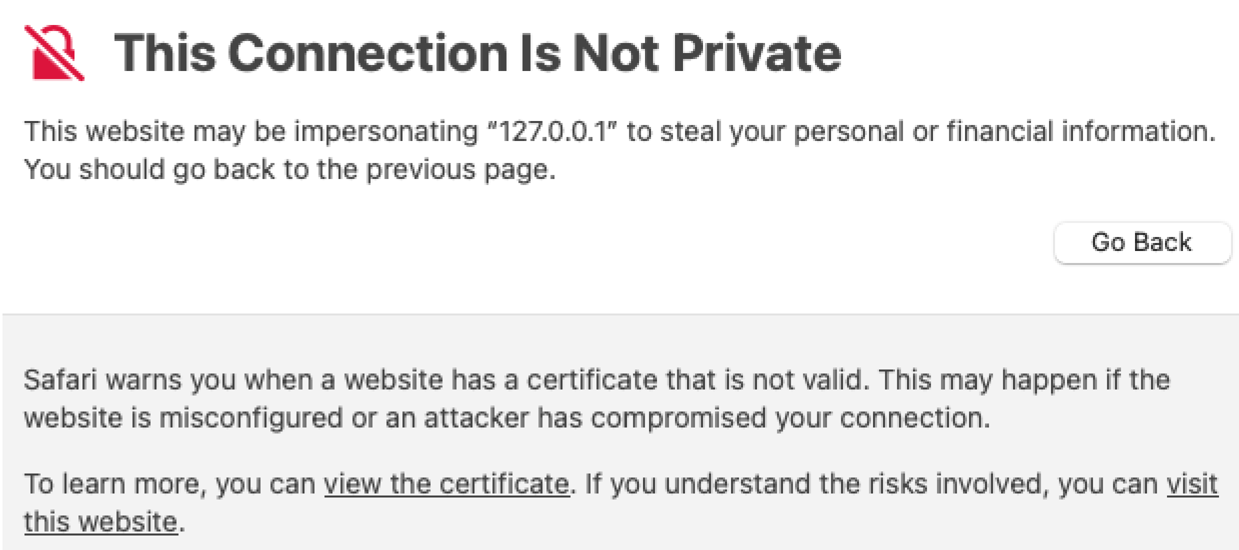
SSL Certificates

SSL (Secure Sockets Layer) is a fundamental technology for securing web applications by encrypting data, ensuring data integrity, authenticating servers, and enhancing user trust.

SSL layers encrypt the data transmitted between a user's web browser and server. This encryption ensures that the data remains unreadable and secure even if intercepted by malicious actors. SSL includes mechanisms to ensure data integrity during transmission using cryptographic hash functions to detect unauthorized modifications or tampering. In the event of altered data during transit, the recipient can see and reject the modified data.

BrainD allows loading a user's SSL certificate through the *settings and configuration dialog*. Many external service providers can provide an SSL certificate; one may already have an SSL certificate for BrainD installations.

Use of a self-signed certificate may cause a warning to appear when accessing the REST endpoints. This is caused by the browser failing to authenticate the received certificate against a trusted certificate authority.



OpenSSL is an option that one can use to create a self-signed SSL certificate. The following command may be used to generate a self-signed certificate:

Bash

```bash
openssl req -new -x509 -days 365 -nodes -keyout example.key -out example.crt
```

By default, BrainD will serve an insecure web page over HTTP, without any certificates or encryption. If insecure communications over HTTP are not desirable, the *settings and configuration dialog* can be used to enable HTTPS/SSL communication. If a certificate is not specified, then BrainD will automatically generate a self-signed certificate.

Certificates are stored in the following directory:

1. Windows: `%AppData%\Acroname\BrainD\config`
2. Mac: `~/Library/Application\ Support/Acroname/BrainD/config`
3. Linux: `~/.acroname/BrainD/config`

2.2.5 Configuration

Preferences

To change the default settings of BrainD, the desktop application provides a simple dialog window that enables users to change the HTTP server settings and specify *SSL/TLS Certificates*. To open the dialog, click on the taskbar menu and select "Preferences".

The following diagram lists the available configuration options in the Settings dialog:

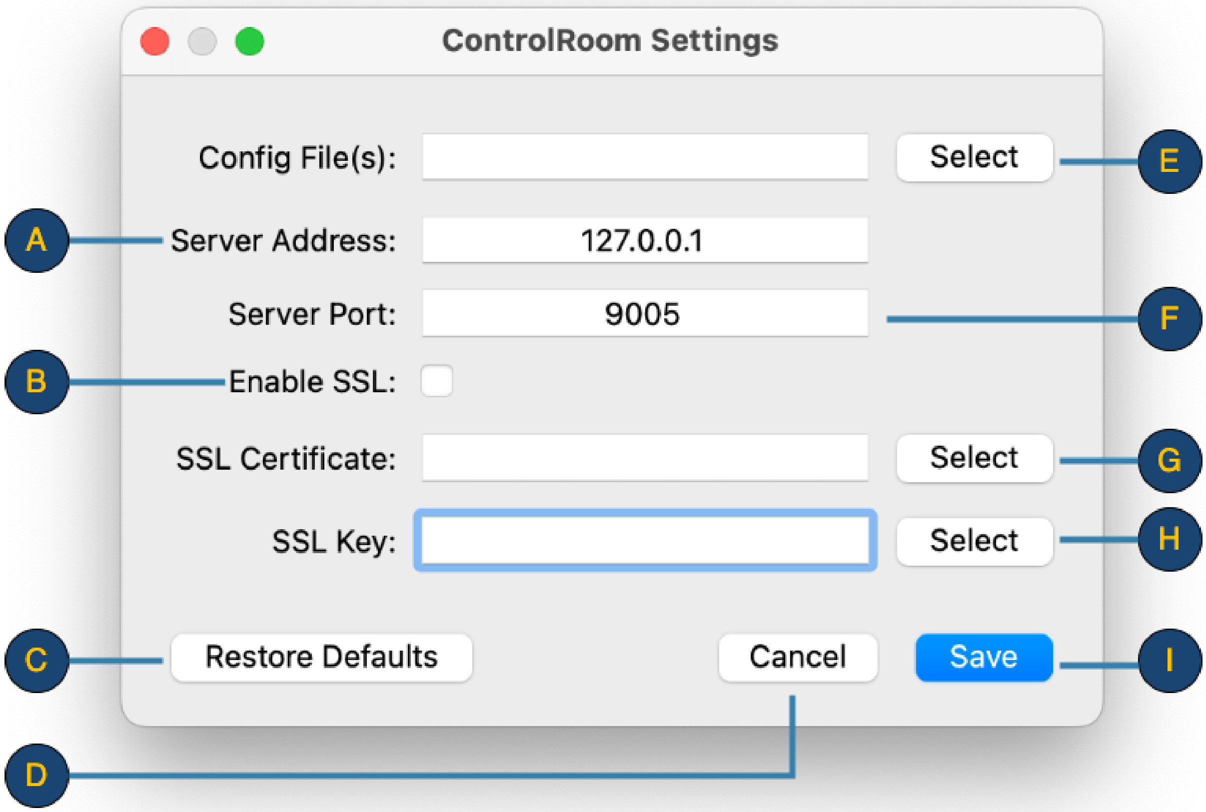


Table 28: Preferences Details

Item	Description
A	Default server address. Default is 127.0.0.1, which will allow connection from external devices.
B	*Enable SSL/TLS* on client connections.
C	Restore configuration settings to factory defaults.
D	Close the preferences window without saving.
E	Optional path to a JSON Configuration File.
F	Server port to bind to for clients. Default is 9005.
G	*SSL/TLS Certificate (.crt) file*. If not specified, a self-signed cert will be generated.
H	*SSL/TLS Key (.key) file*. If not specified, a self-signed cert will be generated.
I	Save application settings.

Configuration Files

BrainD allows for more fine-grained control of the backend service through the use of JSON configuration files. This configuration includes settings for *logging*, polling rates, and web server configuration.

The BrainD service will read the following configuration file for the current operating system:

1. Windows: `%AppData%\Acroname\BrainD\config\config.json`
2. Mac: `~/Library/Application Support/Acroname/BrainD/config/config.json`
3. Linux: `~/.acroname/BrainD/config/config.json`

Each entry in the `config.json` file will be merged into the default configuration object, overriding each field that is listed. In addition, if any configuration files are specified in the "Config File(s)" entry of the Settings menu, these files will be read and merged as well. The default configuration object is defined as the following:

```
{
    "logging": {
        "loggers": {
            "default": "debug"
        },
        "sinks": {
            "consoleSinks": [
                {
                    "enable": true,
                    "level": "debug",
                    "enableColors": true,
                    "colorPattern": "\u001B[0m[\u001B[32m%Y-%m-%d %H:%M:%S.%e\
↪u001B[0m] [%^%l%$] [\u001B[35m%n\u001B[0m:\u001B[33m%#\u001B[0m] %v",
                    "noColorPattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v"
                }
            ],
            "fileSinks": [
                {
                    "enable": true,
                    "level": "info",
                    "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
                    "path": {
                        "win": "%AppData%\\Acroname\\BrainD\\log",
                        "mac": "~/Library/Logs/Acroname/BrainD",
                        "lin": "~/.acroname/BrainD/log"
                    },
                    "baseFileName": "braind.log",
                    "maxFiles": 5,
                    "maxFileSize": 5242880
                }
            ]
        },
        "flushInterval": 3000
    },
    "httpServer": {
        "enable": false,
        "address": "0.0.0.0",
        "port": 9005,
        "threadCount": 1
    },
    "httpsServer": {
        "enable": true,
```

(continues on next page)

(continued from previous page)

```
        "address": "0.0.0.0",
        "port": 9006,
        "threadCount": 1,
        "sslCert": "",
        "sslKey": ""
    },
    "pollingRates": {
        "acronameDevicesState": 300,
        "devices": 300,
        "brainstem": 1000
    }
}
```

Note: The `config['httpServer']` and `config['httpsServer']` configuration objects will have no effect if the BrainD desktop application is used. The value in the Settings dialog takes priority.

Examples

Add an additional log file output:

```
{
    "logging": {
        "sinks": {
            "fileSinks": [
                {
                    "enable": true,
                    "level": "info",
                    "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
                    "path": {
                        "win": "C:\\Users\\username\\Desktop",
                        "mac": "~/Desktop",
                        "lin": "~/Desktop"
                    },
                    "baseFileName": "custom-log-file.log"
                }
            ]
        }
    }
}
```

Increase the log flush rate from 3000ms to 500ms, and reduce the USB tree polling rate from 300ms to 1000ms:

```
{
    "logging": {
        "flushInterval": 3000
    },
    "pollingRates": {
        "devices": 1000
    }
}
```

2.2.6 Logging

BrainD generates runtime logging to aid in debugging and understanding for many events. These log files are stored by default in the following path for each operating system:

1. Windows: `%AppData%\Acroname\BrainD\log`
2. Mac: `~/Library/Logs/Acroname/BrainD`
3. Linux: `~/.acroname/BrainD/log`

By default, these log files will rotate when they reach 5MB in size, up to a maximum of five log files. After this point, old log files will be removed to make room for new ones.

For information on how to adjust the logging parameters or specify new log files, see *Configuration*.

2.2.7 Platform Specific Considerations

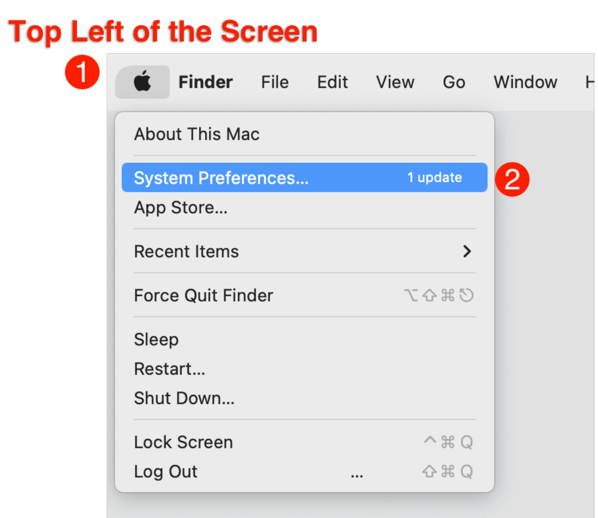
Configure MacOS to Advertise Its Hostname

Name resolution is a process where a readable name, e.g. `ElizaPC`, can be resolved into a numerical IP address, e.g. `192.168.41.75`. Name resolution eases server connections, by allowing usage of URLs like `https://ElizaPC` instead of a hard-to-remember, and fragile, `https://192.168.41.75`.

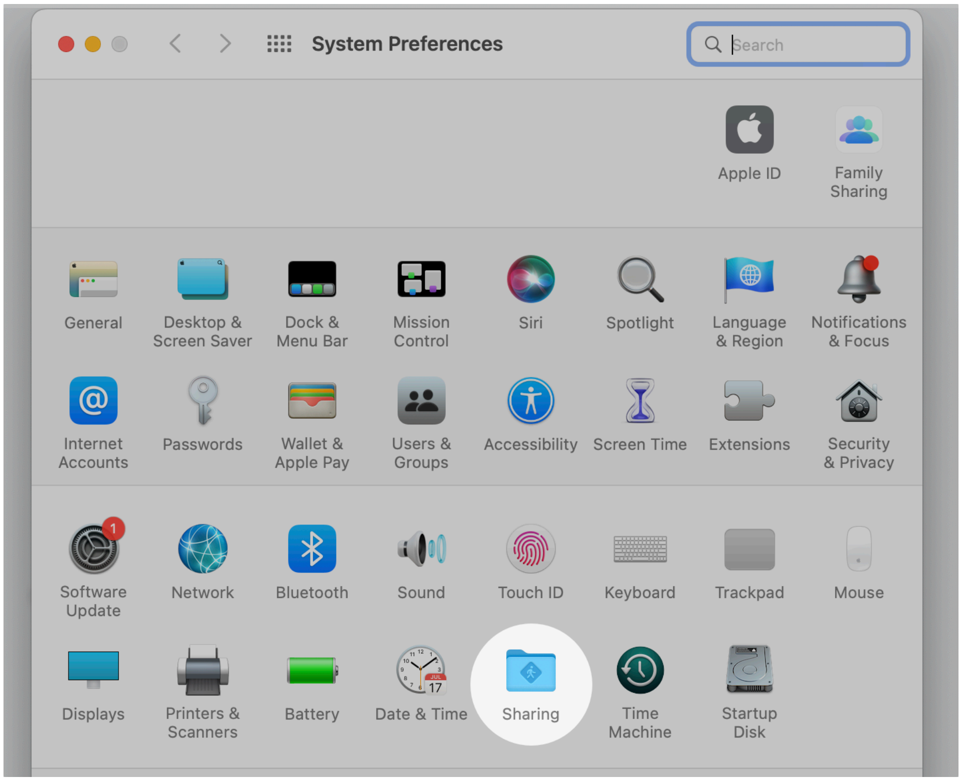
MacOS, in default configuration, doesn't provide resolution for the host's own name.

Here's how to fix that.

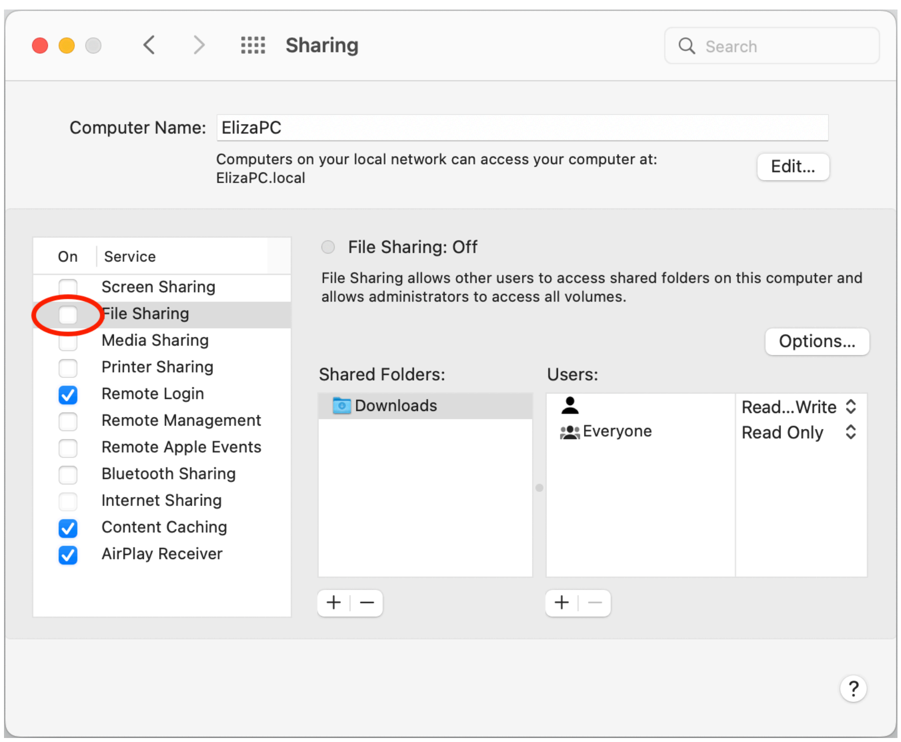
1. Get to System Preferences via the Apple Menu in the top left of the screen.



2. Go to the "Sharing" Panel



3. Enable File Sharing by checking this box. Other devices on the network can now resolve this machine's name. Note: File sharing does not have to remain enabled but must be enabled at least once for name resolution.



4. Configure these settings to your application's needs. This configuration is just for pictorial example. Other devices on the network can still resolve this machine's name, if File Sharing *was* ever enabled.

2.2.8 Functionality and Features

1. Full remote control of an Acroname device.
2. Telemetry and status information on the host computer's entire usb tree.
3. Simultaneous access from multiple client devices.
4. Extends Acroname's Brainstem protocol, the shared core of all Acroname products.
5. Convenient JSON response format.
6. Mirrors the existing Brainstem interface providing users a new, convenient way to control their devices.

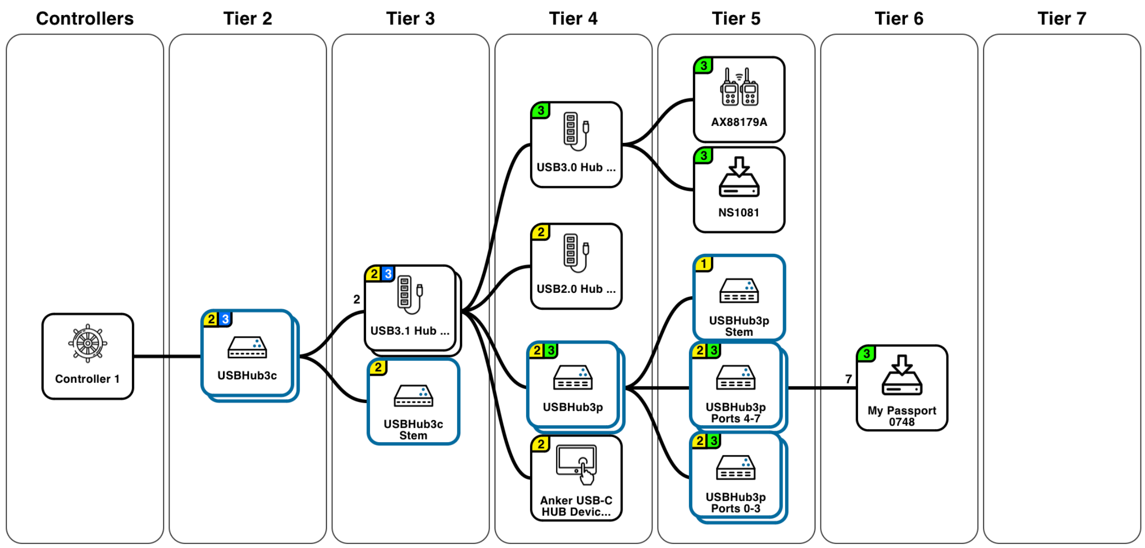
2.2.9 Audio-Video Conferencing Solutions

1. **Peripheral Management**: BrainD can efficiently handle USB peripherals such as microphones, cameras, and speakers.
2. **Enhanced Quality**: Proper management of USB devices can lead to improved audio and video quality during conferences. BrainD can highlight bandwidth allocation and ensure that devices and installations work optimally.
3. **Remote Diagnostics**: Troubleshooting issues with connected devices becomes more straightforward with BrainD by providing diagnostic information and assisting in resolving technical problems remotely.

2.2.10 Test and Measurement Applications

1. **Flexibility**: The ability to dynamically add or remove devices based on specific testing needs enhances flexibility. Engineers can adapt their setups quickly and efficiently, responding to changing requirements without significant disruptions.
2. **Data Logging and Analysis**: BrainD can facilitate real-time data logging and analysis, allowing research, development, and quality control applications to monitor measurements, perform in-depth analysis, and visualize results conveniently.

2.3 ControlRoom



ControlRoom is a service and web application that lets you control, monitor, and reset USB ports in connected conference rooms through a browser. ControlRoom gives a real-time view of your system's connected USB devices for easy troubleshooting and to ensure correct installation. Remotely resolve issues that would normally require a physical unplug / plug cycle.

2.3.1 Installation

To install ControlRoom, download the appropriate installer[66] for your operating system.

Requirements

ControlRoom requires a supported Acroname hub with the ControlRoom add-on software feature[67]:

- USBHub3c[68] – USB-C Hub with Power Delivery Analyzer + Tester
- USBHub3p[69] – Programmable Industrial 8-port USB 5Gbps Hub
- USBHub2x4[70] – Industrial Intelligent 4-port Hi-Speed USB Hub

66 https://acroname.com/controlroom
67 https://acroname.com/store/controlroom-feature
68 https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro
69 https://acroname.com/store/programmable-industrial-hub-s79-usbhub-3p
70 https://acroname.com/store/industrial-intelligent-4-port-hub-s77-usbhub-2x4

Supported operating systems

ControlRoom binaries are available for the following platforms:

- Windows 10 and higher
- Intel and Apple Silicon Macs running macOS 10.15 or higher
- Linux:
  - x86_64 Ubuntu LTS 20.04, 22.04
  - arm64v8 Ubuntu LTS 20.04
  - armhf Debian Bullseye

While ControlRoom may work on earlier OS versions, support is limited to the listed versions.

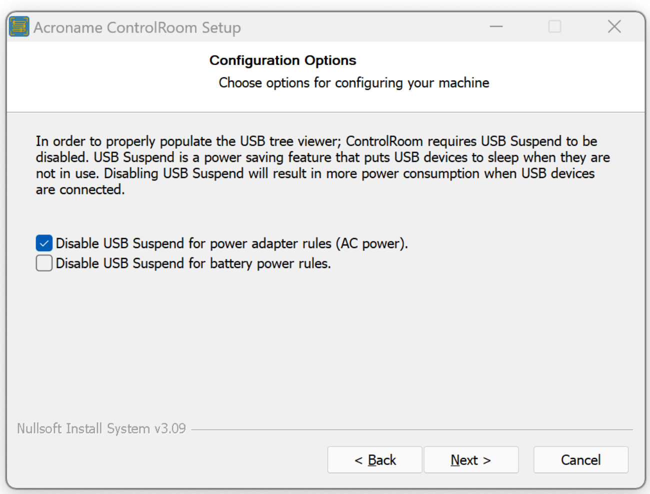
Supported browsers

The ControlRoom web application supports the latest stable versions of Safari, Brave, Edge, Chrome, and Firefox.

Install and launch

Windows

- Open the downloaded .exe file to launch the installer
- Click "Run" in the security popup
- Follow the installer prompts to accept the license

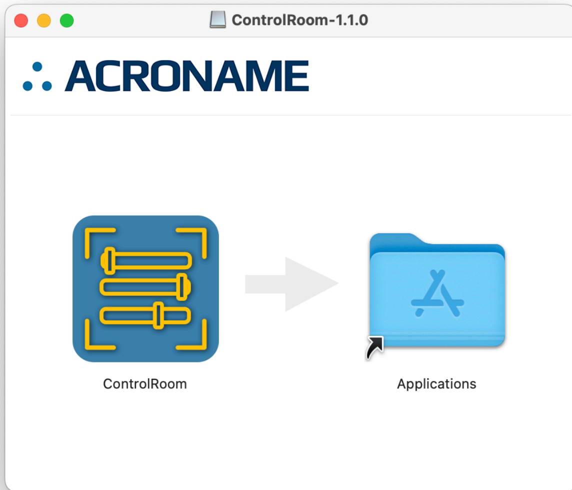


- To reliably view the USB tree, follow the prompt to disable USB suspend
- Choose an install location and start menu folder

Mac

- Open the downloaded .dmg disk image file

- Drag the ControlRoom app into the applications folder inside the disk image



Linux

- The downloaded file will be a .deb software package
- In a command line, run `sudo dpkg -i <image file>.deb` to install the ControlRoom application

Note: If dependencies are missing, the above command may fail. Run `sudo apt-get -f install` to install the dependencies, then re-run the previous install command.

- After installation, open ControlRoom from the applications selector or from the command line

Note: On Linux, the USB Tree may not enumerate all devices on the system if the user does not have write permissions to `/dev/bus/usb`.

2.3.2 Usage

Getting Started

To get started with ControlRoom, first confirm that it's *installed*, then connect a supported Acroname device with a valid ControlRoom License[71]. This guide will use a USBHub3c[72].

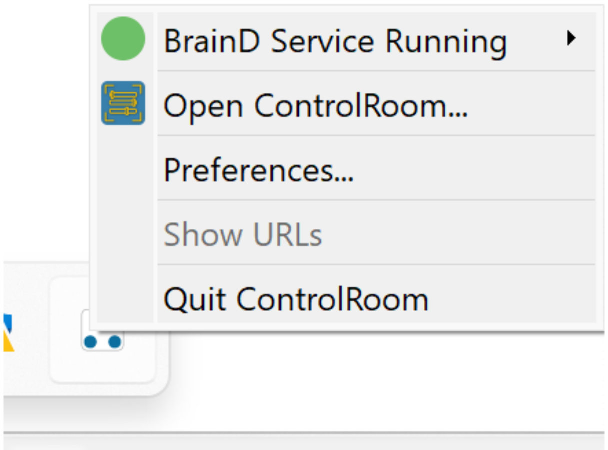
Windows

- Start ControlRoom by selecting it in the Start Menu under "Acroname"
- Acroname's 3-dot icon ( ) will appear in the System Tray, or in Hidden Icons
- Right click to pop up the ControlRoom menu

[71] https://acroname.com/store/controlroom-feature

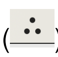
[72] https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro

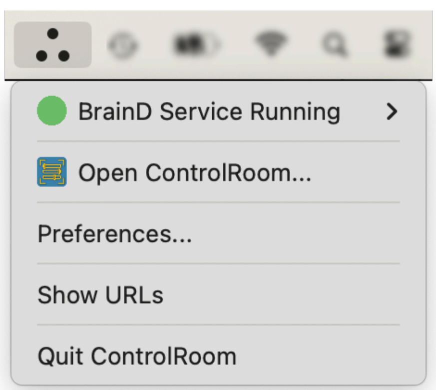
- In the ControlRoom popup menu, select “Open ControlRoom” to launch the viewer in a browser:



ControlRoom popup menu (Windows)

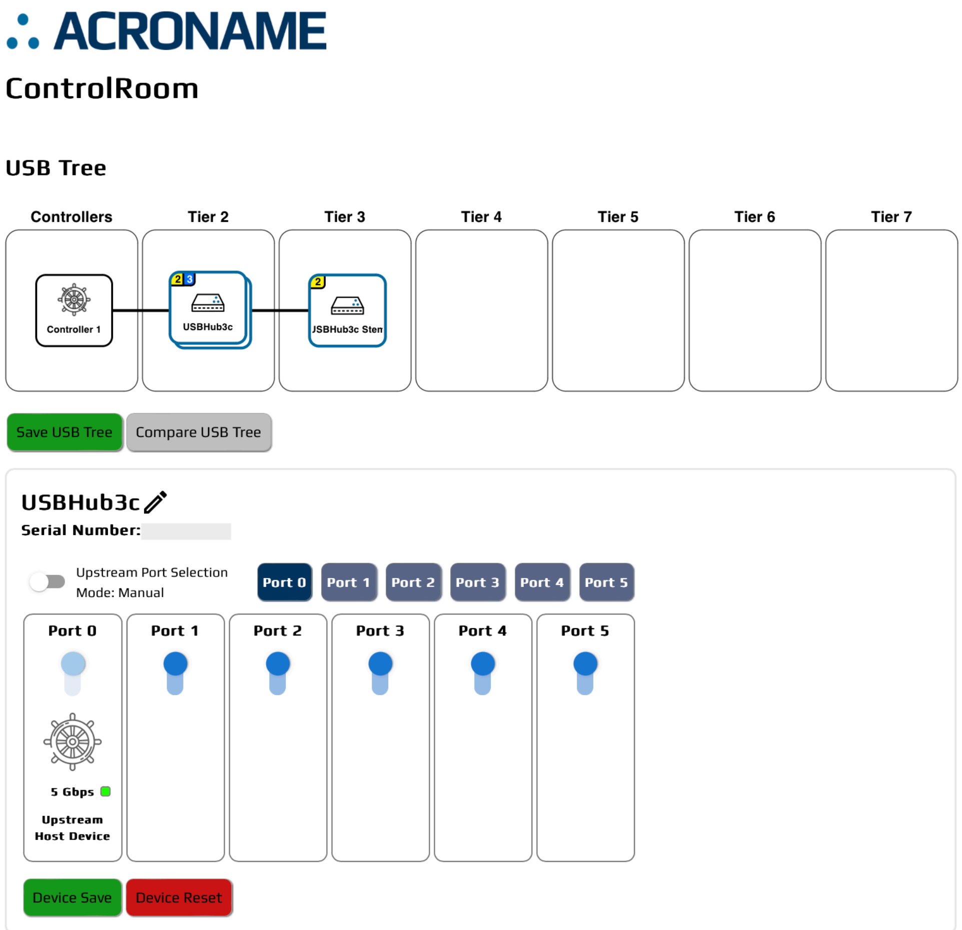
Mac

- Launch the ControlRoom app from the applications folder
- Acroname’s 3-dot icon () will appear in the Mac menu bar
- Click to pop up the ControlRoom menu
- In the ControlRoom popup menu, select “Open ControlRoom” to launch the viewer in a browser:



ControlRoom popup menu (Mac)

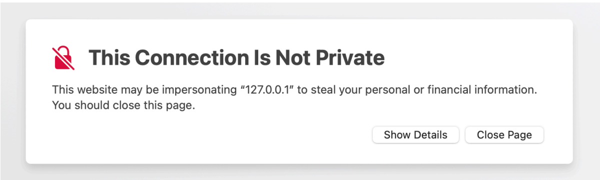
ControlRoom viewer web app



ControlRoom view with one USBHub3c with no attached devices

The ControlRoom viewer web app shows the USB tree and port control panels for each connected Acroname device.

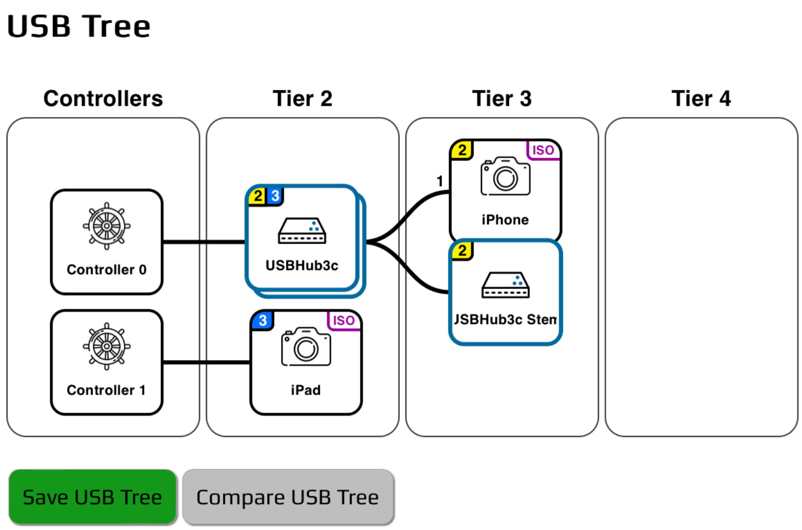
Note: If ControlRoom is configured to use HTTP Secure, a warning may pop up if using a self-signed certificate. Select show details and "visit this website" to continue.



USB tree

The top of the page contains the USB Tree section. This section shows all of the USB devices connected to the system, as well as the hierarchy of the devices. The tree will continuously update as devices are connected and disconnected. See *the USB Tree section* for more information.

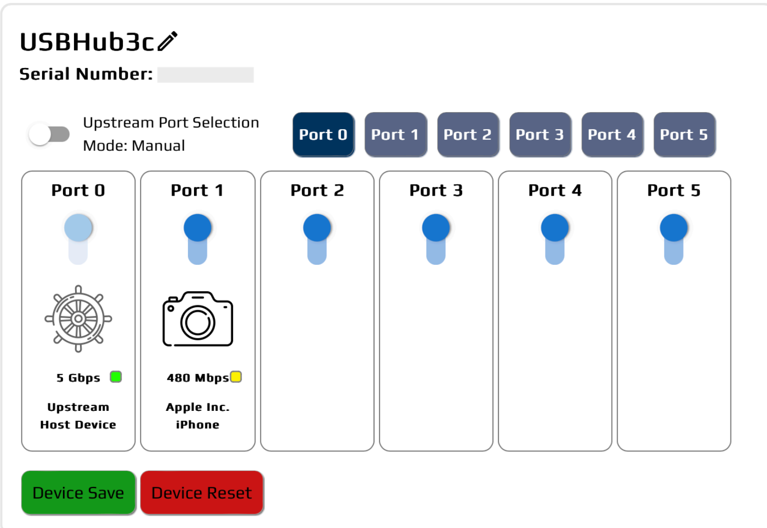
In this example, USBHub3c is the hub, USBHub3c Stem is a microcontroller responsible for controlling the hub and communicating with ControlRoom. If a box shows “2” and “3” (2 3), then the device is connected to the host via both USB 2 and USB 3. If just a “2” is showing (2), the device is only connected via USB 2.



USB tree with an iPhone connected

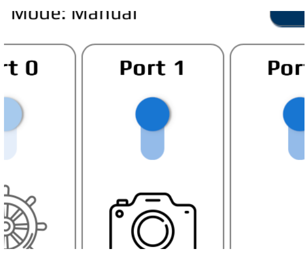
To demonstrate, connect a USB Device into USBHub3c Port 1. It will appear in the tree as a child of the USBHub3c. For this example an iPhone is connected, and the device icon shows that it is connected by USB 2 and the “ISO” mark indicates that it it capable of isochronous transfers such as streaming video and audio.

Port control panel



Port control panel for USBHub3c with an iPhone connected

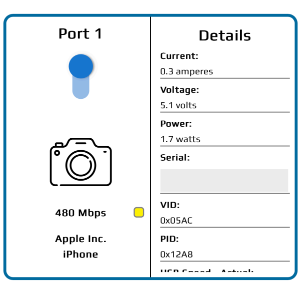
Below the USB Tree are panels representing each connected Acroname device. This section allows a user to power ports on and off, view attached devices, and more. See *the Port Control section* for more information.



Port Toggle

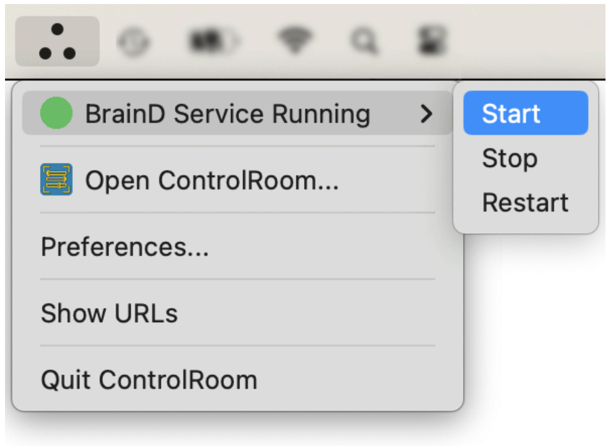
To demonstrate, disable the device on USBHub3c Port 1 by toggling the blue enable switch. The device will be removed from the USB Tree.

Detailed information about each connected device can be seen by clicking on the device icon below each port power switch:



Taskbar menu

The ControlRoom background application enables simple configuration and manipulation of the application service though a taskbar menu.



ControlRoom taskbar menu

BrainD Service [Running or Stopped] – shows whether the BrainD service that communicates with Acroname devices is running:

- Start – starts the BrainD service. This is required for ControlRoom to operate
- Stop – stops the BrainD service
- Restart – restarts the BrainD service. All active connections will terminate

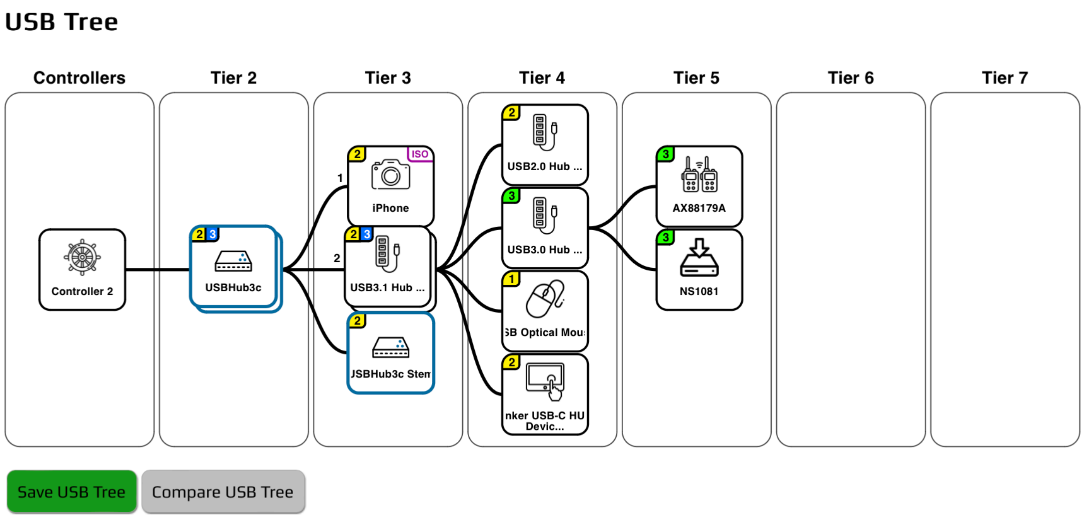
Open ControlRoom – launches the ControlRoom web app in a browser

Preferences – opens the *settings and configuration dialog*

Show URLS – displays all broadcasted URLs that can be used to access the local ControlRoom session

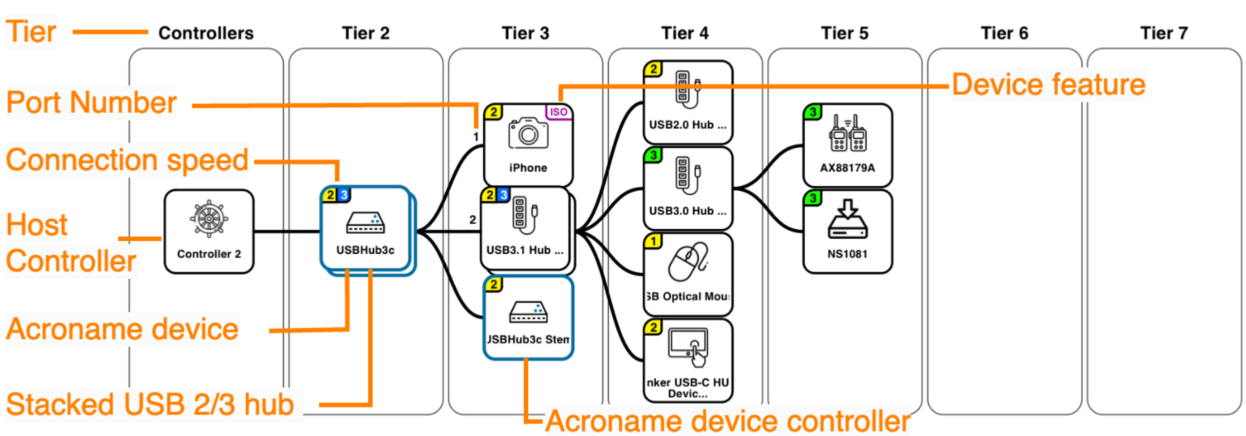
Quit ControlRoom – closes and quits the ControlRoom taskbar and stops the BrainD service

USB Tree



The USB Tree is a great way to visualize the USB device tree for the host processor. USB has the concept of tiers: the host and root hub are on the first tier. A device connected to the root hub is on the second tier. If a hub is added in between the device and the root hub, the device moves down to the third tier. Up to 5 non-root hubs can be added in series for a maximum of 7 USB tiers (5 Hubs + root hub + device).

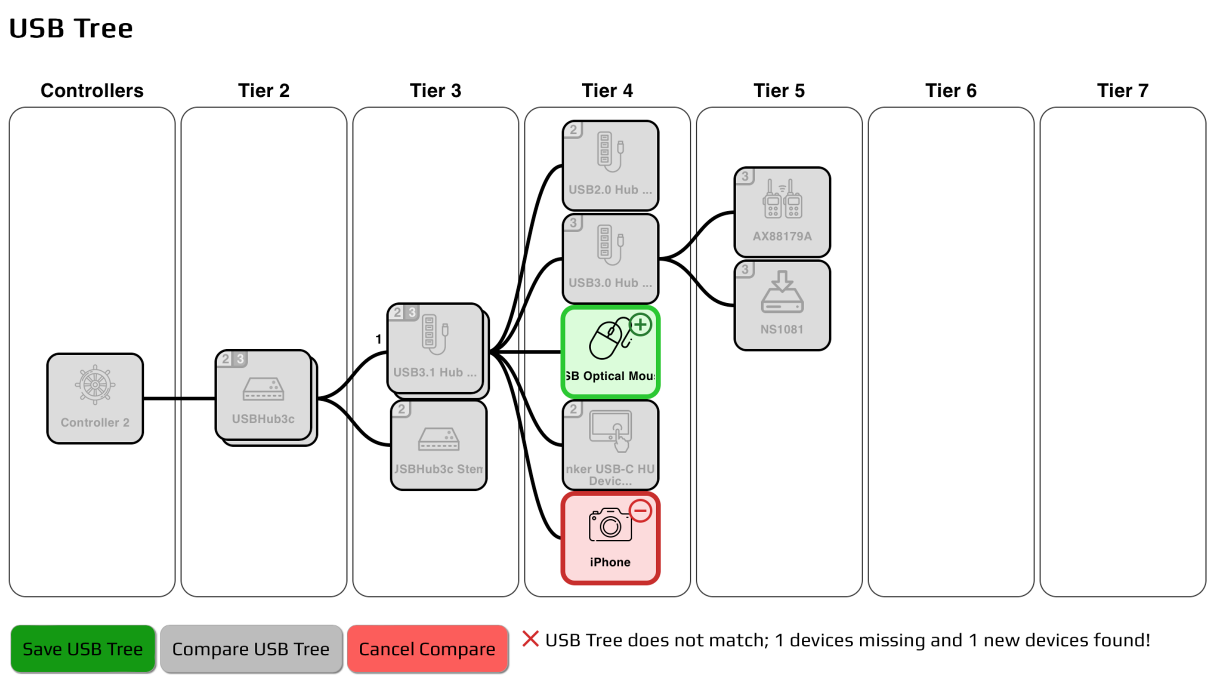
The USB Tree provides a simple visualization of the 7 hub tiers along with each device and how it is connected back to the host controller.



Label	Description
Tier	Depth of USB tree, up to 7
Port number	Physical port of the Acroname product the USB device is connected to (Only available when connected to Acroname products)
Connection speed	3 = USB 3 (blue = 10 Gb/s, green = 5 Gb/s)
	2 = USB 2 480 Mb/s
	1 = USB 1
Host controller	Host controller node, tier 1
Acroname device	Acroname devices are outlined in blue to for easy indentification
Stacked USB 2/3 hub	Hubs with both USB 2 and USB 3 connections are shown as stacks
Acroname device controller (stem)	Responsible for controlling the hub and communicating with ControlRoom
Device feature	USB device feature. **ISO** indicates that the device is capable of isochronous transfers such as streaming video and audio

USB Tree Comparisons

The USB Tree can be saved as a PDF with embedded data. The current tree can then be compared with a saved tree to easily identify changes.



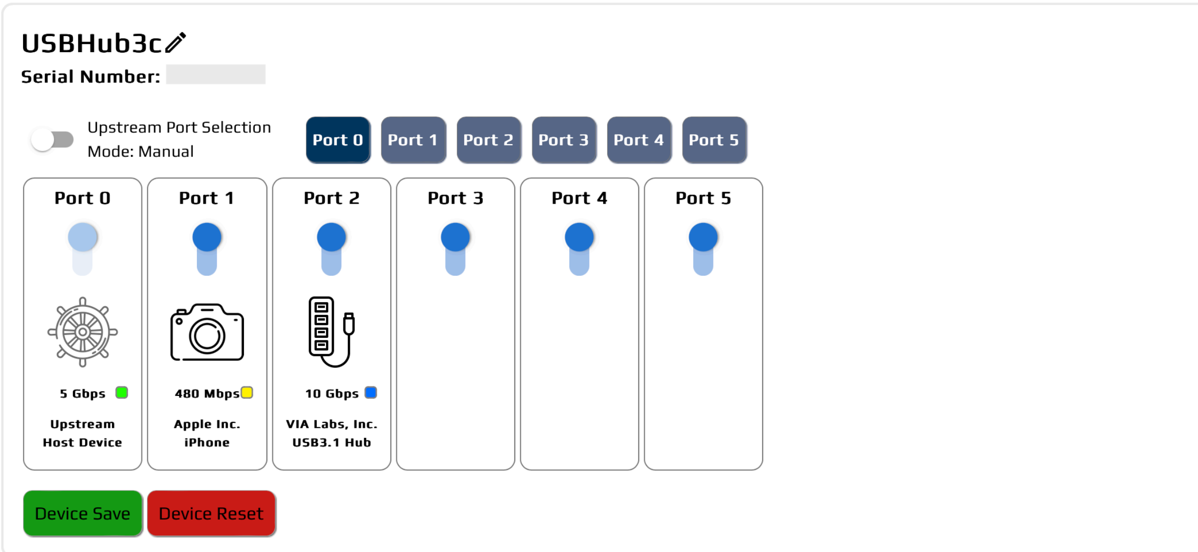
Save USB Tree – saves to a PDF file. Saved USB Tree PDF files can be used as recipes for room designs

Compare the USB Tree – uploads a saved USB Tree PDF file for dynamic comparison with the current tree. USB devices that match the loaded USB Tree recipe will show a gray background. Missing devices will be shown in red and new devices in green. If a device has been moved to a different port, it will appear twice, once in green as new, and once in red as missing

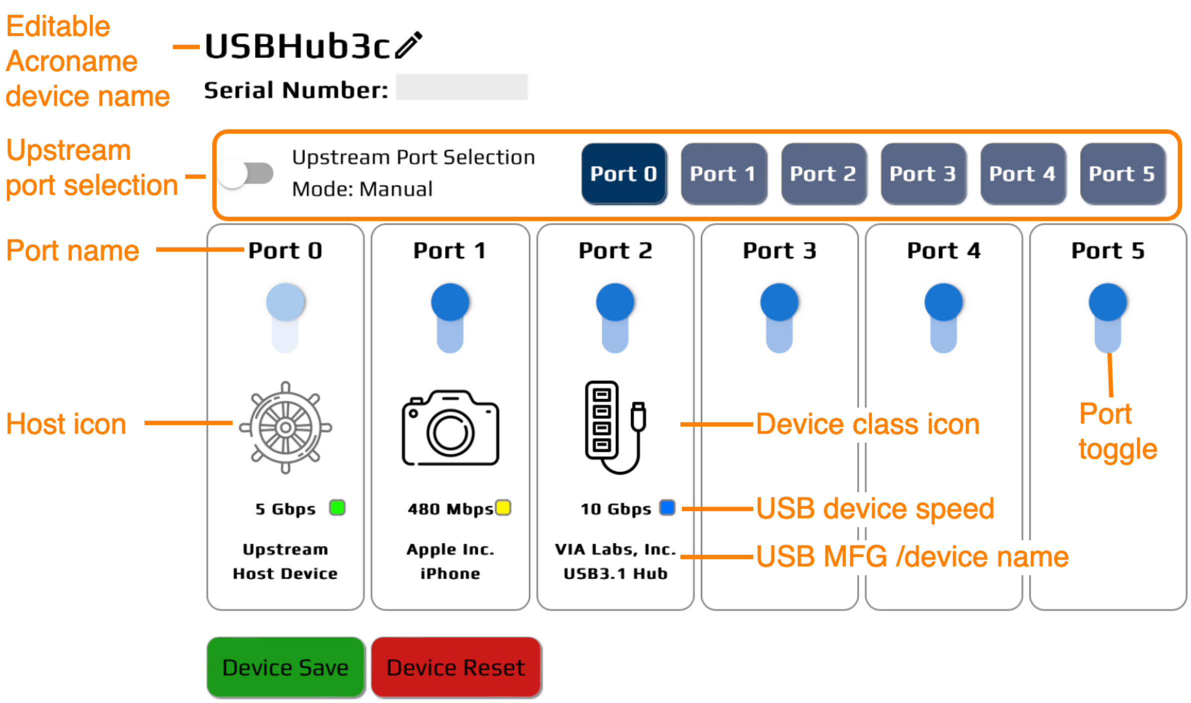
Cancel Compare – cancels the USB Tree comparison

Summary – if the current tree does not match, a summary will be displayed indicating how many devices are new, and how many devices are missing

Port Control



The Port control panels represent each connected Acroname device, providing detailed control and information per port.



Editable Acroname device name – click the pencil icon to change the friendly name, click *device save* to store the change to the device internal memory. Default – product family name

Serial number – unique to the Acroname device

Upstream port selection – toggle to enable manual or auto (lowest port number priority) host port selection. In manual mode, the upstream port can be selected directly by clicking the dark blue port name buttons

Port name – name of Acroname device port. Default – numerical value of port

Device class icon – unique for each device class

Port toggle – independently enables and disables each port. Up = on

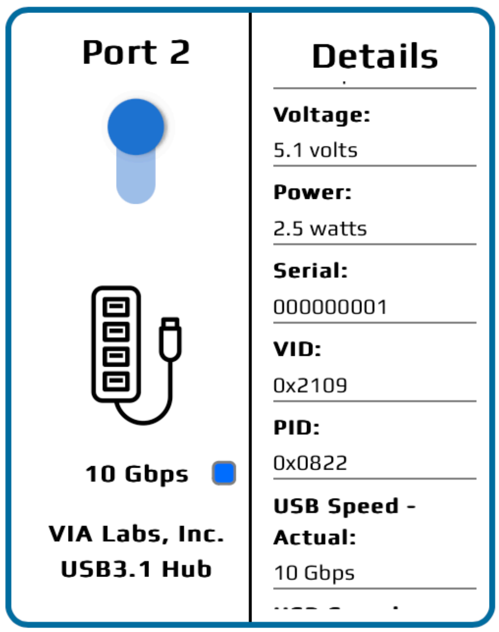
USB device speed – nominal device connection speed

USB MFG / device name – manufacturer and device name descriptors

Device save – stores changes to the hub friendly name and port enabled / disabled toggles. Saved changes will persist through power cycles

Device reset – reboots the Acroname device, temporarily disconnecting and reconnecting all connected USB devices

Port Details Expanded



Expanded port view with details

Click the USB device icon to expand each port view for realtime details:

- Current
- Voltage
- Power
- Serial number
- Vendor and Product ID
- Actual and maximum USB speeds
- Device manufacturer and device name
- Cable information (if advertised):
  - Cable current max

- Cable voltage max
- Cable speed max
- Cable orientation
- Cable type

2.3.3 Advanced

For more advanced ControlRoom options, see configuration, logging, and security.

Configuration

Preferences

To change the default settings of ControlRoom, the desktop application provides a simple dialog window that enables users to change the HTTP server settings and specify *SSL/TLS Certificates*. To open the dialog, click on the taskbar menu and select “Preferences”.

Basic settings

Basic ControlRoom settings

Config File(s) – optional path to load JSON *configuration files*

Start BrainD at login – toggles BrainD service to start automatically

Allow remote access – lets other computers view the local USB tree information over a network connection

Use HTTP Secure – *Enables SSL/TLS* on client connections.

Restore Defaults – sets configuration to factory defaults

Cancel – closes the preferences window without saving

Save – saves application settings, restart ControlRoom to apply

Advanced settings

Advanced ControlRoom settings

Server Address – IP address to allow remote browsers to connect and view the local ControlRoom instance. Default is 127.0.0.1

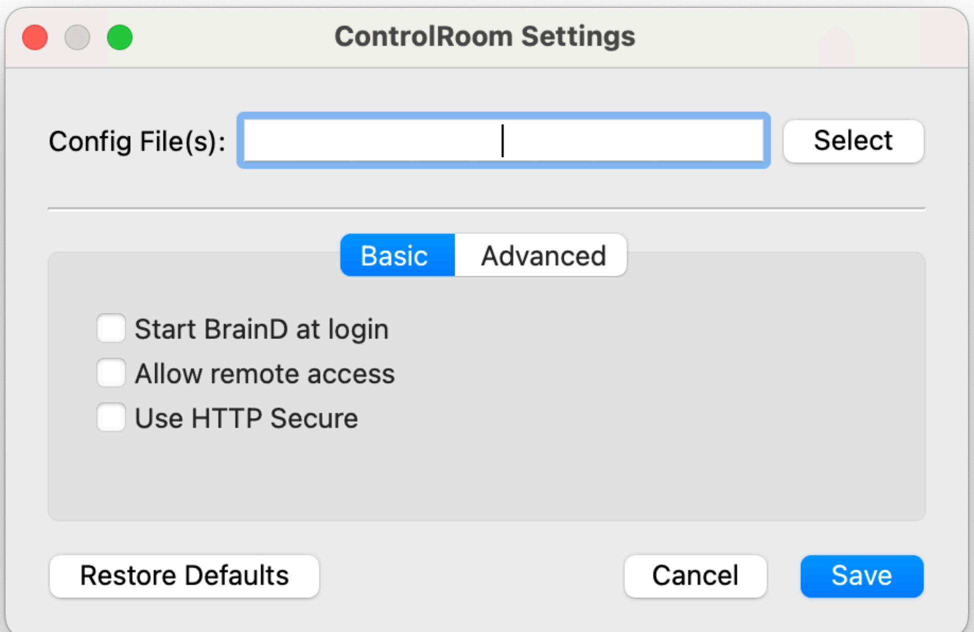
Server Port – server port for remote connections. Default is 9005

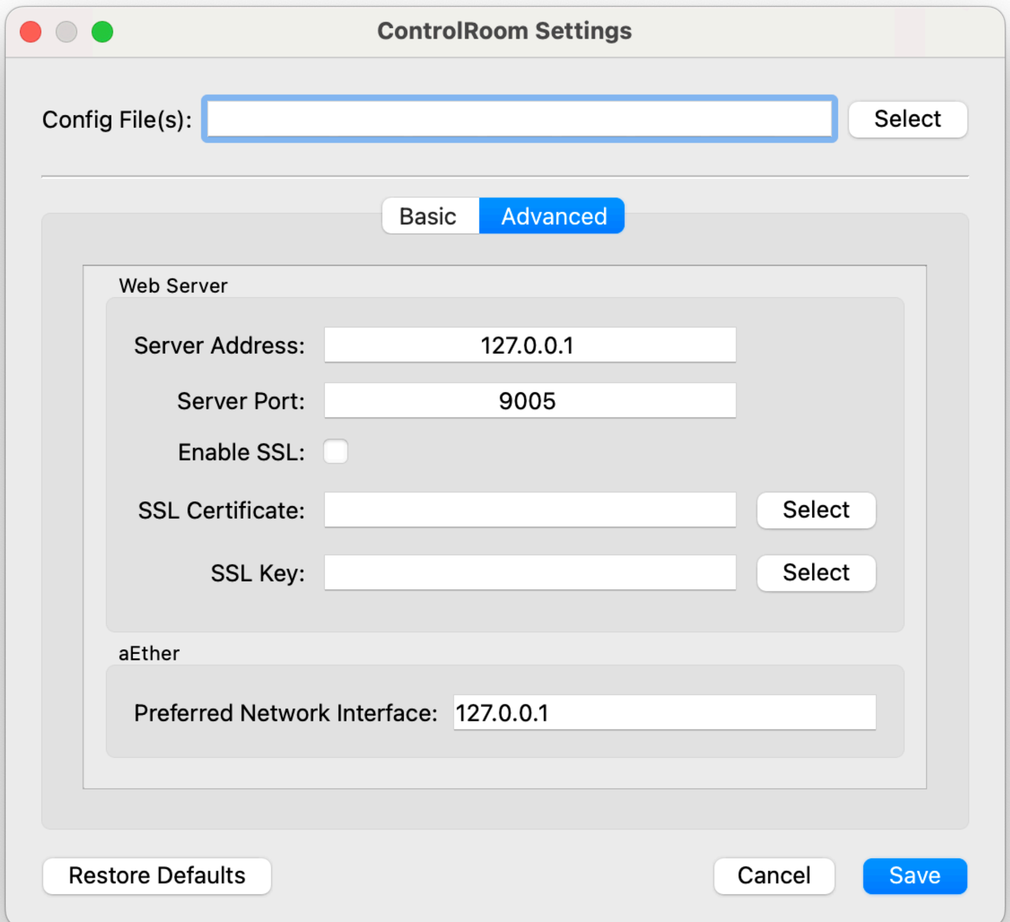
Enable SSL – *enables SSL/TLS* on client connections.

SSL Certificate – *SSL/TLS Certificate (.crt) file*. If not specified, a self-signed cert will be generated.

SSL Key – *SSL/TLS Key (.key) file*. If not specified, a self-signed cert will be generated.

Preferred Network Interface – IP address to allow remote applications to connect over aEther. Default is 127.0.0.1


ControlRoom Settings
Config File(s):
Select
Basic
Advanced
Start BrainD at login
Allow remote access
Use HTTP Secure
Restore Defaults
Cancel
Save


ControlRoom Settings
Config File(s):
Select
Basic
Advanced
Web Server
Server Address:
127.0.0.1
Server Port:
9005
Enable SSL:
SSL Certificate:
Select
SSL Key:
Select
aEther
Preferred Network Interface:
127.0.0.1
Restore Defaults
Cancel
Save

Configuration Files

ControlRoom allows for more fine-grained control of the backend service through the use of JSON configuration files. This configuration includes settings for *logging*, polling rates, and web server configuration.

The BrainD service will read the following configuration file for the current operating system:

1. Windows: `%AppData%\Acroname\BrainD\config\config.json`
2. Mac: `~/Library/Application Support/Acroname/BrainD/config/config.json`
3. Linux: `~/.acroname/BrainD/config/config.json`

Each entry in the `config.json` file will be merged into the default configuration object, overriding each field that is listed. In addition, if any configuration files are specified in the "Config File(s)" entry of the Settings menu, these files will be read and merged as well. The default configuration object is defined as the following:

```
{
    "logging": {
        "loggers": {
            "default": "debug"
        },
        "sinks": {
            "consoleSinks": [
                {
                    "enable": true,
                    "level": "debug",
                    "enableColors": true,
                    "colorPattern": "\u001B[0m[\u001B[32m%Y-%m-%d %H:%M:%S.%e\
↪u001B[0m] [%^%l%$] [\u001B[35m%n\u001B[0m:\u001B[33m%#\u001B[0m] %v",
                    "noColorPattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v"
                }
            ],
            "fileSinks": [
                {
                    "enable": true,
                    "level": "info",
                    "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
                    "path": {
                        "win": "%AppData%\\Acroname\\BrainD\\log",
                        "mac": "~/Library/Logs/Acroname/BrainD",
                        "lin": "~/.acroname/BrainD/log"
                    },
                    "baseFileName": "braind.log",
                    "maxFiles": 5,
                    "maxFileSize": 5242880
                }
            ]
        },
        "flushInterval": 3000
    },
    "httpServer": {
        "enable": false,
        "address": "0.0.0.0",
        "port": 9005,
        "threadCount": 1
    },
    "httpsServer": {
        "enable": true,
```

(continues on next page)

(continued from previous page)

```
        "address": "0.0.0.0",
        "port": 9006,
        "threadCount": 1,
        "sslCert": "",
        "sslKey": ""
    },
    "pollingRates": {
        "acronameDevicesState": 300,
        "devices": 300,
        "brainstem": 1000
    }
}
```

Note: The `config['httpServer']` and `config['httpsServer']` configuration objects will have no effect if the ControlRoom desktop application is used. The value in the Settings dialog takes priority.

Examples

Add an additional log file output:

```
{
    "logging": {
        "sinks": {
            "fileSinks": [
                {
                    "enable": true,
                    "level": "info",
                    "pattern": "[%Y-%m-%d %H:%M:%S] [%l] [%n:%#] %v",
                    "path": {
                        "win": "C:\\Users\\username\\Desktop",
                        "mac": "~/Desktop",
                        "lin": "~/Desktop"
                    },
                    "baseFileName": "custom-log-file.log"
                }
            ]
        }
    }
}
```

Increase the log flush rate from 3000ms to 500ms, and reduce the USB tree polling rate from 300ms to 1000ms:

```
{
    "logging": {
        "flushInterval": 3000
    },
    "pollingRates": {
        "devices": 1000
    }
}
```

Logging

ControlRoom generates runtime logging to aid in debugging and understanding for many events. These log files are stored by default in the following path for each operating system:

1. Windows: `%AppData%\Acroname\BrainD\log`
2. Mac: `~/Library/Logs/Acroname/BrainD`
3. Linux: `~/.acroname/BrainD/log`

By default, these log files will rotate when they reach 5MB in size, up to a maximum of five log files. After this point, old log files will be removed to make room for new ones.

For information on how to adjust the logging parameters or specify new log files, see *Configuration*.

Security

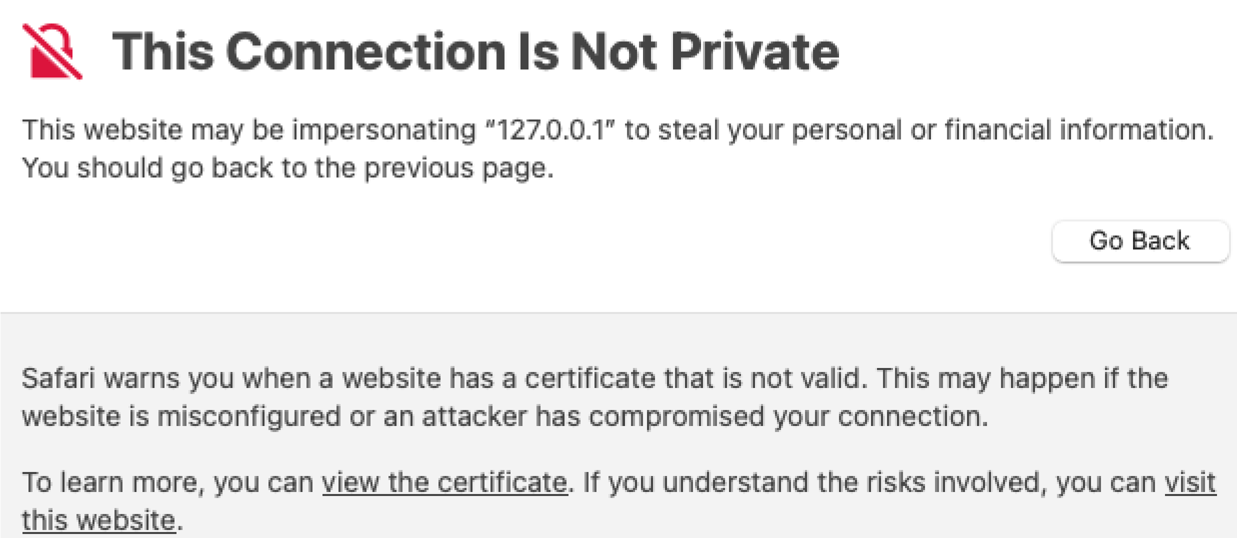
SSL Certificates

SSL (Secure Sockets Layer) is a fundamental technology for securing web applications by encrypting data, ensuring data integrity, authenticating servers, and enhancing user trust.

SSL layers encrypt the data transmitted between a user's web browser and server. This encryption ensures that the data remains unreadable and secure even if intercepted by malicious actors. SSL includes mechanisms to ensure data integrity during transmission using cryptographic hash functions to detect unauthorized modifications or tampering. In the event of altered data during transit, the recipient can see and reject the modified data.

ControlRoom allows loading a user's SSL certificate through the *settings and configuration dialog*. Many external service providers can provide an SSL certificate; one may already have an SSL certificate for ControlRoom installations.

Use of a self-signed certificate may cause a warning to appear when accessing the ControlRoom webpage or BrainD REST endpoints. This is caused by the browser failing to authenticate the received certificate against a trusted certificate authority.



OpenSSL is an option that one can use to create a self-signed SSL certificate. The following command may be used to generate a self-signed certificate:

Bash

```
openssl req -new -x509 -days 365 -nodes -keyout example.key -out example.crt
```

By default, ControlRoom will serve an insecure web page over HTTP, without any certificates or encryption. If insecure communications over HTTP are not desirable, the *settings and configuration dialog* can be used to enable HTTPS/SSL communication. If a certificate is not specified, then ControlRoom will automatically generate a self-signed certificate.

Certificates are stored in the following directory:

1. Windows: `%AppData%\Acroname\BrainD\config`
2. Mac: `~/Library/Application\ Support/Acroname/BrainD/config`
3. Linux: `~/.acroname/BrainD/config`

Platform Specific Considerations

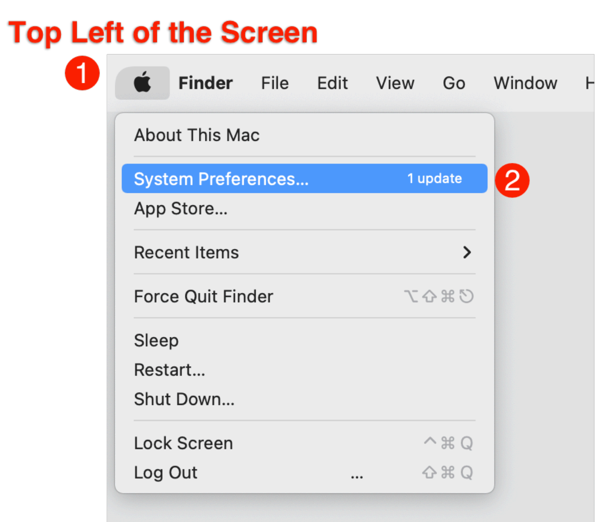
Configure MacOS to Advertise Its Hostname

Name resolution is a process where a readable name, e.g. `ElizaPC`, can be resolved into a numerical IP address, e.g. `192.168.41.75`. Name resolution eases server connections, by allowing usage of URLs like `https://ElizaPC` instead of a hard-to-remember, and fragile, `https://192.168.41.75`.

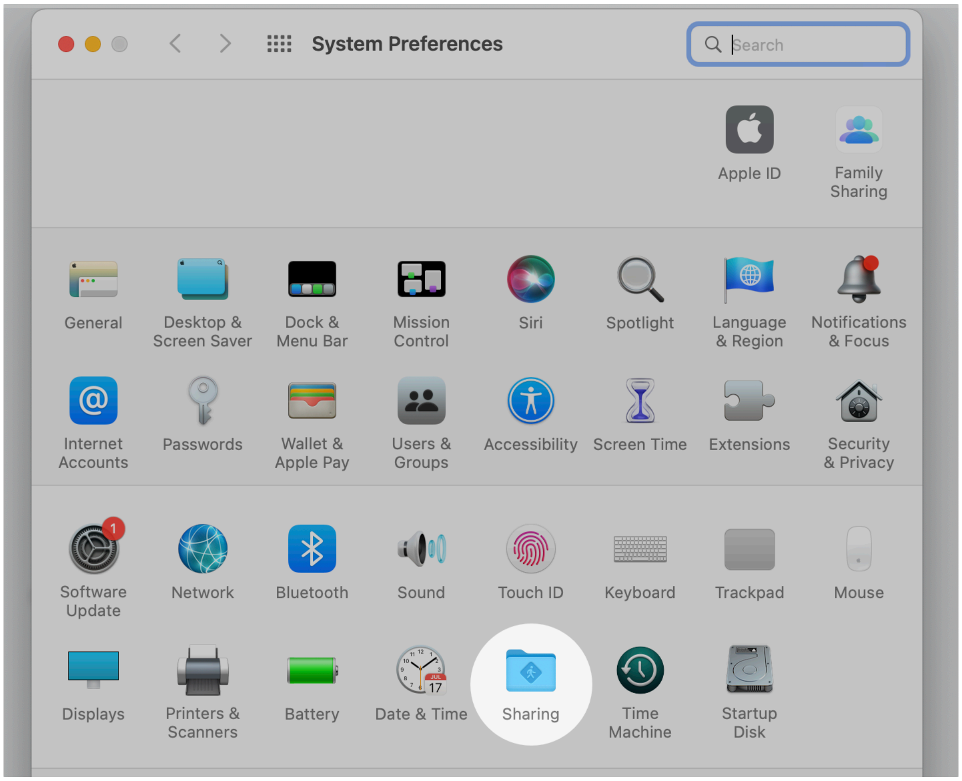
MacOS, in default configuration, doesn't provide resolution for the host's own name.

Here's how to fix that.

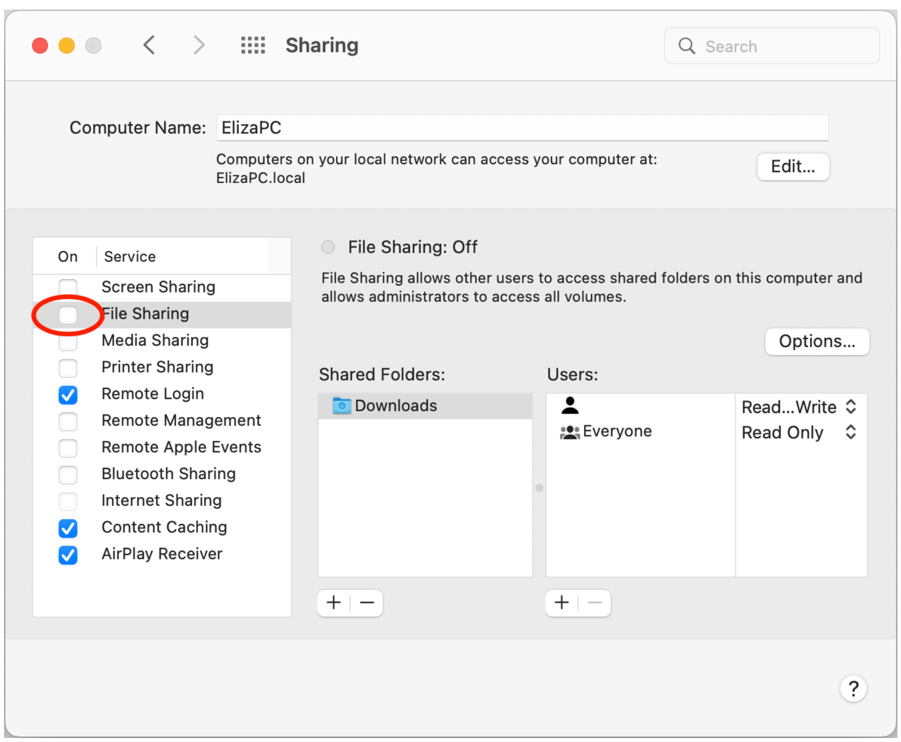
1. Get to System Preferences via the Apple Menu in the top left of the screen.



2. Go to the "Sharing" Panel



3. Enable File Sharing by checking this box. Other devices on the network can now resolve this machine's name. Note: File sharing does not have to remain enabled but must be enabled at least once for name resolution.



4. Configure these settings to your application's needs. This configuration is just for pictorial example. Other devices on the network can still resolve this machine's name, if File Sharing *was* ever enabled.

2.3.4 Features and Functions

- Visualize the USB tree in a browser in real-time
- Create device layout recipes via snapshots
- Compare the current device layout to a loaded snapshot
- View device status including:
  - Device connection speed
  - Custom devices names
  - Troubleshooting information
- Control Acroname device functions including:
  - Downstream port toggle, simulating a device unplug / plug cycle
  - Upstream port swap
  - Upstream port selection mode
  - Save settings to non-volatile storage
  - Remote Acroname device reset

2.4 Q-Sys

The Q-Sys OS and Designer Software serves as the software-based singular foundation that drives and manages a multitude of Q-Sys Products within the platform, including native software, services and hardware. Acroname's products bring unique control and telemetry of USB devices and peripherials.

The Q-Sys Designer Software's modern IT architecture and a set of development tools (called "Q-Sys Open") enable an entire Ecosystem of third-party integrations developed by approved/endorsed Q-Sys Partners as well as a worldwide community of Q-Sys programmers and developers. Acroname is part of the approved Partners.

2.4.1 Installation

Download the appropriate Q-Sys Design Software[73] for your operating system. See Q-Sys support documentation for installation instructions.

Requirements

- Q-Sys Designer Software Version 9.9 and higher is recommended when using a virtual core instances.

Q-Sys requires a supported Acroname USBHub3c with the Serial Control add-on software feature[74]:

- USBHub3c[75] – USB-C Hub with Power Delivery Analyzer + Tester

Install Plugin

The Q-Sys Asset Manger is the recommended method for managing plugins. To install an Acroname plugin:

1. Click the plugin icon to open the Asset Manager
2. In the Browse tab, search for `Acroname` and select the plugin that correlating the Acroname hardware.
3. Review the Description information for any additional notes or requirements.
4. Click `Install`. When the installation is complete, the Acroname product plugin will appear in the Plugins categoray within the Schematic Elements section of Q-Sys Designer.

2.4.2 Quick Start Guide

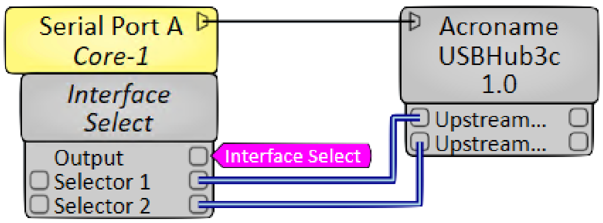
To get started with Q-Sys, follow these steps from the Q-Sys Designer Software:

1. Install Q-Sys Plugin following the *Installation Guide*.
2. Connect an Acroname product with a valid Acroname USBHub3c with the *Serial Control add-on software feature <https://acroname.com/store/t99-serial>* installed. For this guide, a *USBHub3c*.
3. Add an Acroname USBHub3c plugin to the design canvas to create rich USB based applications.

73 https://www.qsys.com/resources/software-and-firmware/q-sys-designer-software/
74 https://acroname.com/store/t99-serial
75 https://acroname.com/store/programmable-industrial-power-delivery-hub-s99-usbhub-3c-pro

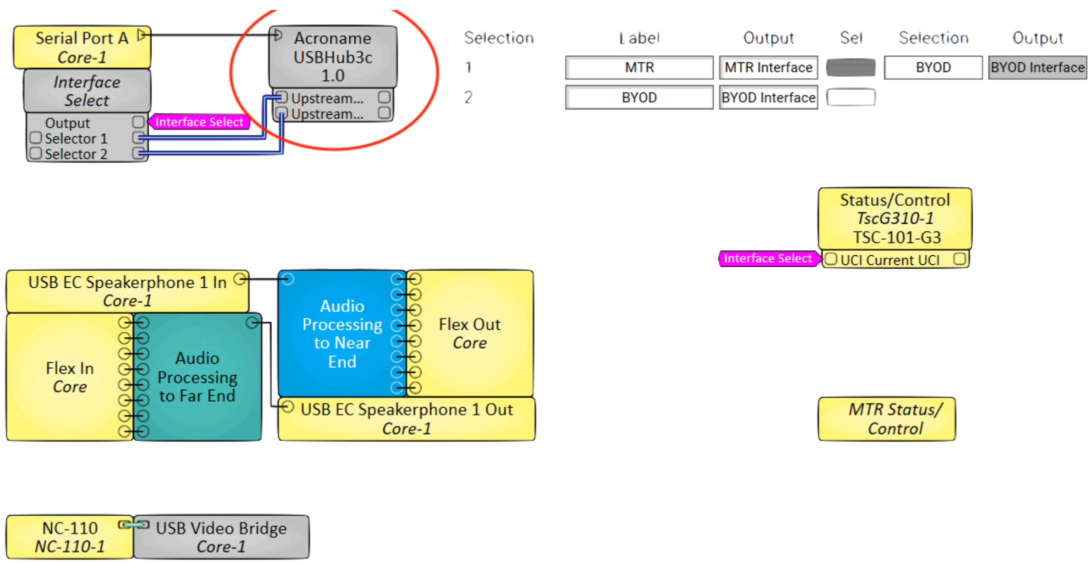
Connecting through Serial Port

The following example shows highlights that a serial port on the Core can be connected to a USBHub3c serial port.



Change Upstream Port

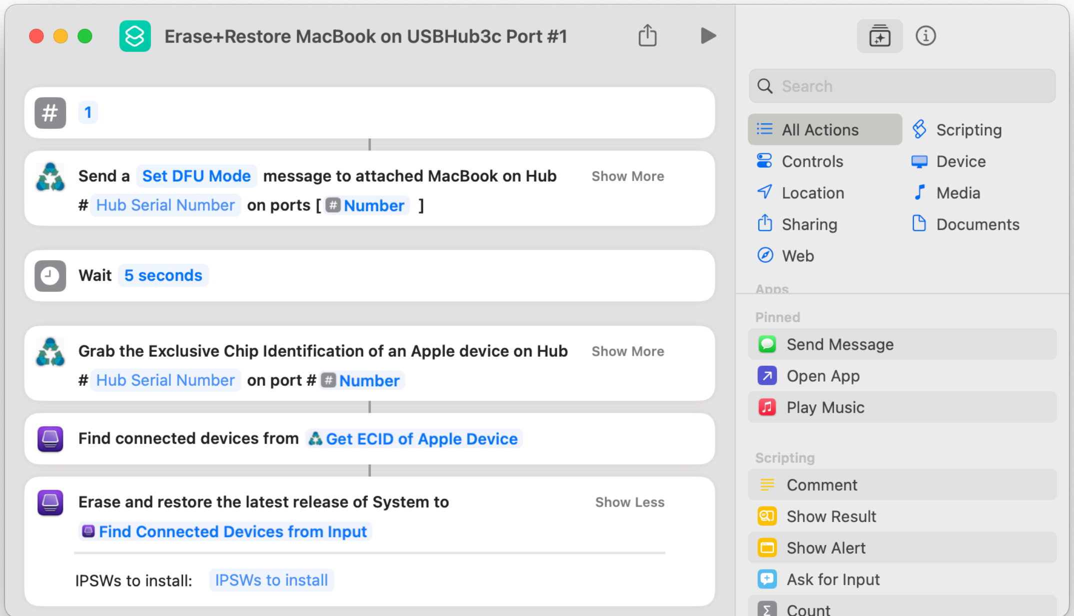
The following example shows how an output selector through the Q-Sys user interface can change the upstream port on the USBHub3c. Switching the upstream port allows USB devices to communicate with two different host machines, which is often found in BYOD applications and environments.



2.4.3 Features and Functions

- Control Acroname device functions including:
  - Downstream port toggle, simulating a device unplug / plug cycle
  - Downstream port connection status
  - Upstream port selection
  - Upstream port connection status
  - Firmware and model information

2.5 DFU Automator



Apple Devices can be put into Device Firmware Update (DFU) mode to allow for firmware updating and restoration, as well as the deployment of custom OS images. This is achieved by sending Vendor-Specific Messages (VDMs) to a DFU-capable port on the target device. VDMs are point-to-point and are not passed on by a USB Hub; however, USBHub3c can generate and send arbitrary VDMs directly to connected devices. This lets Mac Sysadmins re-image multiple Macs at once, greatly speeding up deployment.

DFU Automator is a small application that enables Apple Shortcuts to control USBHub3c's DFU functions. DFU Automator operates as a background tool, responding to commands issued by Apple Shortcuts. When combined with Apple Configurator actions, shortcuts can fully automate Mac deployment and provisioning.

Included with DFU Automator is an example shortcut showing how to DFU restore a Macbook on port 1 of USBHub3c.

2.5.1 DFU Automator Features

- Send DFU Mode or Reboot VDMs to a MacBook
- Send arbitrary VDMs (Hexstring input)
- Enable or disable hub ports
- Retrieve the ECID of an attached Apple device

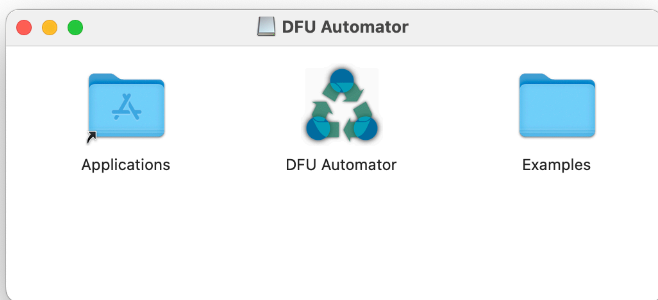
Installation and Setup

Install Apple Configurator

- Open the **App Store** on your Mac
- Search for **Apple Configurator**
- Click **Install**

Install DFU Automator

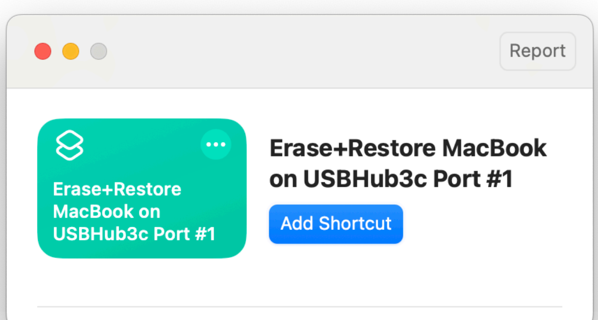
1. *Download DFU Automator* from the official source
2. Open the downloaded *.dmg* file



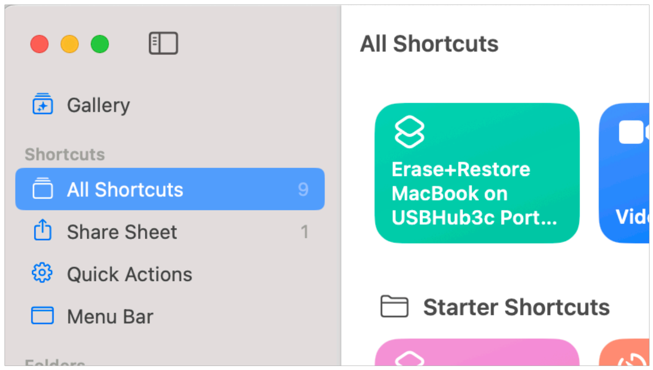
3. Drag **DFU Automator** into the **Applications** folder
4. Open DFU Automator from the **Applications** folder once to allow Shortcuts to recognize it

Adding the Example Shortcut

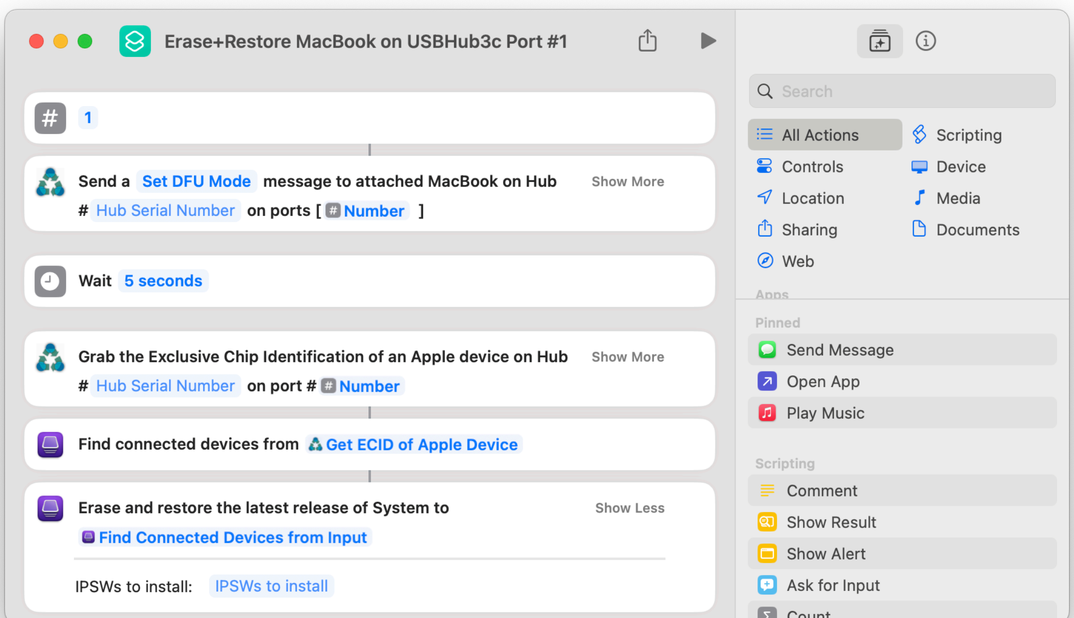
1. In the **Examples Folder**, open the shortcut titled **Erase+Restore MacBook on USBHub3c Port #1**



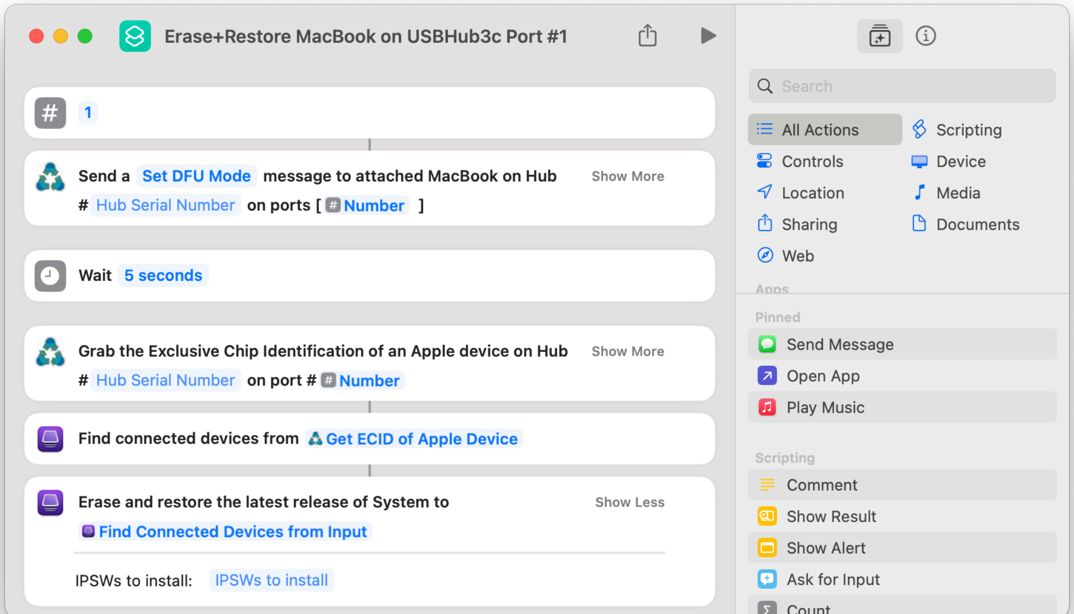
2. Click **Add Shortcut**
3. The shortcut will now appear in the **Shortcuts app** under **All Shortcuts**



4. Click the shortcut's icon (not the play button that appears on hover) to view the workflow:



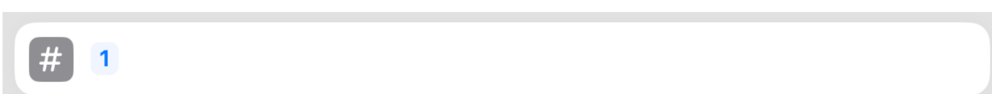
Shortcut Example



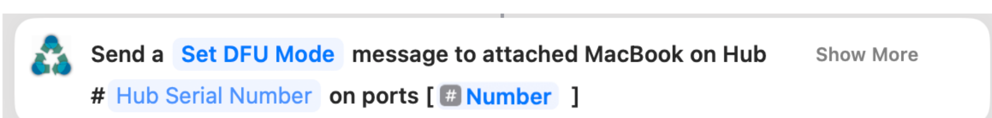
Shortcut Workflow Step-by-Step

To understand each step of the shortcut, **Control-click** an action and select **Show Info**.

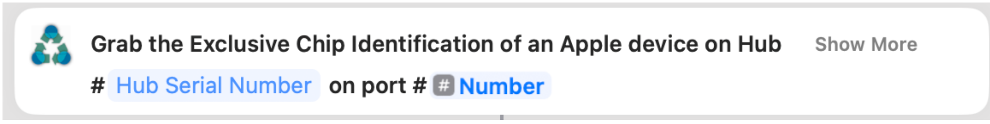
Key Actions



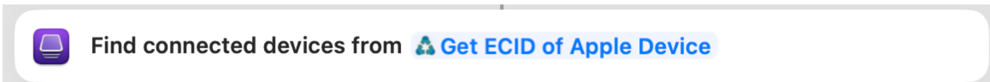
- Sets the USBHub3c port number for the MacBook to be updated (set to 1 by default)



- Sends a *Set DFU Mode* message on hub # *Hub Serial Number* on ports [# *Number*]
  - Tells DFU Automator to have USBHub3c send a Vendor-Defined-Message to put a MacBook on port [# *Number*] into DFU mode
  - If only one hub connected, *Hub Serial Number* can be blank
- Waits 5 seconds to give the MacBook time to enter DFU Mode



- Tells DFU Automator to read the ECID of the attached MacBook on the specified port



- Tells Apple Configurator to convert the ECID to a *Connected Device Entity* for use in other Apple Configurator shortcut actions
- Erases and restores the connected devices

> **Warning:** All data and settings will be removed!

- *IPSWs to install*: Select additional IPSW files to install on connected devices. Configurator will choose the correct IPSW for the device being updated

Example Usage

To use the example shortcut to DFU-restore a MacBook:

1. Locate the DFU-enabled USB-C port on the target Mac:

   Apple Silicon MacBooks: Left side, towards the hinge

   Intel MacBooks with T2 chip: Left side, away from the hinge

   Desktops: Refer to Apple Support[76] for more details

2. Connect the host Mac to **USBHub3c Port 0**
3. Connect the target MacBook's DFU-enabled port to **USBHub3c Port 1**

> **Warning:** Target MacBook will be erased!

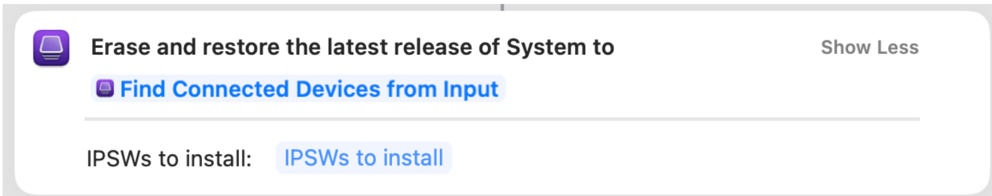
4. In **Shortcuts** , click the **play button** in the upper right of the shortcut icon or the expanded view of the shortcut
5. Click "Allow" to let the shortcut run actions

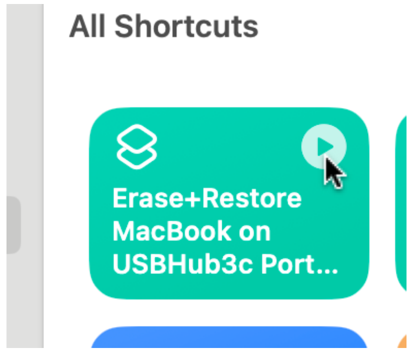
- The target MacBook will chime and boot into DFU mode with a black screen

6. Approve any other privacy popups that appear:

- Allow configurator to find devices
- Allow the shortcut share the device ECID with Apple Configurator

[76] https://support.apple.com/en-us/108900





8. Apple Configurator will erase and restore the MacBook on port 1

Restoring Multiple Devices Simultaneously

Shortcuts and Configurator actions can be run in parallel, allowing multiple MacBooks to be restored asynchronously. The restore shortcut can be copied and the port number can be set for each port. To use, connect a MacBook to an available USBHub3c port and manually trigger the corresponding restore shortcut matching the port number.

More information about using Apple Configurator with Shortcuts can be found here[77].

HubTool is a utility that lets users view detailed information and control settings of Acroname devices.

BrainD is a desktop service that exposes an HTTP REST interface to connected BrainStem Devices.

Control-Room

[77] https://support.apple.com/en-au/guide/apple-configurator-mac/acm12a6f7b56/mac

is a web browser application built on BrainD, designed for audio/video applications and USB device diagnostics.

Q-Sys plugin interface for BrainStem Devices to be used in the Q-Sys Designer Software for audio/video applications.

DFU Automator is a helper application that enables Apple Shortcuts to control USBHub3c's Device Firmware Update (DFU) mode operations for Apple devices.

3 API Reference

This API reference is organized by programming language. You will find product specific documentation in the *products* section.

3.1 BrainStem Entities

3.1.1 Analog Entity

API Documentation: *[cpp] [python] [.NET] [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 << 4 = 0xFFF0) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

Setting Values

The reading resolution will return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the value sent by the API will get propagated up to 16 bits wide.

For example, if a 10-bit DAC engine exists in the target module's hardware, the reading will get down shifted 5 bits to derive the 10 bit value (0x8000 =: 0x8000 >> 5 = 0x0400) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

Configuration (Get/Set)

```
analog [ index ] . getConfiguration <= (unsigned char) configuration
analog [ index ] . setConfiguration => (unsigned char) configuration
```

Getting Configuration

Some analog entities may be single purpose functionality or can be configured for multiple different behaviors depending on the hardware. Configuration information includes whether the entities is an input only, output only, or can be configured as either and input or output.

Setting Configuration

Analog entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most analog entities are typically as inputs, but will vary by module hardware.

Code Examples

C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

stem.analog[0].getValue(value); // gets the value of A2D channel 0 into variable value
stem.analog[3].setValue(1234); // sets the DAC on channel 3 to a value of 1234
stem.analog[0].setConfiguration(analogConfigurationInput);
```

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
```

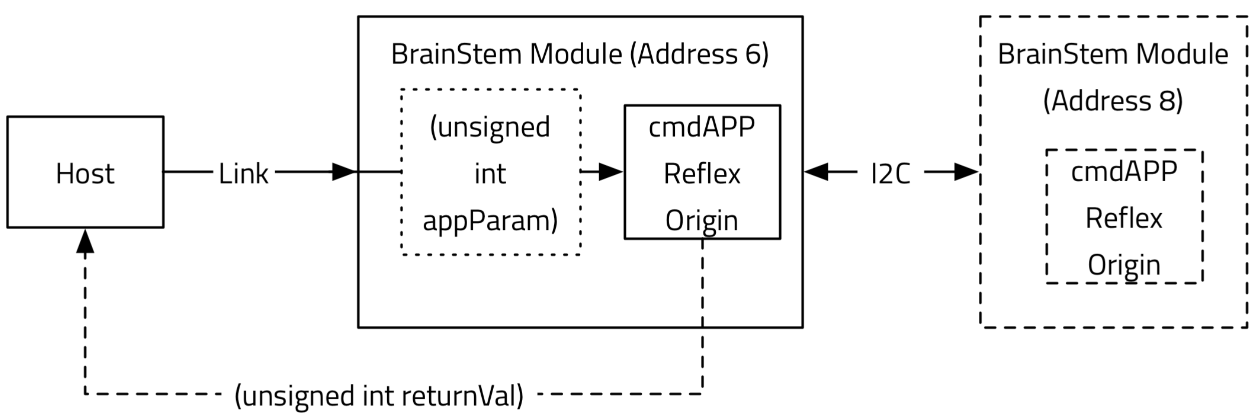
Python

```
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
```

3.1.2 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 comming in future release.
```

3.1.3 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);
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);
```

3.1.4 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
```

3.1.5 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);
```

3.1.6 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
```

3.1.7 Mux Entity

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 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 mux entities can simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

Channel (Set/Get)

```
mux [ index ] . setChannel => (unsigned char) channel
mux [ index ] . getChannel <= (unsigned char) channel
```

Gets/Sets the currently selected channel

Enable/Disable (Set/Get)

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

Enables/Disables the mux.

Get Channel Voltage (Get)

```
mux [ index ] . getChannelVoltage <= ((unsigned char) channel, (unsigned char)␣
↪voltage)
```

Returns the voltage of the supplied channel.

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);
err = stem.mux[0].setEnable(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].setEnable(1);
stem.mux[0].setEnable(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].setEnable(1)
err = stem.mux[0].setEnable(0)
voltage = stem.mux[0].getChannel(1)
print voltage.value
err = stem.mux[0].setChannel(3)
```

3.1.8 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);
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()
```

(continues on next page)

(continued from previous page)

```
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)
```

3.1.9 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.

Port Name (Get/Set)

```
port [index] . getName <= (unsigned char[]) name
port [index] . setName => (unsigned char[]) name
```

Allows for setting a friendly name to the port with a 32 character limit.

Downstream Data Speed (Get)

```
port [index] . getDataSpeed <= (unsigned int)
```

Gets the speed of the enumerated device.

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

3.1.10 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 POD'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 possiable 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

3.1.11 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 if 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].setEnable(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
```

3.1.12 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].setEnable(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].setEnable(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].setEnable(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].setEnable(1)
#Set servo to middle/neutral position
err = stem.servo[0].getPosition(&pPosition)
```

3.1.13 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].setEnable(1);
err = stem.relay[1].setEnable(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].setEnable(0);
err = stem.relay[1].setEnable(0);
```

Python

```
err = stem.relay[0].setEnable(1);
err = stem.relay[1].setEnable(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].setEnable(0);
err = stem.relay[1].setEnable(0);
```

3.1.14 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.

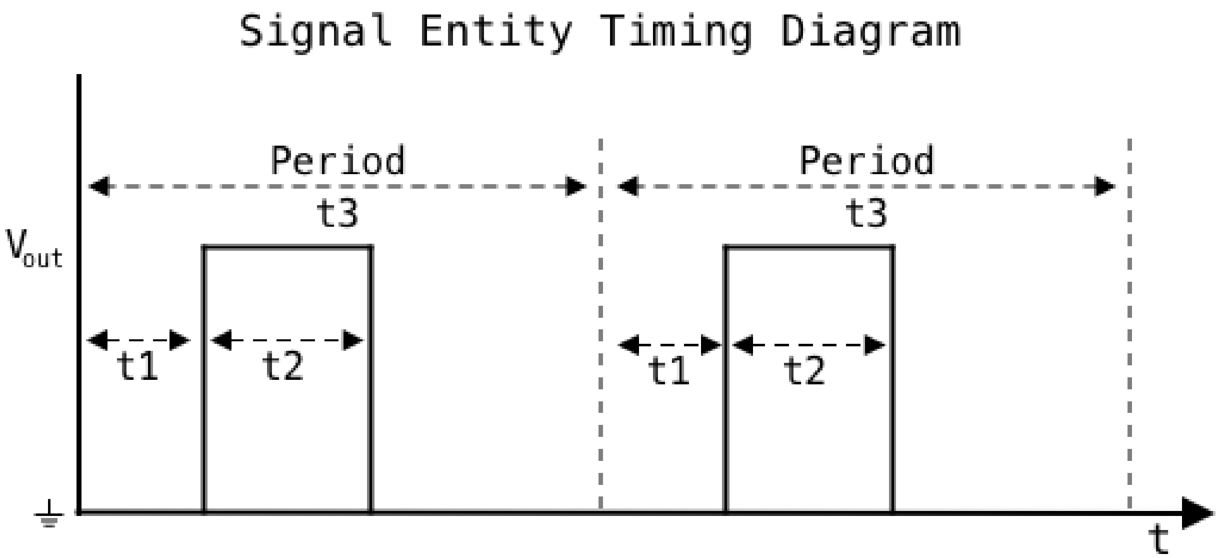
Signal entity to Digital entity mapping varies from device to device. Please refer to the datasheet.

Timing

Set/Get enable

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

Enables the Signal Entity for a given index.



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].setEnable(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].setEnable(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;
```

3.1.15 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
```

3.1.16 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.

LED Brightness (Get/Set)

```
system . setLEDMaxBrightness => (unsigned char) value
system . getLEDMaxBrightness <= (unsigned char) value
```

Gets or Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels.

Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Boot Slot (Get/Set)

```
system . setBootSlot => (unsigned char) slot
system . getBootSlot <= (unsigned char) slot
```

BrainStem modules can be configured to enable a reflex file at boot. The reflex file must be loaded into a slot in the internal store. Setting the boot slot to the value 255 will disable on boot functionality. For more information about stores and slots please see the store section of the reference manual. For more information about reflexes please see the Reflex section of the manual.

Input Voltage (Get)

```
system . getInputVoltage <= (unsigned int) inputVoltage
```

The input voltage system command is a read only command and will return the input supply voltage of the BrainStem module in micro volts.

Serial Number (Get)

```
system . getSerialNumber <= (unsigned int) serialNumber
```

Read only command that returns the unique module serial number. The returned value is an unsigned int. In Acroname UI applications the serial number is generally represented as an 8 character Hexadecimal number.

BrainStem Model (Get)

```
system . getModel <= (unsigned char) BrainStem model.
```

Read only command that returns the model of the BrainStem module.

Hardware Version (Get)

```
system . getHardwareVersion <= (unsigned int) Hardware Version.
```

Read only command that returns the hardware version of the module. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

Version (Get)

```
system . getVersion <= (unsigned int) version number.
```

Read only command that returns the version number of the BrainStem firmware. This is a packed format. The `aVersion.h` C API can represent this version in a human readable manner. The format of the version number is 3 digits separated by `.`.

```
major . minor . patch
```

Module Address (Get)

```
system . getModule <= (unsigned char) module
```

The module address is the number used to address the module on the BrainStem network and from the host. This is a combination of the module base address, any software offset that is applied and any hardware module offset.

Module Base Address (Get)

```
system . getModuleBaseAddress <= (unsigned char) module
```

The module base address is the default or base address of the module, before any offsets are applied.

Module Software Offset (Set/Get)

```
system . getModuleSoftwareOffset <= (unsigned char) software offset
system . setModuleSoftwareOffset => (unsigned char) software offset
```

The module software offset is added to the module's base address and any hardware offsets to determine the final module address of the module. This setting is not applied until saved and the module has been reset.

Module Hardware Offset (Get)

```
system . getModuleHardwareOffset <= (unsigned char) module hardware offset
```

MTM BrainStems have a set of module offset pins which will adjust the module address via hardware. See the data sheet for your MTM module for more information about these hardware settings. The module offset command is a read only command that returns the offset that will be added to the base module address and any software offset to determine the operating address of the MTM BrainStem module. Changes to the hardware offset are applied when the Device is reset.

Router Address (Get/Set)

```
system . setRouter => (unsigned char) module
system . getRouter <= (unsigned char) module
```

The BrainStem router address refers to the BrainStem module address of the module that will coordinate communication with the host system. This setting is not applied until it is saved and the module has been reset.

Changing the router address can have negative consequences for communicating with the BrainStem network. Please see the appendix on the BrainStem Network setup for more information.

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

HeartBeat Interval (Get/Set)

```
system . setHBInterval => (unsigned char) interval
system . getHBInterval <= (unsigned char) interval
```

Gets or sets the heartbeat interval to control the amount of heartbeat traffic. This value is set at approximately 1/50th of a second resolution. Heartbeat packets are handled by the underlying system, and are indicated on the brainstem by the blinking green heartbeat LED. UI applications also have Heartbeat indicators. Default value is 12.

System Name (Get/Set)

```
system . getName <= (unsigned char[]) name
system . setName => (unsigned char[]) name
```

Allows for setting a friendly name to the device with a 32 character limit.

Code Examples

C++

```
// Get requests fill the parameter with the current system value upon success.
// All commands return aErr values when errors are encountered and aErrNone on
// success.

stem.system.save();
stem.system.reset();
stem.system.setLED(1);
stem.system.getLED(state);
stem.system.setLEDMaxBrightness(255);
stem.system.getLEDMaxBrightness(value);
stem.system.setBootSlot(5);
stem.system.getBootSlot(slot);
stem.system.getInputVoltage(voltage);
stem.system.getModule(address);
stem.system.getRouter(address);
stem.system.setRouter(6);
stem.system.getModuleBaseAddress(address);
stem.system.setModuleSoftwareOffset(16);
stem.system.getModuleSoftwareOffset(offset);
stem.system.getModuleHardwareOffset(offset);
stem.system.getSerialNumber(serialNumber);
stem.system.getModel(model);
stem.system.getHardwareVersion(hardwareVersion);
stem.system.getVersion(version);
stem.system.getHBInterval(interval);
stem.system.setHBInterval(interval);
```

Python

```
stem.system.save()
stem.system.reset()
stem.system.setLED(1)
```

(continues on next page)

(continued from previous page)

```
state = stem.system.getLED()
print state.value
stem.system.setLEDMaxBrightness(255);
brightness = stem.system.getLEDMaxBrightness();
print brightness.value
stem.system.setBootSlot(5)
slot = stem.system.getBootSlot()
print slot.value
inputVoltage = stem.system.getInputVoltage()
print inputVoltage.value
module = stem.system.getModule()
print module.value
address = stem.system.getModuleBaseAddress();
print address.value
stem.system.setModuleSoftwareOffset(16);
offset = stem.system.getModuleSoftwareOffset();
print offset.value
offset = stem.system.getModuleHardwareOffset();
print offset.value
serialNumber = stem.system.getSerialNumber()
print serialNumber.value
model = stem.system.getModel()
hardwareVersion = stem.system.getHardwareVersion()
print hardwareVersion.value
version = stem.system.getVersion()
print brainstem.version.get_version_string(version.value)
hbInterval = stem.system.getHBInterval()
print hbInterval.value
stem.system.setHBInterval(12)
```

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);
```

3.1.17 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

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

3.1.18 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 it 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.
```

3.1.19 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);
```

3.1.20 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. The mode parameter can be defined as the following:

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

Upstream State (Get)

```
usb . getUpstreamState <= (unsigned char) state
```

Gets the upstream switch state for the USB upstream ports. Returns none if no ports are plugged in, port 0 if the mode is set correctly and a cable is plugged into port 0, and port 1 if the mode is set correctly and a cable is plugged into port 1

Enumeration Delay (Get/Set)

```
usb . getEnumerationDelay <= (unsigned int) ms_delay
usb . setEnumerationDelay => (unsigned int) ms_delay
```

Gets/Sets the inter-port enumeration delay in milliseconds. The enumeration delay sequentially enables data and power to downstream ports after the defined delay time. After setting and saving this parameter all downstream ports will be initially disabled upon system power-on or reset. Similarly, if there is no upstream connection, all downstream ports will be disabled. When an upstream connection is applied, or after the system boots, the system will wait for the defined delay time and enable the lowest port number. The system will then wait for the defined delay time and then enable the next highest port. This behavior repeats until all ports are enabled.

Inconsistent behavior from race conditions may occur if enumeration delay is used in conjunction with Reflex programs which also manipulate the downstream port states. Care should be taken to ensure no conflicts between the enumeration delay and Reflex programs.

Note: This setting should be saved with a stem.system.save() call.

Upstream Boost Mode (Get/Set)

```
usb . getUpstreamBoostMode <= (unsigned char) setting
usb . setUpstreamBoostMode => (unsigned char) setting
```

Gets/Sets the upstream boost mode. Boost mode increase the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Modes: 0 = no boost, 1 = 4% boost, 2 = 8% boost, 3 = 12% boost.

Note: This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Down Stream Boost Mode (Get/Set)

```
usb . getDownstreamBoostMode <= (unsigned char) setting
usb . setDownstreamBoostMode => (unsigned char) setting
```

Gets/Sets the Downstream boost mode. Boost mode increase the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Modes: 0 = no boost, 1 = 4% boost, 2 = 8% boost, 3 = 12% boost.

Note: This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Port Current Limit (Get/Set)

```
usb . setPortCurrentLimit  => (unsigned char) channel, (unsigned int) microamps
usb . getPortCurrentLimit (unsigned char) channel <=  (unsigned int) microamps
```

Gets/Sets the current limit for the downstream channel. There are a number of settings for current limits ranging from 100 mAmps to 2.5 amps. See the USB hub datasheet for specific settings information.

Port Mode setting (Get/Set)

```
usb . setPortMode => (unsigned char) channel, (unsigned char) mode
usb . getPortMode (unsigned char) channel <= (unsigned char) mode
```

Gets/Sets the Port mode for the channel specified. The portmode is a bitmapped setting. Device specific mode options are listed in the data-sheet. There is a unifed 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
...
```

3.1.21 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.

Upstream Connection HighSpeed (Get/Set)

```
usbsystem . setUpstreamHS => (unsigned char) enable
usbsystem . getUpstreamHS <= (unsigned char) enable
```

Many acroname products have multiple upstream port selections, some even have the ability to move just the HighSpeed or SuperSpeed signals. This function is used to access and control that functionality for the HighSpeed signals only.

Upstream Connection SuperSpeed (Get/Set)

```
usbsystem . setUpstreamSS => (unsigned char) enable
usbsystem . getUpstreamSS <= (unsigned char) enable
```

Many acroname products have multiple upstream port selections, some even have the ability to move just the HighSpeed or SuperSpeed signals. This function is used to access and control that functionality for the SuperSpeed signals only.

Enumeration Delay (Get/Set)

```
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 lover level port modes. It is equivalent to calling get/set mode from the *PortClass* on all ports at once.

State List (Get)

```
usbsystem . getModeList <= (unsigned int [NUM_PORTS]) list
usbsystem . setModeList => (unsigned int [NUM_PORTS]) list
```

The state list function gives you access and control to all lover 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.

Data HighSpeed Max Datarate (Get/Set)

```
usbsystem . setDataHSMaxDatarate <= (unsigned int) config
usbsystem . getDataHSMaxDatarate => (unsigned int) config
```

The Max Datarate APIs will limit the device to a maximum specified datarate for the specific signal set. This API modifies the max datarate on the USB HighSpeed signals.

Enumeration	Name	Description
0	None	Configure HighSpeed Signals to no connection
1	Low Speed	Configure HighSpeed Signals to a maximum datarate of 1.5Mbps
2	Full Speed	Configure HighSpeed Signals to a maximum datarate of 12Mbps
3	High Speed	Configure HighSpeed Signals to a maximum datarate of 480Mbps

Data SuperSpeed Max Datarate (Get/Set)

```
usbsystem . setDataSSMaxDatarate <= (unsigned int) config
usbsystem . getDataSSMaxDatarate => (unsigned int) config
```

The Max Datarate APIs will limit the device to a maximum specified datarate for the specific signal set. This API modifies the max datarate on the USB SuperSpeed signals.

Enumeration	Name	Description
0	None	Configure SuperSpeed Signals to no connection
1	Super Speed	Configure SuperSpeed Signals to a maximum datarate of 5Gbps
2	Super Speed Plus	Configure SuperSpeed Signals to a maximum datarate of 10Gbps

Override (Get/Set)

```
usbsystem . getOverride <= (unsigned int) config
usbsystem . setOverride => (unsigned int) config
```

The system inherently goes towards compliant behavior, in some conditions you may not want compliant behavior and this is what the override bit field allows. There are the following override bits that can be set.

Bit	Name	Description
0	Auto Vbus Toggle Disable	This bit is used to disable the auto vbus toggle behavior on re-enumeration of the upstream port.
1	Vbus Detect Disable	This bit is used to disable the requirement of an upstream connection for enabling the hub chip.

3.2 Python API Reference

Welcome to the BrainStem Python API reference documentation. This documentation covers the Python Acroname BrainStem module. This reference assumes that you understand the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation.

- *BrainStem Overview*
- *BrainStem Terminology*
- *Getting Started with the BrainStem.*

Next check out the python *Getting Started* section.

3.2.1 Getting (Quickly) Started

The BrainStem python package allows you to interact with a collection of BrainStem modules from python. The API is similar to both the C++ and Reflex API's, with a few significant differences. The remainder of this section details the structure and functionality of the python API.

Most modern operating systems come with all the tools needed to immediately install the BrainStem python libraries and create python based applications. As such, simply download the latest development package[78], 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

[78] https://acroname.com/software/brainstem-development-kit

libffi

The Brainstem python library relies on libffi, on macOS and Windows this is generally available via pip. On Linux you may need to install libffi via your distro's package manager.

Python development headers

Also on Linux, you may need to install the development package for python via your distro's package manager before you can install.

CentOS package manager

On CentOS and yum based distros the following command will install the required packages.

```
$> sudo yum install libffi-devel python-devel
```

Installation

Install the python package.

Note: '#>' indicates that the command must be run with admin privileges on MacOS and Linux, either via sudo or su.

```
#> pip install brainstem-*.whl
```

If you need to uninstall the library, the easiest way to do so is with pip.

```
$> pip uninstall brainstem
```

A Tour of the Python Example

To run the example, go to Development/python in the "BrainStem2 Development Kit" package and type:

```
$> python brainstem_example.py
```

The example requires that you have a USB BrainStem link module connected to your host computer. If you see the following message, you probably don't have a module connected:

```
Creating USB stem and connecting to first module found
Could not find a module.
```

Once the example starts running, it will connect to the first USBStem it finds connected to your computer and then blink the user LED on the module.

```
$> python brainstem_example.py
Creating USB stem and connecting to first module found
Connecting to Module with serial number: 0x40F5849A
Flashing the user LED
```

The following is a brief introduction interacting with the brainstem via the python interactive interpreter. The first step is to import some modules that we'll need later. There are multiple ways to import the brainstem package. For this example we will use the simplest method.

```
>>> import brainstem
```

See the *Package Structure <package>* section of the python reference for more information about the brainstem package, and the modules it includes.

Next we discover a USBStem module, and connect to it.

```
>>> spec = brainstem.discover.findFirstModule(brainstem.link.Spec.USB)
>>> print spec
LinkType: USB(serial: 0x40F5849A, module: 0)
>>> stem = brainstem.stem.USBStem()
>>> stem.connect(0x40F5849A)
```

Information about specific modules can be found in the *Modules <Modules>* section.

Now that we have created a *USBStem*, we can turn on the user LED using the *system* entity:

```
>>> stem.system.setLED(1)
```

Finally lets blink the LED in a loop.

```
>>> from time import sleep
>>> for i in range(0,100):
...     err = stem.system.setLED(i % 2)
...     if err != 0:
...         print "error %d"% err
...         break
...     sleep(0.5)
...
>>>
```

As you can see the call to setLED returns an error value. In this case that is an error value, that will be 0 on success and some other number if there is an error. The brainstem library generally avoids raising exceptions, and instead passes information via result objects, or result error codes. More information about these errors, and the result object can be found in the *Result <result>* section of the python reference

Help is available from within the python interpreter, calling help() on a stem or other object will yield context specific documentation.

```
>>> import brainstem
>>> help(brainstem.stem.USBStem)
Help on class USBStem in module brainstem.stem:

class USBStem(brainstem.module.Module)
|  Concrete Module implementation for 40Pin and MTM USBStem modules
|
|  USBStem modules contain Analogs, Digital IO's, and I2C entities
|  in addition to the system entity.
|
|  Method resolution order:
|      USBStem
...
```

Enjoy!

The Acroname Team.

Support

If you are having issues, please let us know. We have a mailing list located at: support@acroname.com

3.2.2 Acroname Modules

Quick Access:

- *USBHub3c*
- *USBHub3p*
- *USBHub2x4*
- *USBCSwitch*
- *MTMDAQ2*
- *MTMEtherStem*
- *MTMIOSerial*
- *MTMLOAD1*
- *MTMPM1*
- *MTMRelay*
- *MTMUSBStem*
- *MTMDAQ1*
- *EtherStem*
- *USBStem*

Each type of BrainStem module is represented by a corresponding concrete Module implementation. The following classes are instantiated to allow communication through to the corresponding BrainStem module hardware.

The instantiation and subsequent connection to each module is as follows

```
>>> stem = USBStem()
# 0xXXXXXXXX is the serial number of the module.
>>> stem.connect(0xXXXXXXXX)
```

Connecting to the BrainStem module can take multiple forms, the simplest way to connect when you know the module's serial number is to call connect with the serial number, as in the above code snippet. If you don't know the serial number of the module, you can perform a discovery of the modules currently connected and print that information. For details of the connection functions API please see *Connections* in this reference.

USBHub3c

class brainstem.stem.**USBHub3c**(*address=6*, *enable_auto_networking=True*, *model=24*)

Concrete Module implementation for the USBHub3c.

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

Entities:

- system
- app[0-3]
- pointers[0-3]
- store[0-2]
- temperature[0-2]
- timer[0-7]
- hub
- hub.port[0-7]
- rail[0-6]
- pd[0-7]
- usb
- uart

Useful Constants:

- BASE_ADDRESS (6)
- NUMBER_OF_STORES (2)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_USB_PORTS (8)
- NUMBER_OF_RAILS (7)
- STORE_INTERNAL_INDEX (0)
- STORE_RAM_INDEX (1)
- STORE_EEPROM_INDEX (2)
- PORT_ID_CONTROL_INDEX (6)
- PORT_ID_POWER_C_INDEX (7)
- NUMBER_OF_PORTS (8)

class Hub(*module*, *index*)

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

i2c

usb entity adds minimal legacy support

Back to the top

USBHub3p

class brainstem.stem.**USBHub3p**(*address=6*, *enable_auto_networking=True*, *model=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)
- NUMBER_OF_PORTS (12)

Bit defines for port state UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if (state & brainstem.BIT(aUSBHUB3P_USB_VBUS_ENABLED))

- aUSBHUB3P_USB_VBUS_ENABLED (0)

- aUSBHUB3P_USB2_DATA_ENABLED (1)
- aUSBHUB3P_USB3_DATA_ENABLED (3)
- aUSBHUB3P_USB_SPEED_USB2 (11)
- aUSBHUB3P_USB_SPEED_USB3 (12)
- aUSBHUB3P_USB_ERROR_FLAG (19)
- aUSBHUB3P_USB2_BOOST_ENABLED (20)
- aUSBHUB3P_DEVICE_ATTACHED (23)

Bit defines for port error UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if (error & brainstem.BIT(aUSBHUB3P_ERROR_VBUS_OVERCURRENT))

- aUSBHUB3P_ERROR_VBUS_OVERCURRENT (0)
- aUSBHUB3P_ERROR_VBUS_BACKDRIVE (1)
- aUSBHUB3P_ERROR_HUB_POWER (2)
- aUSBHUB3P_ERROR_OVER_TEMPERATURE (3)

class Hub(*module*, *index*)

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

USBHub2x4

class brainstem.stem.**USBHub2x4**(*address=6*, *enable_auto_networking=True*, *model=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)
- NUMBER_OF_PORTS (6)

Bit defines for port error UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if (error & brainstem.BIT(aUSBHUB2X4_USB_VBUS_ENABLED))

- aUSBHUB2X4_USB_VBUS_ENABLED (0)
- aUSBHUB2X4_USB2_DATA_ENABLED (1)
- aUSBHUB2X4_USB_ERROR_FLAG (19)
- aUSBHUB2X4_USB2_BOOST_ENABLED (20)
- aUSBHUB2X4_DEVICE_ATTACHED (23)
- aUSBHUB2X4_CONSTANT_CURRENT (24)

Bit defines for port error UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if (error & brainstem.BIT(aUSBHUB3P_ERROR_VBUS_OVERCURRENT))

- aUSBHUB2X4_ERROR_VBUS_OVERCURRENT (0)
- aUSBHUB2X4_ERROR_OVER_TEMPERATURE (3)
- aUSBHub2X4_ERROR_DISCHARGE (4)

class Hub(*module*, *index*)

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

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*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMDAQ2

class brainstem.stem.**MTMDAQ2** (*address=10*, *enable_auto_networking=True*, *model=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 (10)
- NUMBER_OF_STORES (2)
- NUMBER_OF_INTERNAL_SLOTS (12)
- NUMBER_OF_RAM_SLOTS (1)
- NUMBER_OF_DIGITALS (2)
- NUMBER_OF_ANALOGS (20)
- NUMBER_OF_I2C (1)
- NUMBER_OF_POINTERS (4)
- NUMBER_OF_TIMERS (8)
- NUMBER_OF_APPS (4)
- ANALOG_RANGE_P0V064N0V064 (0)
- ANALOG_RANGE_P0V64N0V64 (1)
- ANALOG_RANGE_P0V128N0V128 (2)
- ANALOG_RANGE_P1V28N1V28 (3)
- ANALOG_RANGE_P1V28N0V0 (4)
- ANALOG_RANGE_P0V256N0V256 (5)
- ANALOG_RANGE_P2V56N2V56 (6)
- ANALOG_RANGE_P2V56N0V0 (7)
- ANALOG_RANGE_P0V512N0V512 (8)
- ANALOG_RANGE_P5V12N5V12 (9)
- ANALOG_RANGE_P5V12N0V0 (10)
- ANALOG_RANGE_P1V024N1V024 (11)
- ANALOG_RANGE_P10V24N10V24 (12)
- ANALOG_RANGE_P10V24N0V0 (13)
- ANALOG_RANGE_P2V048N0V0 (14)
- ANALOG_RANGE_P4V096N0V0 (15)
- ANALOG_BULK_CAPTURE_MAX_HZ (500000)
- ANALOG_BULK_CAPTURE_MIN_HZ (1)

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMEtherStem

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMIOSerial

class brainstem.stem.**MTMIOSerial** (*address=8*, *enable_auto_networking=True*, *model=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)
- NUMBER_OF_USB (1)
- NUMBER_OF_USB_PORTS (4)
- NUMBER_OF_PORTS (5)
- aMTMIOSERIAL_USB_VBUS_ENABLED (0)
- aMTMIOSERIAL_USB2_DATA_ENABLED (1)
- aMTMIOSERIAL_USB_ERROR_FLAG (19)
- aMTMIOSERIAL_USB2_BOOST_ENABLED (20)
- aMTMIOSERIAL_ERROR_VBUS_OVERCURRENT (0)

class Hub(*module*, *index*)

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMLOAD1

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMPM1

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMRelay

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMUSBStem

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

MTMDAQ1

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

EtherStem

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

USBStem

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

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

Returns

An error result from the list of defined error codes in brainstem.result

Back to the top

3.2.3 Package Structure

The BrainStem package consists of a number of modules, which together form the BrainStem python API.

brainstem.module

A module that provides base classes for BrainStem Modules and Entities.

The Module and Entity classes are designed to be extended for specific types of BrainStem Modules and Entities. For more information about Brainstem Modules and Entities, please see the Terminology[79] section of the Acroname BrainStem Reference[80]

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[81] section of the Acroname BrainStem Reference[82]

79 https://acroname.com/reference/brainstem/terms.html
80 https://acroname.com/reference
81 https://acroname.com/reference/api/entities
82 https://acroname.com/reference

brainstem.link

A module that provides a Spec class for specifying a connection to a BrainStem module.

A Spec instance fully describes a connection to a brainstem module. In the case of USB based stems this is simply the serial number of the module. For TCPIP based stems this is an IP address and TCP port.

For more information about links and the Brainstem network see the Acroname BrainStem Reference[83]

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[84]

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[85]

[83] https://acroname.com/reference
[84] https://acroname.com/reference
[85] https://acroname.com/reference

brainstem.version

Provides version access utilities.

3.2.4 Analog

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

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.

getBulkCaptureNumberOfSamples()

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

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getBulkCaptureSampleRate()

Get the current sample rate setting for this analog when bulk capturing. upon success filled with current sample rate in samples per second (Hertz).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getBulkCaptureState()

Get the current bulk capture state for this analog. the state of bulk capture. - Idle: bulkCaptureIdle = 0 - Pending: bulkCapturePending = 1 - Finished: bulkCaptureFinished = 2 - Error: bulkCaptureError = 3

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getConfiguration()

Get the analog configuration. - Current configuration of the analog entity.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getEnable()

Get the analog output enable status. 0 if disabled 1 if enabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getRange()

Get the analog input range. 8 bit value corresponding to a discrete range option

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getValue()

Get the raw ADC output value in bits. 16 bit analog reading with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference. Note: Not all modules are provide 16 useful bits; this value's least significant bits are zero-padded to 16 bits. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVoltage()

Get the scaled micro volt value with reference to ground. 32 bit signed integer (in microvolts) based on the board's ground and reference voltages. Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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
An error result from the list of defined error codes in brainstem.result

Return type
unsigned byte

setBulkCaptureNumberOfSamples(*value*)

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

Parameters
value (*const unsigned int*) – number of samples. Minimum # of Samples: 0 Maximum # of Samples: (BRAINSTEM_RAM_SLOT_SIZE / 2) = (3FFF / 2) = 1FFF = 8191

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

Return type
unsigned byte

setBulkCaptureSampleRate(*value*)

Set the sample rate for this analog when bulk capturing.

Parameters
value (*const unsigned int*) – sample rate in samples per second (Hertz). Minimum rate: 7,000 Hz Maximum rate: 200,000 Hz

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

Return type
unsigned byte

setConfiguration(*configuration*)

Set the analog configuration. aErrConfiguration - Entity does not support this configuration.

Parameters
configuration (*const unsigned char*) –

- bitAnalogConfigurationOutput configures the analog entity as an output.

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

Return type
unsigned byte

setEnable(*enable*)

Set the analog output enable state.

Parameters
enable (*const unsigned char*) – set 1 to enable or 0 to disable.

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

Return type
unsigned byte

setRange(*range*)

Set the analog input range.

Parameters
range (*const unsigned char*) – 8 bit value corresponding to a discrete range option

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

Return type
unsigned byte

setValue(*value*)

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

Parameters
value (*const unsigned short*) – 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference. Note: Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

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

Return type
unsigned byte

setVoltage(*microvolts*)

Set the voltage level of an analog output (DAC) in microvolts.

Parameters
microvolts (*const int*) – 32 bit signed integer (in microvolts) based on

the board's ground and reference voltages. Note: Voltage range is dependent on the specific DAC channel range.

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

Return type
unsigned byte

3.2.5 App

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

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.

`execute`(*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. aErrNone success. aErrTimeout The request timed out waiting to start execution. aErrConnection No active link connection. aErrNotFound the app reflex was not found or not enabled on the module.

Parameters
appParam (*const unsigned int*) – The app parameter handed to the reflex.

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

Return type
unsigned byte

`executeAndReturn`(*appParam*, *msTimeout*)

Execute the app reflex on the module. Wait for a return from the reflex execution for msTimoue milliseconds. This method will block for up to msTimeout. The return value filled in from the result of executing the reflex routine. aErrNone success. aErrTimeout The request timed out waiting for a response. aErrConnection No active link connection. aErrNotFound the app reflex was not found or not enabled on the module.

Parameters
- **appParam** (*const unsigned int*) – The app parameter handed to the reflex.
- **msTimeout** (*const unsigned int*) – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

3.2.6 Clock

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

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.

`getDay`()

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month). The two digit day portion of the real-time clock value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getHour`()

Get the two digit hour value (0-23). The two digit hour portion of the real-time clock value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getMinute`()

Get the two digit minute value (0-59). The two digit minute portion of the real-time clock value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getMonth`()

Get the two digit month value (1-12). The two digit month portion of the real-time clock value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getSecond`()

Get the two digit second value (0-59). The two digit second portion of the real-time clock value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getYear`()

Get the four digit year value (0-4095). Get the year portion of the real-time clock value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`setDay`(*day*)

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Parameters
day (*const unsigned char*) – The two digit day portion of the real-time clock value.

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

Return type
unsigned byte

`setHour`(*hour*)

Set the two digit hour value (0-23).

Parameters
hour (*const unsigned char*) – The two digit hour portion of the real-time clock value.

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

Return type
unsigned byte

`setMinute`(*min*)

Set the two digit minute value (0-59).

Parameters
min (*const unsigned char*) – The two digit minute portion of the real-time clock value.

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

Return type
unsigned byte

`setMonth`(*month*)

Set the two digit month value (1-12).

Parameters
month (*const unsigned char*) – The two digit month portion of the real-time clock value.

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

Return type
unsigned byte

`setSecond`(*sec*)

Set the two digit second value (0-59).

Parameters
sec (*const unsigned char*) – The two digit second portion of the real-time clock value.

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

Return type
unsigned byte

`setYear`(*year*)

Set the four digit year value (0-4095).

Parameters
year (*const unsigned short*) – Set the year portion of the real-time clock value.

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

Return type
unsigned byte

3.2.7 Definitions

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

brainstem.defs.**model_info**(*model*)
Get Model information.

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

Returns
String containing model information.

brainstem.defs.**model_name**(*model*)
Get Model Name.

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

Returns
A string containing model name.

3.2.8 Digital

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

getConfiguration()
Get the digital configuration. - Current configuration of the digital entity.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getState()
Get the 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getStateAll()

Gets the logical state of all available digitals in a bit mapped representation. Number of digitals varies across BrainStem modules. Refer to the datasheet for the capabilities of your module. The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

setConfiguration(*configuration*)

Set the digital configuration to one of the available 5 states. Note: Some configurations are only supported on specific pins. aErrConfiguration Entity does not support this configuration.

Parameters

configuration (*const unsigned char*) – The configuration to be applied - Digital Input: digitalConfigurationInput = 0 - Digital Output: digitalConfigurationOutput = 1 - RCServo Input: digitalConfigurationRCServoInput = 2 - RCServo Output: digitalConfigurationRCServoOutput = 3 - High Z State: digitalConfigurationHiZ = 4 - Digital Input: digitalConfigurationInputPullUp = 0 - Digital Input: digitalConfigurationInputNoPull = 4 - Digital Input: digitalConfigurationInputPullDown = 5

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setState(*state*)

Set the logical state.

Parameters

state (*const unsigned char*) – The state to be set. 0 is logic low, 1 is logic high.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setStateAll(*state*)

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

Parameters

state (*const unsigned int*) – The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

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[86]

`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)
  - 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*, *aether_config=<brainstem.link.aEtherConfig object>*, *buffer_length=128*)

Return a list of Specs for all modules found on the transports given.

Transports can be presented as a list, and the results would be a list of all modules found for those transports. TCPIP modules take a little longer to find due to the Multicast and gather necessary for finding modules on the local network segment.

Parameters

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

[86] https://acroname.com/reference

Returns
A list of the Spec objects for all modules found.

Return type
list(*Spec*)

`brainstem.discover.`**`findFirstModule`**(*transports*, *aether_config=<brainstem.link.aEtherConfig object>*)

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

Parameters

- **`transports`** (*`int or list(int)`*) – A list of transports or a single transport.
- **`aether_config`** (`aEtherConfig`) – Allows configuration of aEther other than the default.

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

Return type
Spec

`brainstem.discover.`**`findModule`**(*transports*, *serial_number*, *aether_config=<brainstem.link.aEtherConfig object>*)

Return the Spec for the module with the given serial number.

Transports can be presented as a list. TCPIP modules take a little longer to find due to the Multicast and gather necessary for finding modules on the local network segment.

Parameters

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

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

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

Gets downstream device USB information for all Acroname hubs.

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

Returns
Result object containing NO_ERROR and a tuple of DeviceNode's containing the detected downstream devices:: - **aErrParam**: Passed in values are not valid (NULL, size, etc). - **aErrMemory**: No more room in the list. - **aErrNotFound**: No Acroname devices were found.

Return type
Result

`brainstem.discover.`**`getIPv4Interfaces`**(*list_length=30*)

Populates a list with all of the available IPv4 Interfaces.

Parameters
`list_length` (*`unsigned int`*) – Size of list to allocate for.

Returns
A tuple of IPv4 interfaces.

Return type
tuple(unsigned int)

3.2.10 Entity

class brainstem.Entity_Entity.**Entity**(*module*, *cmd*, *index*)

Base class for BrainStem Entity.

Provides the default implementation for a functional entity within the BrainStem. This can include IO like GPIOs, Analogs etc. For a more detailed description of Entities see the Terminology[87] section of the brainstem reference for more information.

call_UEI(*option*)

Call a set UEI on this entity.

Parameters
option (*byte*) – The command option.

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

property command

Return the entity command.

Type
int

drain_UEI(*option*)

Drain UEI packets matching option.

Parameters
option (*byte*) – The command option.

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

getStreamStatus(*buffer_length=1024*)

Gets all available stream values associated with the cmd and index of the called API.

Parameters
buffer_length (*unsigned int*) – Size of the buffer to allocate

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

get_UEI16(*option*)

Get a UEI short value.

Parameters
option (*byte*) – The command option.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

get_UEI16_with_subindex(*option*, *subIndex*)

Call a get UEI short value with a subIndex.

Parameters
- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

get_UEI32(*option*)
Get a UEI int value.

Parameters
option (*byte*) – The command option.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

get_UEI32_with_subindex(*option*, *subIndex*)
Call a get UEI int value with a subIndex.

Parameters
- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

get_UEI8(*option*)
Get a UEI byte value.

Parameters
option (*byte*) – The command option.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

get_UEI8_with_subindex(*option*, *subIndex*)
Call a get UEI byte value with a subIndex.

Parameters
- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

get_UEIBytes(*option*, *buffer_length=65536*)
Get a UEI Bytes buffer on this entity.

Parameters
- **option** (*byte*) – The command option.
- **buffer_length** (*unsigned int*) – The subIndex of the entity.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

property index
Return the entity index

Type
int

property module
returns the associated module object.

Type
Module

registerOptionCallback(*option*, *enable*, *cb*, *pRef*)
Registers a callback function based on a specific option code. Option code applies to the cmd and index of the called API.

:param option The option code for the entities command and index. :type option: byte

:param enable Enable (True) or disable (False) streaming. :type enable: bool

:param cb Callback to be executed on the provided criteria. :type cb: @ffi.callback("unsigned char(aPacket*, void*)")

:param pRef Handle to be passed to the provided callback. This handle must be kept alive by the caller. :type pRef: ffi handle

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

setStreamEnabled(*enable*)
Enables streaming for all possible option codes within the cmd and index the entity was created for.

Parameters
enable (*bool*) – Enable (True) or disable (False) streaming.

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

set_UEI16(*option*, *value*)
Call a set UEI with short value on this entity.

Parameters
- **option** (*byte*) – The command option.
- **value** (*short*) – The short parameter to send.

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

set_UEI16_with_subindex(*option*, *subIndex*, *value*)
Call a set UEI short value with a subIndex.

Parameters
- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.
- **value** (*short*) – The short parameter to send.

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

set_UEI32(*option*, *value*)
Call a set UEI with int value on this entity.

Parameters
- **option** (*byte*) – The command option.

- **value** (*int*) – The int parameter to send.

set_UEI32_with_subindex(*option*, *subIndex*, *value*)

Call a set UEI int value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.
- **value** (*int*) – The int parameter to send.

Returns

An error result from the list of defined error codes in brainstem.result

set_UEI8(*option*, *value*)

Call a set UEI with byte value on this entity.

Parameters

- **option** (*byte*) – The command option.
- **value** (*byte*) – The byte parameter to send.

Returns

An error result from the list of defined error codes in brainstem.result

set_UEI8_with_subindex(*option*, *subIndex*, *value*)

Call a set UEI byte value with a subIndex.

Parameters

- **option** (*byte*) – The command option.
- **subIndex** (*byte*) – The subIndex of the entity.
- **value** (*byte*) – The byte parameter to send.

Returns

An error result from the list of defined error codes in brainstem.result

set_UEIBytes(*option*, *buffer*)

Call a set UEI with buffer and length of buffer on this entity.

Parameters

- **option** (*byte*) – The command option.
- **buffer** (*bytearray()*) – The buffer to be sent

Returns

An error result from the list of defined error codes in brainstem.result

3.2.11 Equalizer

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

EqualizerClass: 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. Configuration of the receiver.

Parameters

channel (*const unsigned char*) – The equalizer receiver channel.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

[87] https://acroname.com/reference/brainstem/terms.html

getTransmitterConfig()

Gets the transmitter configuration Configuration of the Transmitter.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

setReceiverConfig(*channel*, *config*)

Sets the receiver configuration for a given channel.

Parameters
- **channel** (*const unsigned char*) – The equalizer receiver channel.
- **config** (*const unsigned char*) – Configuration to be applied to the receiver.

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

Return type
unsigned byte

setTransmitterConfig(*config*)

Sets the transmitter configuration

Parameters
config (*const unsigned char*) – Configuration to be applied to the transmitter.

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

Return type
unsigned byte

3.2.12 I2C

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

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.

getSpeed()

Get I2C bus speed. This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz - The speed setting value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

read(*address*, *readLength*)

Read from a device on this I2C bus.

Parameters
- **address** (*const int*) –
  - The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **readLength** (*const int*) –
  - The length of the data to read in bytes.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

setPullup(*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 (*const bool*) –
- true enables pull-ups false disables them.

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

Return type
unsigned byte

setSpeed(*speed*)

Set I2C bus speed. This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz

Parameters
speed (*const unsigned char*) –
- The speed setting value.

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

Return type
unsigned byte

write(*address*, *buffer*)

Write to a device on this I2C bus.

Parameters
address (*const int*) –
- The I2C address (7bit <XXXX-XXX0>) of the device to write.

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

Return type
unsigned byte

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[88]

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.

[88] https://acroname.com/reference

Parameters

- **transport** (*int*) – One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*int*) – The module serial number.
- **module** – The module address on the Brainstem network.
- **model** – The device model number of the Brainstem module.
- ****keywords** – For TCPIP, SERIAL and AETHER connections. The possibilities are,
  - ip_address: (int/str) The IPV4 address for a TCPIP/AETHER connection type.
  - ip_port: (int/str) The port for a TCPIP/AETHER connection type.
  - port: (str) The serial port for a SERIAL connection type.
  - baudrate: (int/str) The baudrate for a SERIAL connection type.

AETHER = 4

AETHER transport type.

INVALID = 0

INVALID Undefined transport type.

SERIAL = 3

SERIAL transport type.

TCPIP = 2

TCPIP transport type.

USB = 1

USB transport type.

static cca_spec_to_python_spec(*cca_spec*)

Internal: Translate cffi spec into python Spec

class brainstem.link.**Status**

Status variables represent the link status possibilities for Brainstem Links.

Status States:

- STOPPED (0)
- INITIALIZING (1)
- RUNNING (2)
- STOPPING (3)
- SYNCING (4)
- INVALID_LINK_STREAM (5)
- IO_ERROR (6)
- UNKNOWN_ERROR (7)

class brainstem.link.**StreamStatusEntry**(*key*, *value*)

property key

A unique key made up of module, cmd, option, index, subindex

Type

unsigned long long (64bit)

property value

The Value associated with the key

Type
unsigned int (32bit)

class brainstem.link.**aEtherConfig**

aEther configuration class for configuring AETHER connection types.

Note: If localOnly == false AND networkInterface is default (0 or LOCALHOST_IP_ADDRESS) it will be populated with the auto-selected interface upon successful connection.

enabled

True: Client-Server model is used; False: Direct module control is used.

fallback

True: If connections fails it will automatically search for network connections.

localOnly

True: Restricts access to localhost; False: Expose device to external network.

assignedPort

Server assigned port after successful connection.

networkInterface

Network interface to use for connections.

3.2.14 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[89] section of the Acroname BrainStem Reference[90]

class brainstem.module.**Module**(*address*, *enable_auto_networking=True*, *model=0*)

The Module Entity provides a generic interface to a BrainStem hardware module. The Module Class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

property address

Module address of the device

Type
unsigned byte

property bAutoNetworking

Return the current networking mode.

Type
bool

classQuantity(*command*)

Queries the module to determine how many entities of the specified class are implemented by the module. Zero is a valid return value. For example, calling classQuantity with the command parameter of cmdANALOG would return the number of analog entities implemented by the module.

[89] https://acroname.com/reference/brainstem/terms.html
[90] https://acroname.com/reference

Parameters
: **command** (*unsigned byte*) – One of the UEI commands (cmdXXX).

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

connect(*transport*, *serial_number*)

Connect to a Module with a transport type and serial number.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

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

connectFromSpec(*spec*)

Connect to a BrainStem module with a Spec.

Parameters
: **spec** (*Spec*) – The specifier for the connection.

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

connectThroughLinkModule(*module*)

Connects to a Brainstem module on a BrainStem network, through the module given as an argument. The module passed in must have an active valid connection.

Parameters
: **module** (*Module*) – The brainstem module to connect through.

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

disconnect()

Disconnect from the Brainstem module.

discoverAndConnect(*transport*, *serial_number=0*)

Discover and connect from the Module level.

A discover-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 or None as the serial number will create a link to the first link module found on the specified transport.

Parameters

- **transport** (*Spec.transport*) – (Spec.transport): One of USB, TCPIP, SERIAL or AETHER.
- **serial_number** (*unsigned int*) – The module serial_number to look for.

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

entityGroup(*command*, *index*)

Queries the module the group assigned to an entity and index. Entities groups are used to specify when certain hardware features are fundamentally related. E.g. certain hardware modules may have some digital pins associated with an adjustable voltage rail; these digitals would be in the same group as the rail. Zero is the default group.

Parameters
command (*unsigned byte*) – One of the UEI commands (cmdXXX).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

getBuild()

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

getConfig()

Gets the links current aEther configuration

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result containing a aEtherConfig

getModuleAddress()

Get the address of the module object.

This method changes the local address of the module, not of the device. It is possible to get the module address of the device via system.getModuleSoftwareOffset().

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

getStatus()

Returns the status of the BrainStem connection See brainstem.link.Status for the possible states.

hasUEI(*command*, *option*, *index*, *flags*)

Queries the module to determine if it implements a UEI. Each UEI has a command, option or variant, index and flag. The hasUEI method queries for a fully specified UEI. Returns aErrNone if the variation is supported and an appropriate error if not. This call is blocking for up to the nMSTimeout period.

Parameters

- **command** (*unsigned byte*) – One of the UEI commands (cmdXXX).
- **option** (*unsigned byte*) – The option or variant of the command.
- **index** (*unsigned byte*) – The entity index.
- **flags** (*unsigned byte*) – The flags (ueiOPTION_SET or ueiOPTION_GET).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

property id

A unique identifier of the associated module

Type
unsigned int

isConnected()

Returns true if the Module has an active connection or false otherwise

property link

return the current link or None.

Type
Link

property model

Model number of the device

Type
unsigned byte

reconnect()

Reconnect a lost connection to a Brainstem module.

setConfig(*config*)

Sets the links aEther configuration. Note: Configuration must be set BEFORE connection.

Parameters
config (`aEtherConfig`) – (aEtherConfig object): aEther configuration to be set.

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

setModuleAddress(*address*)

Set the address of the module object.

This method changes the local address of the module, not of the device. It is possible to set the module address of the device via system.setModuleSoftwareOffset().

Parameters
address (*unsigned byte*) – The module address to switch to for this module instance.

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

setNetworkingMode(*mode*)

Changes the networking mode of the stem object. Auto mode is enabled by default which allows automatic adjustment of the module/stems networking configuration. Refer to BrainStem Networking at www.acroname.com/support

Parameters
mode (*bool*) – Mode to be set. True = Auto; False = Manual

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

subClassQuantity(*command*, *index*)

Queries the module to determine how many subclass entities of the specified class are implemented by the module for a given entity index. This is used for entities which may be 2-dimensional. E.g. cmdMUX subclasses are the number of channels supported by a particular mux type (index);

as a specific example, a module may support 4 UART channels, so subClassQuantity(cmdMUX, aMUX_UART...) could return 4. Zero is a valid return value.

Parameters
: **command** (*unsigned byte*) – One of the UEI commands (cmdXXX).

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

3.2.15 Mux

class brainstem.entity.**Mux**(*module*, *index*)

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).

getChannel()

Get the current selected mux channel. Indicates which chanel is selected.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getChannelVoltage(*channel*)

Get the voltage of the indicated mux channel. 32 bit signed integer (in microvolts) based on the board's ground and reference voltages. Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters
: **channel** (*const unsigned char*) – The channel in which voltage was requested.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getConfiguration()

Get the configuration of the mux. integer representing the mux configuration either default, or split-mode.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getEnable()

Get the mux enable/disable status true: mux is enabled, false: the mux is disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getSplitMode()

Get the current split mode mux configuration. integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

setChannel(*channel*)

Set the current mux channel.

Parameters
channel (*const unsigned char*) – mux channel to select.

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

Return type
unsigned byte

setConfiguration(*config*)

Set the configuration of the mux.

Parameters
config (*const int*) – integer representing the mux configuration either muxConfig_default, or muxConfig_splitMode.

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

Return type
unsigned byte

setEnable(*bEnable*)

Enable the mux.

Parameters
bEnable (*const unsigned char*) – true: enables the mux for the selected channel.

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

Return type
unsigned byte

setSplitMode(*splitMode*)

Sets the mux's split mode configuration.

Parameters
splitMode (*const int*) – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

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

Return type
unsigned byte

3.2.16 PDChannelLogger

class `brainstem.pd_channel_logger.`**`BS_PD_Packet`**(*channel=0*, *seconds=0*, *uSeconds=0*, *direction=0*, *sop=0*, *event=0*, *payload=[]*)

Python representation of BS_PD_Packet_t (C structure)

- channel (uint8_t): Channel/Index
- seconds (uint8_t): Seconds in device time since power on.
- uSeconds (uint32_t): Micro Seconds in device time since power on.
- **direction (enumeration): Direction of packet transmission relative to the device.**
  - Invalid = 0
  - Transmit = 1
  - Receive = 2
  - Unknown = 3
- **sop (enumeration): See bs_pd_packet.h for more details**
  - SOP = 0
  - SOP' = 1
  - SOP'' = 2
  - Unknown = 3
- **event (enumeration): See powerdeliveryLogEvent in aProtocolDefs.h**
  - pdEventNone = 0
  - pdEventPacket = 1
  - pdEventConnect = 2
  - pdEventDisconnect = 3
  - pdEventCableResetReceived = 4
  - pdEventCableResetSent = 5
  - pdEventHardResetReceived = 6
  - pdEventHardResetSent = 7
  - pdEventMessageTransmitFailed = 8 // No GoodCRC received
  - pdEventMessageTransmitDiscarded = 9 // Incoming message detected so tx discarded
  - pdEventPDFunctionDisabled = 10 // PD Stack is giving up on PD Comms
  - pdEventVBUSEnabled = 11
  - pdEventVBUSDisabled = 12
  - pdEventVCONNEnabled = 13
  - pdEventVCONNDisabled = 14

- pdEventRp1A5 = 15 // Used for Src Atomic Message Sequences
- pdEventRp3A0 = 16 // Used for Src Atomic Message Sequences
- pdEventBistEnter = 17
- pdEventBistExit = 18
- pdEventLast = 19 // Should always be last!!

- payload (list): Raw PD Packet data

class brainstem.pd_channel_logger.**PDChannelLogger**(*module*, *index*, *buffer_length=1024*)

Manages BrainStem Power Delivery logging packets.

Parameters

- **module** (Module) – : Reference to an existing BrainStem Module
- **index** (*unsigned byte*) – Index/channel logging should be enabled for.
- **buffer_length** (*unsigned short*) – Number of packets the class should queue before dropping.

property buffer_length

Gets the buffer length

Returns
Buffer length of the associated object.

Return type
unsigned int

getPacket()

Attempts to takes a packet from the internal buffer.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

getPackets(*buffer_length=100*)

Attempts to take a multiple packets (up to a maximum) from the internal buffer.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

property index

Gets the Index/Channel

Returns
Index/channel of the associated object.

Return type
unsigned byte

property module

Gets the Module object.

Returns
The associated module object.

Return type
Module

setEnabled(*enabled*)

Enables Power Delivery logging.

Parameters

enable (*bool*) – True enables logging; False disables logging

return: An error result from the list of defined error codes in brainstem.result

3.2.17 Pointer

class brainstem.entity.**Pointer**(*module*, *index*)

PointerClass: Allows access to the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the BrainStem scratchpad from the host, and provides a mechanism for allowing the host application and BrainStem relexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

getChar()

Get a char (1 byte) value from the pointer at this object's index, where elements are 1 byte long. The value of a single character (1 byte) stored in the pointer. All possible standard UEI return values.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getInt()

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long The value of a single int (4 byte) stored in the pointer. All possible standard UEI return values.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getMode()

Get the mode of the pointer The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT. All possible standard UEI return values.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getOffset()

Get the offset of the pointer The value of the offset. All possible standard UEI return values.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getShort()

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long The value of a single short (2 byte) stored in the pointer. All possible standard UEI return values.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getTransferStore()

Get the handle to the store. The handle of the store. All possible standard UEI return handles.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

initiateTransferFromStore(*transferLength*)

Transfer data from the store. All possible standard UEI return values.

Parameters
transferLength (*unsigned char*) – The length of the data transfer.

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

Return type
unsigned byte

initiateTransferToStore(*transferLength*)

Transfer data to the store. All possible standard UEI return values.

Parameters
transferLength (*unsigned char*) – The length of the data transfer.

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

Return type
unsigned byte

setChar(*value*)

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long. All possible standard UEI return values.

Parameters
value (*const unsigned char*) – The single char (1 byte) value to be stored in the pointer.

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

Return type
unsigned byte

setInt(*value*)

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long All possible standard UEI return values.

Parameters
: **value** (*const unsigned int*) – The single int (4 byte) value to be stored in the pointer.

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

Return type
: unsigned byte

setMode(*mode*)

Set the mode of the pointer All possible standard UEI return values.

Parameters
: **mode** (*unsigned char*) – The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.

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

Return type
: unsigned byte

setOffset(*offset*)

Set the offset of the pointer All possible standard UEI return values.

Parameters
: **offset** (*unsigned short*) – The value of the offset.

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

Return type
: unsigned byte

setShort(*value*)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long. All possible standard UEI return values.

Parameters
: **value** (*const unsigned short*) – The single short (2 byte) value to be set in the pointer.

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

Return type
: unsigned byte

setTransferStore(*handle*)

Set the handle to the store. All possible standard UEI return handles.

Parameters
: **handle** (*unsigned char*) – The handle of the store.

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

Return type
: unsigned byte

3.2.18 Port

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

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.

getAllocatedPower()

Gets the currently allocated power This value is determined by the power manager which is responsible for budgeting the systems available power envelope. Variable to be filled with the allocated power in milli-watts (mW).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getAvailablePower()

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope. Variable to be filled with the available power in milli-watts (mW).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCC1Enabled()

Gets the current enable value of the CC1 lines. Sub-component of getCCEnabled. 1 = CC1 enabled; 0 = CC1 disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCC1State()

Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance. Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows: - None = 0 = portCC1State_None - Invalid = 1 = portCC1State_Invalid - Rp (default) = 2 = portCC1State_RpDefault - Rp (1.5A) = 3 = portCC1State_Rp1p5 - Rp (3A) = 4 = portCC1State_Rp3p0 - Rd = 5 = portCC1State_Rd - Ra = 6 = portCC1State_Ra - Managed by controller = 7 = portCC1State_Managed - Unknown = 8 = portCC1State_Unknown

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCC2Enabled()

Gets the current enable value of the CC2 lines. Sub-component of getCCEnabled. 1 =

CC2 enabled; 0 = CC2 disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCC2State()

Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance. Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows: - None = 0 = portCC2State_None - Invalid = 1 = portCC2State_Invalid - Rp (default) = 2 = portCC2State_RpDefault - Rp (1.5A) = 3 = portCC2State_Rp1p5 - Rp (3A) = 4 = portCC2State_Rp3p0 - Rd = 5 = portCC2State_Rd - Ra = 6 = portCC2State_Ra - Managed by controller = 7 = portCC2State_Managed - Unknown = 8 = portCC2State_Unknown

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCCCurrentLimit()

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations. Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCCEnabled()

Gets the current enable value of the CC lines.: Sub-component (CC) of getEnabled. 1 = CC enabled; 0 = CC disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCurrentLimit()

Gets the current limit of the port. Variable to be filled with the limit in microAmps (uA).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCurrentLimitMode()

Gets the current limit mode. The mode determines how the port will react to an over current condition. Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataEnabled()

Gets the current enable value of the data lines.: Sub-component (Data) of getEnabled. 1 = Data enabled; 0 = Data disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataHS1Enabled()

Gets the current enable value of the High Speed A side (HSA) data lines.: Sub-component of getDataHSEnabled. 1 = Data enabled; 0 = Data disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataHS2Enabled()

Gets the current enable value of the High Speed B side (HSB) data lines.: Sub-component of getDataHSEnabled. 1 = Data enabled; 0 = Data disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataHSEnabled()

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled. 1 = Data enabled; 0 = Data disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataHSRoutingBehavior()

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines. Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataRole()

Gets the Port Data Role. The data role to be set. See datasheet for details.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataSS1Enabled()

Gets the current enable value of the Super Speed A side (SSA) data lines.: Sub-component of getDataSSEnabled. 1 = Data enabled; 0 = Data disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataSS2Enabled()

Gets the current enable value of the Super Speed B side (SSB) data lines.: Sub-component of getDataSSEnabled. 1 = Data enabled; 0 = Data disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataSSEnabled()

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled. 1 = Data enabled; 0 = Data disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataSSRoutingBehavior()

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines. Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataSpeed()

Gets the speed of the enumerated device. Bit mapped value representing the devices speed. See "Devices" reference for details.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getEnabled()

Gets the current enable value of the port. 1 = Fully enabled port; 0 = One or more disabled components.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getErrors()

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again. Bit mapped field representing the current errors of the ports

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getHSBoost()

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode. An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getMode()

Gets current mode of the port Bit mapped value representing the ports mode. See "Devices" reference for details.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getName(*buffer_length=65536*)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment. Length that was actually received and filled.

Parameters
buffer_length – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerEnabled()

Gets the current enable value of the power lines.: Sub-component (Power) of getEnabled. 1 = Power enabled; 0 = Power disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerLimit()

Gets the user defined power limit for the port. Variable to be filled with the power limit in milli-watts (mW).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerLimitMode()

Gets the power limit mode. The mode determines how the port will react to an over power condition. Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerMode()

Gets the Port Power Mode: Convenience Function of get/setPortMode The current power mode.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getState()

A bit mapped representation of the current state of the port. Reflects what he port IS which may differ from what was requested. Variable to be filled with the current state.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVbusAccumulatedPower()

Gets the Vbus Accumulated Power The accumuled power on Vbus in milliwatt-hours.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVbusCurrent()

Gets the Vbus Current The current in microamps (1 == 1e-6A) currently present on Vbus.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVbusVoltage()

Gets the Vbus Voltage The voltage in microvolts (1 == 1e-6V) currently present on Vbus.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVconn1Enabled()

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled. 1 = Vconn1 enabled; 0 = Vconn1 disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVconn2Enabled()

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled. 1 = Vconn2 enabled; 0 = Vconn2 disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVconnAccumulatedPower()

Gets the Vconn Accumulated Power The accumuled power on Vconn in milliwatt-hours.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVconnCurrent()

Gets the Vconn Current The current in microamps (1 == 1e-6A) currently present on Vconn.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVconnEnabled()

Gets the current enable value of the Vconn lines.: Sub-component (Vconn) of getEnabled. 1 = Vconn enabled; 0 = Vconn disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVconnVoltage()

Gets the Vconn Voltage The voltage in microvolts (1 == 1e-6V) currently present on Vconn.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVoltageSetpoint()

Gets the current voltage setpoint value for the port. the voltage setpoint of the port in uV.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

resetEntityToFactoryDefaults()

Resets the PortClass Entity to it factory default configuration.

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

Return type
unsigned byte

resetVbusAccumulatedPower()

Resets the Vbus Accumulated Power to zero.

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

Return type
unsigned byte

resetVconnAccumulatedPower()

Resets the Vconn Accumulated Power to zero.

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

Return type
unsigned byte

setCC1Enabled(*enable*)

Enables or disables the CC1 lines. Sub-component of setCCEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable CC1 lines; 0 = Disable CC1 lines.

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

Return type
unsigned byte

setCC2Enabled(*enable*)

Enables or disables the CC2 lines. Sub-component of setCCEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable CC2 lines; 0 = Disable CC2 lines.

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

Return type
unsigned byte

setCCCurrentLimit(*value*)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations.

Parameters
value (*const unsigned char*) – Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A

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

Return type
unsigned byte

setCCEnabled(*enable*)

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable CC lines; 0 = Disable CC lines.

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

Return type
unsigned byte

setCurrentLimit(*limit*)

Sets the current limit of the port.

Parameters
limit (*const unsigned int*) – Current limit to be applied in microAmps (uA).

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

Return type
unsigned byte

setCurrentLimitMode(*mode*)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters
mode (*const unsigned char*) – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

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

Return type
unsigned byte

setDataEnabled(*enable*)

Enables or disables the data lines. Sub-component (Data) of setEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable data; 0 = Disable data.

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

Return type
unsigned byte

setDataHS1Enabled(*enable*)

Enables or disables the Hight Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable data; 0 = Disable data.

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

Return type
unsigned byte

setDataHS2Enabled(*enable*)

Enables or disables the Hight Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable data; 0 = Disable data.

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

Return type
unsigned byte

setDataHSEnabled(*enable*)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable data; 0 = Disable data.

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

Return type
unsigned byte

setDataHSRoutingBehavior(*mode*)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters
mode (*const unsigned char*) – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

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

Return type
unsigned byte

setDataSS1Enabled(*enable*)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable data; 0 = Disable data.

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

Return type
unsigned byte

setDataSS2Enabled(*enable*)

Enables or disables the Super Speed B side (SSB) data lines. Sub-component of setDataSSEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable data; 0 = Disable data.

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

Return type
unsigned byte

setDataSSEnabled(*enable*)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Parameters

enable (*const unsigned char*) – 1 = Enable data; 0 = Disable data.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setDataSSRoutingBehavior(*mode*)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode (*const unsigned char*) – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setEnabled(*enable*)

Enables or disables the entire port.

Parameters

enable (*const unsigned char*) – 1 = Fully enable port; 0 = Fully disable port.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setHSBoost(*boost*)

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Parameters

boost (*const unsigned char*) – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setMode(*mode*)

Sets the mode of the port

Parameters

mode (*const unsigned int*) – Port mode to be set. See "Devices" documentation for details.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setName(*buffer*)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

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

Return type
unsigned byte

setPowerEnabled(*enable*)

Enables or Disables the power lines. Sub-component (Power) of setEnable.

Parameters
enable (*const unsigned char*) – 1 = Enable power; 0 = Disable disable.

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

Return type
unsigned byte

setPowerLimit(*limit*)

Sets a user defined power limit for the port.

Parameters
limit (*const unsigned int*) – Power limit to be applied in milli-watts (mW).

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

Return type
unsigned byte

setPowerLimitMode(*mode*)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters
mode (*const unsigned char*) – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

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

Return type
unsigned byte

setPowerMode(*powerMode*)

Sets the Port Power Mode: Convenience Function of get/setPortMode

Parameters
powerMode (*const unsigned char*) – The power mode to be set.

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

Return type
unsigned byte

setVconn1Enabled(*enable*)

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable Vconn1 lines; 0 = Disable Vconn1 lines.

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

Return type
unsigned byte

setVconn2Enabled(*enable*)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable Vconn2 lines; 0 = Disable Vconn2 lines.

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

Return type
unsigned byte

setVconnEnabled(*enable*)

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

Parameters
enable (*const unsigned char*) – 1 = Enable Vconn lines; 0 = Disable Vconn lines.

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

Return type
unsigned byte

setVoltageSetpoint(*value*)

Sets the current voltage setpoint value for the port.

Parameters
value (*const unsigned int*) – the voltage setpoint of the port in uV.

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

Return type
unsigned byte

3.2.19 Power Delivery

class brainstem.entity.**PowerDelivery**(*module*, *index*)

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.

getCableCurrentMax()

Gets the maximum current capability report by the e-mark of the attached cable. Variable to be filled with an enumerated representation of current. - Unknown/Unattached (0) - 3 Amps (1) - 5 Amps (2)

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCableOrientation()

Gets the current orientation being used for PD communication Variable filled with an enumeration of the orientation. - Unconnected (0) - CC1 (1) - CC2 (2)

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCableSpeedMax()

Gets the maximum data rate capability reported by the e-mark of the attached cable. 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCableType()

Gets the cable type reported by the e-mark of the attached cable. 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCableVoltageMax()

Gets the maximum voltage capability reported by the e-mark of the attached cable. 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getConnectionState()

Gets the current state of the connection in the form of an enumeration. Pointer to be filled with the current connection state.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getFastRoleSwapCurrent()

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap. An enumerated value referring to current swap value. - 0A (0) - 900mA (1) - 1.5A (2) - 3A (3)

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getFlagMode (*flag*)

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors. Variable to be filled with the current mode. - Disabled (0) - Enabled (1) - Auto (2) default

Parameters

flag (*const unsigned char*) – Flag/Advertisement to be modified

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getNumberOfPowerDataObjects (*partner*, *powerRole*)

Gets the number of Power Data Objects (PDOs) for a given partner and power role. Variable to be filled with the number of PDOs.

Parameters

- **partner** (*const unsigned char*) – Indicates which side of the PD connection is in question. - Local = 0 = powerdeliveryPartnerLocal - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** (*const unsigned char*) – Indicates which power role of PD connection is in question. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getOverride ()

Gets the current enabled overrides Bit mapped representation of the current override configuration.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getPeakCurrentConfiguration ()

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit. An enumerated value referring to the current configuration. - Allowable values are 0 - 4

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getPowerDataObject (*partner*, *powerRole*, *ruleIndex*)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index. Variable to be filled with the requested power rule.

Parameters

- **partner** (*const unsigned char*) – Indicates which side of the PD connection is in question. - Local = 0 = powerdeliveryPartnerLocal - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** (*const unsigned char*) – Indicates which power role of PD connection is in question. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*const unsigned char*) – The index of the PDO in question. Valid index are 1-7.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

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). Variable to be filled with enabled state.

Parameters

- **powerRole** (*const unsigned char*) – Indicates which power role of PD connection is in question. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*const unsigned char*) – The index of the PDO in question. Valid index are 1-7.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerDataObjectEnabledList(*powerRole*)

Gets all Power Data Object enables for a given power role. Equivalent of calling PowerDeliveryClass::getPowerDataObjectEnabled() for all indexes. Variable to be filled with a mapped representation of the enabled PDOs for a given power role. Values align with a given rule index (bits 1-7, bit 0 is invalid)

Parameters
powerRole (*const unsigned char*) – Indicates which power role of PD connection is in question. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerDataObjectList(*buffer_length=65536*)

Gets all Power Data Objects (PDOs). Equivalent to calling PowerDeliveryClass::getPowerDataObject() on all partners, power roles, and index's. Length that was actually received and filled. On success this value should be 28 (7 rules * 2 partners * 2 power roles)

Parameters
buffer_length – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to

NO_ERROR(0)

Return type
Result

getPowerRole()

Gets the power role that is currently being advertised by the local partner. (CC Strapping). Variable to be filed with the power role - Disabled = 0 = powerdeliveryPowerRoleDisabled - Source = 1= powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink - Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerRolePreferred()

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping). Value to be applied. - Disabled = 0 = powerdeliveryPowerRoleDisabled - Source = 1= powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

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 Variable to be filled with the current RDO. Zero indicates the RDO is not active.

Parameters
partner (*const unsigned char*) – Indicates which side of the PD connection is in question. - Local = 0 = powerdeliveryPartnerLocal - Remote = 1 = powerdeliveryPartnerRemote

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

packDataObjectAttributes(*attributes*, *partner*, *powerRole*, *ruleIndex*)

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol. aErrNone on success; aErrParam with bad input.

Parameters

- **attributes** (*unsigned char **) – variable to be filled with packed values.
- **partner** (*const unsigned char*) – Indicates which side of the PD connection. - Local = 0 = powerdeliveryPartnerLocal - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** (*const unsigned char*) – Indicates which power role of PD connection. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*const unsigned char*) – Data object index.

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

Return type
unsigned byte

`request(request)`

Requests an action of the Remote partner. Actions are not guaranteed to occur. 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

ef EntityReturnValues "common entity" return values

param request
Request to be issued to the remote partner - pdRequestHardReset (0) - pdRequestSoftReset (1) - pdRequestDataReset (2) - pdRequest-PowerRoleSwap (3) - pdRequestPowerFastRoleSwap (4) - pdRequest-DataRoleSwap (5) - pdRequestVconnSwap (6) - pdRequestSinkGo-ToMinimum (7) - pdRequestRemoteSourcePowerDataObjects (8) - pdRequestRemoteSinkPowerDataObjects (9)

type request
const unsigned char

return
An error result from the list of defined error codes in brainstem.result

rtype
unsigned byte

`requestStatus()`

Gets the status of the last request command sent. Variable to be filled with the status

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`resetEntityToFactoryDefaults()`

Resets the PowerDeliveryClass Entity to it factory default configuration.

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

Return type
unsigned byte

`resetPowerDataObjectToDefault(powerRole, ruleIndex)`

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

Parameters

- **powerRole** (*const unsigned char*) – Indicates which power role of PD connection is in question. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*const unsigned char*) – The index of the PDO in question. Valid index are 1-7.

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

Return type
unsigned byte

`setFastRoleSwapCurrent(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 (*const unsigned char*) – An enumerated value referring to value to be set. - 0A (0) - 900mA (1) - 1.5A (2) - 3A (3)

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

Return type
unsigned byte

setFlagMode(*flag*, *mode*)

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Parameters

- **flag** (*const unsigned char*) – Flag/Advertisement to be modified
- **mode** (*const unsigned char*) – Value to be applied. - Disabled (0) - Enabled (1) - Auto (2) default

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

Return type
unsigned byte

setOverride(*overrides*)

Sets the current enabled overrides

Parameters
overrides (*const unsigned int*) – Overrides to be set in a bit mapped representation.

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

Return type
unsigned byte

setPeakCurrentConfiguration(*configuration*)

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Parameters
configuration (*const unsigned char*) – An enumerated value referring to the configuration to be set - Allowable values are 0 - 4

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

Return type
unsigned byte

setPowerDataObject(*powerRole*, *ruleIndex*, *pdo*)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

Parameters

- **powerRole** (*const unsigned char*) – Indicates which power role of PD connection is in question. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*const unsigned char*) – The index of the PDO in question. Valid index are 1-7.
- **pdo** (*const unsigned int*) – Power Data Object to be set.

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

Return type
unsigned byte

setPowerDataObjectEnabled(*powerRole*, *ruleIndex*, *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** (*const unsigned char*) – Indicates which power role of PD connection is in question. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*const unsigned char*) – The index of the PDO in question. Valid index are 1-7.
- **enabled** (*const unsigned char*) – The state to be set.

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

Return type
unsigned byte

setPowerRole(*powerRole*)

Set the current power role to be advertised by the Local partner. (CC Strapping).

Parameters
powerRole (*const unsigned char*) – Value to be applied. - Disabled = 0 = powerdeliveryPowerRoleDisabled - Source = 1= powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink - Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

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

Return type
unsigned byte

setPowerRolePreferred(*powerRole*)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

Parameters
powerRole (*const unsigned char*) – Value to be applied. - Disabled = 0 = powerdeliveryPowerRoleDisabled - Source = 1= powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink

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

Return type
unsigned byte

setRequestDataObject(*rdo*)

Sets the current Request Data Object (RDO) for a given partner. (Only the local partner can be changed.) RDOs: Are provided by the sinking device. Exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

Parameters
rdo (*const unsigned int*) – Request Data Object to be set.

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

Return type
unsigned byte

unpackDataObjectAttributes(*attributes*, *partner*, *powerRole*)

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol. Data object index. aErrNone on success; aErrParam with bad input.

Parameters
- **attributes** (*const unsigned char*) – variable to be filled with packed values.
- **partner** (*unsigned char **) – Indicates which side of the PD connection. - Local = 0 = powerdeliveryPartnerLocal - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** (*unsigned char **) – Indicates which power role of PD connection. - Source = 1 = powerdeliveryPowerRoleSource - Sink = 2 = powerdeliveryPowerRoleSink

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

3.2.20 Rail

class brainstem.entity.**Rail**(*module*, *index*)

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.

clearFaults()

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

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

Return type
unsigned byte

getCurrent()

Get the rail current. The current in micro-amps (1 == 1e-6A).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCurrentLimit()

Get the rail current limit setting. (Check product datasheet to see if this feature is available) The current in micro-amps (1 == 1e-6A).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getEnable()

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails. true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. Kelvin sensing is enabled or disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. Kelvin sensing is enabled or disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. The current operational mode setting.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getOperationalState()

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states. The current operational state, hardware configuration, faults, and operating mode.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPower()

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities. 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 to determine if this is a measured or stored value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerLimit()

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available) The power in milli-watts (mW).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getResistance()

Get the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities. 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. 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 to determine if this is a measured or stored value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getTemperature()

Get the rail temperature. 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
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. 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 to determine if this is a measured or stored value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVoltageMaxLimit()

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available) The voltage in micro-volts (1 == 1e-6V).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVoltageMinLimit()

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available) The voltage in micro-volts (1 == 1e-6V).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. 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 to determine if this is a measured or stored value.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

setCurrentLimit(*microamps*)

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

Parameters
microamps (*const int*) – The current in micro-amps (1 == 1e-6A).

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

Return type
unsigned byte

setCurrentSetpoint(*microamps*)

Set the rail supply current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Parameters
microamps (*const int*) – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

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

Return type
unsigned byte

setEnable(*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 (*const unsigned char*) – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

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

Return type
unsigned byte

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 (*const unsigned char*) – enable or disable kelvin sensing.

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

Return type
unsigned byte

setOperationalMode(*mode*)

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

Parameters
mode (*const unsigned char*) – The operational mode to employ.

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

Return type
unsigned byte

setPowerLimit(*milliwatts*)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters
milliwatts (*const int*) – The power in milli-watts (mW).

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

Return type
unsigned byte

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 (*const int*) – The power in milli-watts (1 == 1e-3W) to be supplied by the rail.

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

Return type
unsigned byte

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 (*const int*) – The resistance in milli-ohms (1 == 1e-3Ohms) to be drawn by the rail.

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

Return type
unsigned byte

setVoltageMaxLimit(*microvolts*)

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Parameters
microvolts (*const int*) – The voltage in micro-volts (1 == 1e-6V).

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

Return type
unsigned byte

setVoltageMinLimit(*microvolts*)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Parameters
microvolts (*const int*) – The voltage in micro-volts (1 == 1e-6V).

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

Return type
unsigned byte

setVoltageSetpoint(*microvolts*)

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Parameters
microvolts (*const int*) – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

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

Return type
unsigned byte

3.2.21 RCServo

class brainstem.entity.**RCServo**(*module*, *index*)

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.

getEnable()

Get the enable status of the servo channel. The current enable status of the servo entity. 0 is disabled, 1 is enabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPosition()

Get the position of the servo channel The current position of the servo channel. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getReverse()

Get the reverse status of the servo channel The current reverse status of the servo entity. 0 = not reversed, 1 = reversed.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

setEnable(*enable*)

Enable the servo channel

Parameters
enable (*const unsigned char*) – The state to be set. 0 is disabled, 1 is enabled.

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

Return type
unsigned byte

setPosition(*position*)

Set the position of the servo channel

Parameters

position (*const unsigned char*) – The position to be set. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setReverse(*reverse*)

Set the output to be reversed on the servo channel

Parameters

reverse (*const unsigned char*) – 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

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

3.2.22 Relay

class brainstem.entity.**Relay**(*module*, *index*)

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.

getEnable()

Get the state. False or 0 = Disabled, True or 1 = Enabled

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getVoltage()

Get the scaled micro volt value with reference to ground. 32 bit signed integer (in micro Volts) based on the boards ground and reference voltages. Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

setEnable(*bEnable*)

Set the enable/disable state.

Parameters

bEnable (*const unsigned char*) – False or 0 = Disabled, True or 1 = Enabled

Returns

An error result from the list of defined error codes in brainstem.result

Return type
unsigned byte

3.2.23 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[91]

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

static getErrorDescription(*error*, *buffer_length=256*)

Get the description of an error code.

Parameters
error (*int or Result object*) – The error to decode.

Returns
The error code in human readable form.

Return type
string

static getErrorText(*error*)

Get the string representation of an error code.

Parameters
error (*int or Result object*) – The error to decode.

Returns
The error code in human readable form.

Return type
string

property value

Return the value attribute

[91] https://acroname.com/reference

3.2.24 Signal

See the *Signal Entity* for generic information.

class brainstem.entity.**Signal**(*module*, *index*)

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.

getEnable()

Get the Enable/Disable of the signal. True to enable, false to disable

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getInvert()

Get the invert status the signal output. Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2. to invert, false for normal mode.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getT2Time()

Get the signal active period or T2 in nanoseconds. Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getT3Time()

Get the signal period or T3 in nanoseconds. Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

setEnable(*enable*)

Enable/Disable the signal output.

Parameters
enable (*const unsigned char*) – True to enable, false to disable

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

Return type
unsigned byte

setInvert(*invert*)

Invert the signal output. Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert (*const unsigned char*) – to invert, false for normal mode.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setT2Time(*t2_nsec*)

Set the signal active period or T2 in nanoseconds.

Parameters

t2_nsec (*const unsigned int*) – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setT3Time(*t3_nsec*)

Set the signal period or T3 in nanoseconds.

Parameters

t3_nsec (*const unsigned int*) – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

3.2.25 System

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

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.

getBootSlot()

Get the store slot which is mapped when the module boots. The slot number in aSTORE_INTERNAL that is mapped after the module boots.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getBuild()

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware. Variable to be filled with build.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getErrors()

Gets any system level errors. Calling this function will clear the current errors. If the error persists it will be set again. Bit mapped field representing the devices errors

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getHBInterval()

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. The current heartbeat delay.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. The module's hardware version information.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getInputCurrent()

Get the module's input current. The module's input current reported in microamps.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. Variable to be filled with an enumerated value representing behavior.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getInputPowerBehaviorConfig(*buffer_length=65536*)

Gets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power. Length that was actually received and filled.

Parameters
`buffer_length` – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getInputPowerSource()

Provides the source of the current power source in use. Variable to be filled with enumerated representation of the source.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getInputVoltage()

Get the module's input voltage. The module's input voltage reported in microvolts.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. true: LED on, false: LED off.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getLEDMaxBrightness()

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). Brightness value relative to 255

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getLinkInterface()

Gets the link interface configuration. This refers to which interface is being used for control by the device. Variable to be filled with an enumerated value representing interface. - 0 = Auto= systemLinkAuto - 1 = Control Port = systemLinkUSBControl - 2 = Hub Upstream Port = systemLinkUSBHub

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getMaximumTemperature()

Get the module's maximum temperature ever recorded in micro-C (uC) This value will persists through a power cycle. The module's maximum system temperature in micro-C

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getMinimumTemperature()

Get the module's minimum temperature ever recorded in micro-C (uC) This value will persists through a power cycle. The module's minimum system temperature in micro-C

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getModel()

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types. The module's model enumeration.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getModule()

Get the current address the module uses on the BrainStem network. The address the module is using on the BrainStem network.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getModuleBaseAddress()

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address. The address the module is using on the BrainStem network.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getModuleHardwareOffset()

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet. The module address offset.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getModuleSoftwareOffset()

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address. The address for the module. Value must be even from 0-254.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getName(*buffer_length=65536*)

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment. Length that was actually received and filled.

Parameters
buffer_length – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerLimit()

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices. The available power in milli-Watts (mW, 1 t)

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerLimitMax()

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability. Variable to be filled with the power limit in milli-Watts (mW)

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerLimitState()

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit(). Variable to be filled with the state.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getRouter()

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. The address.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getRouterAddressSetting()

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address. The address for the module. Value must be even from 0-254.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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. The module's serial number.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getTemperature()

Get the module's current temperature in micro-C The module's system temperature in micro-C

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getUnregulatedCurrent()

Gets the current passing through the unregulated port. Variable to be filled with the current in micro-Amps (uA).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getUnregulatedVoltage()

Gets the voltage present at the unregulated port. Variable to be filled with the voltage in micro-Volts (uV).

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getUptime()

Get the module's accumulated uptime in minutes The module's accumulated uptime in minutes.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getVersion()

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p. The build version date code.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

logEvents()

Saves system log events to a slot defined by the module (usually ram slot 0).

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

Return type
unsigned byte

reset()

Reset the system. aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

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

Return type
unsigned byte

resetDeviceToFactoryDefaults()

Resets the device to it factory default configuration.

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

Return type
unsigned byte

resetEntityToFactoryDefaults()

Resets the SystemClass Entity to it factory default configuration.

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

Return type
unsigned byte

routeToMe(*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 (*const unsigned char*) – Enable or disable of the route to me function 1 = enable.

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

Return type
unsigned byte

`save()`

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address

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

Return type
unsigned byte

`setBootSlot(`*slot*`)`

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

Parameters
slot (*const unsigned char*) – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

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

Return type
unsigned byte

`setHBInterval(`*interval*`)`

Set the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

Parameters
interval (*const unsigned char*) – The desired heartbeat delay.

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

Return type
unsigned byte

`setInputPowerBehavior(`*behavior*`)`

Sets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Parameters
behavior (*const unsigned char*) – An enumerated representation of behavior to be set.

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

Return type
unsigned byte

`setInputPowerBehaviorConfig(`*buffer*`)`

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

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

Return type
unsigned byte

setLED(*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 (*const unsigned char*) – true: turn the LED on, false: turn LED off.

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

Return type
unsigned byte

setLEDMaxBrightness(*brightness*)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Parameters
brightness (*const unsigned char*) – Brightness value relative to 255

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

Return type
unsigned byte

setLinkInterface(*linkInterface*)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

Parameters
linkInterface (*const unsigned char*) – An enumerated representation of interface to be set. - 0 = Auto= systemLinkAuto - 1 = Control Port = systemLinkUSBControl - 2 = Hub Upstream Port = systemLinkUSBHub

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

Return type
unsigned byte

setModuleSoftwareOffset(*address*)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters
address (*const unsigned char*) – The address for the module. Value must be even from 0-254.

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

Return type
unsigned byte

setName(*buffer*)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

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

Return type
unsigned byte

setPowerLimitMax(*power*)

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Parameters
power (*const unsigned int*) – Limit in milli-Watts (mW) to be set.

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

Return type
unsigned byte

setRouter(*address*)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to "drop off" the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

Parameters
address (*const unsigned char*) – The router address to be used.

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

Return type
unsigned byte

3.2.26 Store

class brainstem.entity.**Store**(*module*, *index*)

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.

getSlotCapacity(*slot*)

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold. The slot capacity.

Parameters
slot (*const unsigned char*) – The slot number.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getSlotLocked(*slot*)

Gets the current lock state of the slot Allows for write protection on a slot. Variable to be filed with the locked state.

Parameters
slot (*const unsigned char*) – The slot number

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getSlotSize(*slot*)

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes. The slot size.

Parameters
slot (*const unsigned char*) – The slot number.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getSlotState(*slot*)

Get slot state. true: enabled, false: disabled.

Parameters
slot (*const unsigned char*) – The slot number.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

loadSlot(*slot*, *buffer*)

Load the slot.

Parameters
slot (*const unsigned char*) – The slot number.

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

Return type
unsigned byte

setSlotLocked(*slot*, *lock*)

Sets the locked state of the slot Allows for write protection on a slot.

Parameters
- **slot** (*const unsigned char*) – The slot number
- **lock** (*const unsigned char*) – state to be set.

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

Return type
unsigned byte

slotDisable(*slot*)

Disable slot.

Parameters
slot (*const unsigned char*) – The slot number.

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

Return type
unsigned byte

slotEnable(*slot*)

Enable slot.

Parameters

slot (*const unsigned char*) – The slot number.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

unloadSlot(*slot*, *buffer_length=65536*)

Unload the slot data. Length of data that was unloaded. Unloaded length will never be larger than dataLength.

Parameters

- **slot** (*const unsigned char*) – The slot number.
- **buffer_length** –
  - The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

3.2.27 Temperature

class brainstem.entity.**Temperature**(*module*, *index*)

TemperatureClass: This entity is only available on certain modules, and provides a temperature reading in microcelsius.

getValue()

Get the modules temperature in micro-C The temperature in micro-Celsius (1 == 1e-6C).

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getValueMax()

Get the module's maximum temperature in micro-C since the last power cycle. The module's maximum temperature in micro-C

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

getValueMin()

Get the module's minimum temperature in micro-C since the last power cycle. The module's minimum temperature in micro-C

Returns

Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type

Result

resetEntityToFactoryDefaults()

Resets the TemperatureClass Entity to it factory default configuration.

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

Return type
unsigned byte

3.2.28 Timer

class brainstem.entity.**Timer**(*module*, *index*)

TimerClass: The Timer Class provides access to a simple scheduler. The timer can set to fire only once, or to repeat at a certain interval. Additionally, a timer entity can execute custom Reflex routines upon firing.

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. The timer expiration duration in microseconds.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getMode()

Get the mode of the timer which is either single or repeat mode. The mode of the time. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

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 (*const unsigned int*) – The duration before timer expiration in microseconds.

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

Return type
unsigned byte

setMode(*mode*)

Set the mode of the timer which is either single or repeat mode. aErrNone Action completed successfully.

Parameters
mode (*const unsigned char*) – The mode of the timer. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.

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

Return type
unsigned byte

3.2.29 UART

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

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.

`getBaudRate`()

Get the UART baud rate. Pointer variable to be filled with baud rate.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getEnable`()

Get the enabled state of the uart. true: enabled, false: disabled.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getProtocol`()

Get the UART protocol. Pointer to where result is placed.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`setBaudRate`(*rate*)

Set the UART baud rate.

Parameters
`rate` (*`const unsigned int`*) – baud rate.

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

Return type
unsigned byte

`setEnable`(*bEnabled*)

Enable the UART channel.

Parameters
`bEnabled` (*`const unsigned char`*) – true: enabled, false: disabled.

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

Return type
unsigned byte

`setProtocol`(*protocol*)

Set the UART protocol.

Parameters
`protocol` (*`const unsigned char`*) – An enumeration of serial protocols.

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

Return type
unsigned byte

3.2.30 USB

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

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.

`clearPortErrorStatus(channel)`

Clear the error status for the given port.

Parameters
channel (*const unsigned char*) – The port to clear error status for.

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

Return type
unsigned byte

`getAltModeConfig(channel)`

Get USB Alt Mode Configuration. The USB configuration for the given channel.

Parameters
channel (*const unsigned char*) – The USB sub channel

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getCC1Current(channel)`

Get the current through the CC1 for a port. The USB channel current in micro-amps (1 == 1e-6A).

Parameters
channel (*const unsigned char*) – The USB sub channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getCC1Enable(channel)`

Get Enable/Disable on the CC1 line. State to be filled - Disabled: 0 - Enabled: 1

Parameters
channel (*const unsigned char*) – USB channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

`getCC1Voltage(channel)`

Get the voltage of CC1 for a port. The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters
channel (*const unsigned char*) – The USB sub channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCC2Current(*channel*)

Get the current through the CC2 for a port. The USB channel current in micro-amps (1 == 1e-6A).

Parameters
channel (*const unsigned char*) – The USB sub channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCC2Enable(*channel*)

Get Enable/Disable on the CC1 line. - State to be filled - Disabled: 0 - Enabled: 1

Parameters
channel (*const unsigned char*) –

- USB channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCC2Voltage(*channel*)

Get the voltage of CC2 for a port. The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters
channel (*const unsigned char*) – The USB sub channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getCableFlip(*channel*)

Get Cable flip setting. The enable/disable status of cable flip.

Parameters
channel (*const unsigned char*) – The USB sub channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getConnectMode(*channel*)

Gets the connect mode of the switch. The current connect mode

Parameters
channel (*const unsigned char*) – The USB sub channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDownstreamBoostMode()

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. The current Downstream boost setting 0, 1, 2, or 3.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDownstreamDataSpeed(*channel*)

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using Filled with the current port data speed - N/A: usbDownstreamDataSpeed_na = 0 - Hi Speed: usbDownstreamDataSpeed_hs = 1 - SuperSpeed: usbDownstream-DataSpeed_ss = 2

Parameters
channel (*const unsigned char*) – USB downstream channel to check.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getEnumerationDelay()

Get the inter-port enumeration delay in milliseconds. Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getHubMode()

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning. The USB hub mode.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPortCurrent(*channel*)

Get the current through the power line for a port. The USB channel current in micro-amps (1 == 1e-6A).

Parameters
channel (*const unsigned char*) – The USB sub channel.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPortCurrentLimit(*channel*)

Get the current limit for the port. The current limit setting.

Parameters
: **channel** (*const unsigned char*) – USB downstream channel to limit.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getPortError(*channel*)

Get the current error for the Port. The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care

Parameters
: **channel** (*const unsigned char*) – USB downstream channel.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getPortMode(*channel*)

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices implement a common bit mapping for port mode at

ef usbPortMode

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

param channel
: USB downstream channel.

type channel
: const unsigned char

return
: Result object containing the requested value when the results error is set to NO_ERROR(0)

rtype
: Result

getPortState(*channel*)

Get the current State for the Port. The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care

Parameters
: **channel** (*const unsigned char*) – USB downstream channel.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getPortVoltage(*channel*)

Get the voltage on the power line for a port. The USB channel voltage in microvolts (1 == 1e-6V).

Parameters
: **channel** (*const unsigned char*) – The USB sub channel.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getSBU1Voltage(*channel*)

Get the voltage of SBU1 for a port. The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters
: **channel** (*const unsigned char*) – The USB sub channel.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getSBU2Voltage(*channel*)

Get the voltage of SBU2 for a port. The USB channel voltage in micro-volts (1 == 1e-6V).

Parameters
: **channel** (*const unsigned char*) – The USB sub channel.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getSBUEnable(*channel*)

Get the Enable/Disable status of the SBU The enable/disable status of the SBU

Parameters
: **channel** (*const unsigned char*) – The USB sub channel.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getUpstreamBoostMode()

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. The current Upstream boost setting 0, 1, 2, or 3.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getUpstreamMode()

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1. The Upstream port mode.

Returns
: Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
: *Result*

getUpstreamState()

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1. The Upstream port state.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

setAltModeConfig(*channel*, *configuration*)

Set USB Alt Mode Configuration.

Parameters
- **channel** (*const unsigned char*) – The USB sub channel
- **configuration** (*const unsigned int*) – The USB configuration to be set for the given channel.

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

Return type
unsigned byte

setCC1Enable(*channel*, *bEnable*)

Set Enable/Disable on the CC1 line.

Parameters
- **channel** (*const unsigned char*) – USB channel.
- **bEnable** (*const unsigned char*) – State to be set - Disabled: 0 - Enabled: 1

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

Return type
unsigned byte

setCC2Enable(*channel*, *bEnable*)

Set Enable/Disable on the CC2 line.

Parameters
- **channel** (*const unsigned char*) – USB channel.
- **bEnable** (*const unsigned char*) – State to be filled - Disabled: 0 - Enabled: 1

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

Return type
unsigned byte

setCableFlip(*channel*, *bEnable*)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters
- **channel** (*const unsigned char*) – The USB sub channel.
- **bEnable** (*const unsigned char*) – The state to be set The state to be set - Disabled: 0 - Enabled: 1

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

Return type
unsigned byte

setConnectMode(*channel*, *mode*)

Sets the connect mode of the switch.

Parameters

- **channel** (*const unsigned char*) – The USB sub channel.
- **mode** (*const unsigned char*) – The connect mode - usbManualConnect = 0 - usbAutoConnect = 1

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setDataDisable(*channel*)

Disable only the data lines for a port without changing the state of the power line.

Parameters

channel (*const unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setDataEnable(*channel*)

Enable the only the data lines for a port without changing the state of the power line.

Parameters

channel (*const unsigned char*) – The USB sub channel.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setDownstreamBoostMode(*setting*)

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Parameters

setting (*const unsigned char*) – Downstream boost setting 0, 1, 2, or 3.

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setEnumerationDelay(*ms_delay*)

Set the inter-port enumeration delay in milliseconds.

Parameters

ms_delay (*const unsigned int*) – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

An error result from the list of defined error codes in brainstem.result

Return type

unsigned byte

setHiSpeedDataDisable(*channel*)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setHiSpeedDataEnable(*channel*)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setHubMode(*mode*)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

Parameters
mode (*const unsigned int*) – The USB hub mode.

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

Return type
unsigned byte

setPortCurrentLimit(*channel*, *microamps*)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a stem.system.save() call.

Parameters

- **channel** (*const unsigned char*) – USB downstream channel to limit.
- **microamps** (*const unsigned int*) – The current limit setting.

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

Return type
unsigned byte

setPortDisable(*channel*)

Disable both power and data lines for a port.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setPortEnable(*channel*)

Enable both power and data lines for a port.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setPortMode(*channel*, *mode*)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode at
ef usbPortMode

param channel
USB downstream channel to set the mode on.

type channel
const unsigned char

param mode
The port mode setting as packed bit field.

type mode
const unsigned int

return
An error result from the list of defined error codes in brainstem.result

rtype
unsigned byte

setPowerDisable(*channel*)

Disable only the power line for a port without changing the state of the data lines.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setPowerEnable(*channel*)

Enable only the power line for a port without changing the state of the data lines.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setSBUEnable(*channel*, *bEnable*)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

Parameters
- **channel** (*const unsigned char*) – The USB sub channel.
- **bEnable** (*const unsigned char*) – The state to be set - Disabled: 0 - Enabled: 1

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

Return type
unsigned byte

setSuperSpeedDataDisable(*channel*)

Disable only the data lines for a port without changing the state of the power line, Super-Speed (3.0) only.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setSuperSpeedDataEnable(*channel*)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters
channel (*const unsigned char*) – The USB sub channel.

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

Return type
unsigned byte

setUpstreamBoostMode(*setting*)

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Parameters
setting (*const unsigned char*) – Upstream boost setting 0, 1, 2, or 3.

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

Return type
unsigned byte

setUpstreamMode(*mode*)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0, usbUpstreamModePort1, and usbUpstreamModeNone.

Parameters
mode (*const unsigned char*) – The Upstream port mode.

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

Return type
unsigned byte

3.2.31 USB System

class brainstem.entity.**USBSystem**(*module*, *index*)

USBSystem Class: The USBSystem class provides high level control of the lower level Port Class.

getDataHSMaxDatarate()

Gets the USB HighSpeed Max datarate Current maximum datarate for the USB HighSpeed signals.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataRoleBehavior()

Gets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections. Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataRoleBehaviorConfig(*buffer_length=65536*)

Gets the current data role behavior configuration Certain data role behaviors use a list of ports to determine priority host priority. Length that was actually received and filled.

Parameters
buffer_length – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataRoleList()

Gets the data role of all ports with a single call Equivalent to calling PortClass::getDataRole() on each individual port. A bit packed representation of the data role for all ports.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getDataSSMaxDatarate()

Gets the USB SuperSpeed Max datarate Current maximum datarate for the USB SuperSpeed signals.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getEnabledList()

Gets the current enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port. Bit packed representation of the enabled status for all ports.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getEnumerationDelay()

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration. the current inter-port delay in milliseconds.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getModeList(*buffer_length=65536*)

Gets the current mode of all ports with a single call. Equivalent to calling PortClass:getMode() on each port. Length that was actually received and filled.

Parameters
buffer_length – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getOverride()

Gets the current enabled overrides Bit mapped representation of the current override configuration.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerBehavior()

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system. i.e. What happens when requested power greater than available power. Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getPowerBehaviorConfig(*buffer_length=65536*)

Gets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power. Length that was actually received and filled.

Parameters
buffer_length – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getSelectorMode()

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used. Variable to be filled with the selector mode

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getStateList(*buffer_length=65536*)

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port. Length that was actually received and filled.

Parameters
buffer_length – Length of the buffer to be filed

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getUpstream()

Gets the upstream port. The current upstream port.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getUpstreamHS()

Gets the USB HighSpeed upstream port. The current upstream port.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

getUpstreamSS()

Gets the USB SuperSpeed upstream port. The current upstream port.

Returns
Result object containing the requested value when the results error is set to NO_ERROR(0)

Return type
Result

resetEntityToFactoryDefaults()

Resets the USBSystemClass Entity to it factory default configuration.

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

Return type
unsigned byte

setDataHSMaxDatarate(*datarate*)

Sets the USB HighSpeed Max datarate

Parameters
datarate (*const unsigned int*) – Maximum datarate for the USB High-Speed signals.

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

Return type
unsigned byte

setDataRoleBehavior(*behavior*)

Sets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters
behavior (*const unsigned char*) – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

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

Return type
unsigned byte

setDataRoleBehaviorConfig(*buffer*)

Sets the current data role behavior configuration Certain data role behaviors use a list of ports to determine host priority.

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

Return type
unsigned byte

setDataSSMaxDatarate(*datarate*)

Sets the USB SuperSpeed Max datarate

Parameters
datarate (*const unsigned int*) – Maximum datarate for the USB SuperSpeed signals.

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

Return type
unsigned byte

setEnabledList(*enabledList*)

Sets the enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

Parameters
enabledList (*const unsigned int*) – Bit packed representation of the enabled status for all ports to be applied.

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

Return type
unsigned byte

setEnumerationDelay(*msDelay*)

Sets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Parameters
msDelay (*const unsigned int*) – The delay in milliseconds to be applied between port enables

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

Return type
unsigned byte

setModeList(*buffer*)

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port

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

Return type
unsigned byte

setOverride(*overrides*)

Sets the current enabled overrides

Parameters
overrides (*const unsigned int*) – Overrides to be set in a bit mapped representation.

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

Return type
unsigned byte

setPowerBehavior(*behavior*)

Sets the behavior of how available power is managed. i.e. What happens when requested power is greater than available power.

Parameters
behavior (*const unsigned char*) – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

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

Return type
unsigned byte

setPowerBehaviorConfig(*buffer*)

Sets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

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

Return type
unsigned byte

setSelectorMode(*mode*)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters
mode (*const unsigned char*) – Mode to be set.

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

Return type
unsigned byte

setUpstream(*port*)

Sets the upstream port.

Parameters
port (*const unsigned char*) – The upstream port to set.

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

Return type
unsigned byte

setUpstreamHS(*port*)

Sets the USB HighSpeed upstream port.

Parameters
port (*const unsigned char*) – The upstream port to set.

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

Return type
unsigned byte

setUpstreamSS(*port*)

Sets the USB SuperSpeed upstream port.

Parameters
port (*const unsigned char*) – The upstream port to set.

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

Return type
unsigned byte

3.2.32 Version

Provides version access utilities.

brainstem.version.**get_version_string**(*packed_version=None*, *buffer_length=256*)

Gets the version string from a packed version.

Parameters

- **packed_version** (*unsigned int*) – If version is provided, it is unpacked and presented as the version string. Most useful for printing the firmware version currently installed on a module.
- **buffer_length** (*unsigned short*) – The amount of C memory to allocate

Returns
The library version as a string

Return type
str

brainstem.version.**unpack_version**(*packed_version*)

Unpacks a packed version.

Parameters
packed_version (*unsigned int*) – The packed version number.

Returns
Returns the library version as a 3-tuple (major, minor, patch)

Return type
str

3.3 C++ API Reference

3.3.1 Acroname Modules

USBHub3c

Class

class **aUSBHub3c** : public Acroname::BrainStem::*Module*

Concrete Module implementation of a USBHub3c Allows a user to connect to and control an attached hub.

Public Types

enum **PORT_ID**

Port ID

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_4**

enumerator **kPORT_ID_5**

enumerator **kPORT_ID_CONTROL**

enumerator **kPORT_ID_POWER_C**

typedef enum *aUSBHub3c*::*PORT_ID* **PORT_ID_t**

Port ID

Public Members

HubClass **hub**
Hub Class

Acroname::BrainStem::*AppClass* **app**[aUSBHUB3C_NUM_APPS]
App Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBHUB3C_NUM_POINTERS]
Pointer Class

Acroname::BrainStem::*PowerDeliveryClass* **pd**[aUSBHUB3C_NUM_PD_PORTS]
Power Delivery Class

Acroname::BrainStem::*RailClass* **rail**[aUSBHUB3C_NUM_RAILS]
Rail Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB3C_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass* **temperature**[aUSBHUB3C_NUM_TEMPERATURES]
Temperature Class

Acroname::BrainStem::*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_PORTS 8
: Number of Ports available

aUSBHUB3C_NUM_PD_PORTS 8
Number of PD compatible ports available

aUSBHUB3C_NUM_PD_RULES_PER_PORT 7
Number of PD Rules per port available

aUSBHUB3C_NUM_RAILS 7
Number of Rail instances available

aUSBHUB3C_NUM_I2C 1
Number of I2C instances available

aUSBHUB3C_NUM_UART 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 Types

enum **PORT_ID**
Port ID 3p

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_4**

enumerator **kPORT_ID_5**

enumerator **kPORT_ID_6**

enumerator **kPORT_ID_7**

enumerator **kPORT_ID_DWNA**

enumerator **kPORT_ID_UP0**

enumerator **kPORT_ID_UP1**

enumerator **kPORT_ID_CONTROL**

typedef enum *aUSBHub3p*::*PORT_ID* **PORT_ID_t**
Port ID 3p

Public Members

HubClass **hub**
Hub Class

Acroname::BrainStem::*AppClass* **app**[aUSBHUB3P_NUM_APPS]
App Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBHUB3P_NUM_POINTERS]
Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB3P_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass* **temperature**
Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBHUB3P_NUM_TIMERS]
Timer Class

Acroname::BrainStem::*USBClass* **usb**
USB Class

class **HubClass** : public Acroname::BrainStem::*USBSystemClass*
Hub class implementation for use with USBHub3p.

Defines

aUSBHUB3P_MODULE 6
USBHub3p module number

aUSBHUB3P_NUM_APPS 4
Number of App instances available

aUSBHUB3P_NUM_POINTERS 4
Number of Pointer instances available

aUSBHUB3P_NUM_STORES 2
Number of Store instances available

aUSBHUB3P_NUM_INTERNAL_SLOTS 12
Store: Number of internal slots instances available

aUSBHUB3P_NUM_RAM_SLOTS 1
Store: Number of RAM slot instances available

aUSBHUB3P_NUM_TIMERS 8
Number of Timer instances available

aUSBHUB3P_NUM_USB 1
Number of USB instances available

aUSBHUB3P_NUM_USB_PORTS 8
Number of USB ports available

aUSBHUB3P_NUM_PORTS 12
Number of Prts available

Port State Defines

aUSBHUB3P_USB_VBUS_ENABLED 0
USB VBUS current state

aUSBHUB3P_USB2_DATA_ENABLED 1
USB2 data current state

aUSBHUB3P_USB3_DATA_ENABLED 3
USB3 data current state

aUSBHUB3P_USB_SPEED_USB2 11
USB2 speed current state

aUSBHUB3P_USB_SPEED_USB3 12
USB3 speed current state

aUSBHUB3P_USB_ERROR_FLAG 19

Error indicator for this port

(see 'Port Errors' below)

aUSBHUB3P_USB2_BOOST_ENABLED 20
USB2 boost current state

aUSBHUB3P_DEVICE_ATTACHED 23
Device attached indicator for this port

Port State Error Defines

aUSBHUB3P_ERROR_VBUS_OVERCURRENT 0
VBUS overcurrent error

aUSBHUB3P_ERROR_VBUS_BACKDRIVE 1
VBUS backdrive (backpower) error

aUSBHUB3P_ERROR_HUB_POWER 2
Hub power error

aUSBHUB3P_ERROR_OVER_TEMPERATURE 3
Over temperature error

aUSBHUB3P_ERROR_DISCHARGE_ERR 4
For compat with USBHub2x4

aUSBHUB3P_ERROR_SHORT_CIRCUIT 5
Short circuit detected

USBHub2x4

Class

class **aUSBHub2x4** : public Acroname::BrainStem::*Module*

Concrete Module implementation of a USBHub2x4 Allows a user to connect to and control an attached hub.

Public Types

enum **PORT_ID**

Port ID 2x4

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_UP0**

enumerator **kPORT_ID_UP1**

typedef enum *aUSBHub2x4*::*PORT_ID* **PORT_ID_t**

Port ID 2x4

Public Members

HubClass **hub**

Hub Class

Acroname::BrainStem::*AppClass* **app**[aUSBHUB2X4_NUM_APPS]

App Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBHUB2X4_NUM_POINTERS]

Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB2X4_NUM_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass* **temperature**
Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBHUB2X4_NUM_TIMERS]
Timer Class

Acroname::BrainStem::*USBClass* **usb**
USB Class

class **HubClass** : public Acroname::BrainStem::*USBSystemClass*
Hub class implementation for use with USBHub2x4.

Defines

aUSBHUB2X4_MODULE 6
USBHub2x4 module number

aUSBHUB2X4_NUM_APPS 4
Number of App instances available

aUSBHUB2X4_NUM_POINTERS 4
Number of Pointer instances available

aUSBHUB2X4_NUM_STORES 2
Number of Store instances available

aUSBHUB2X4_NUM_INTERNAL_SLOTS 12
Store: Number of internal slots instances available

aUSBHUB2X4_NUM_RAM_SLOTS 1
Store: Number of RAM slot instances available

aUSBHUB2X4_NUM_TIMERS 8
Number of Timer instances available

aUSBHUB2X4_NUM_USB 1
Number of USB instances available

aUSBHUB2x4_NUM_USB_PORTS 4
Number of USB ports available

aUSBHUB2x4_NUM_PORTS 6
    Number of Ports available

Port State Defines

aUSBHUB2X4_USB_VBUS_ENABLED 0
    USB VBUS current state

aUSBHUB2X4_USB2_DATA_ENABLED 1
    USB2 data current state

aUSBHUB2X4_USB_ERROR_FLAG 19

    Error indicator for this port

    (see 'Port Errors' below)

aUSBHUB2X4_USB2_BOOST_ENABLED 20
    USB2 boost current state

aUSBHUB2X4_DEVICE_ATTACHED 23
    Device attached indicator for this port

aUSBHUB2X4_CONSTANT_CURRENT 24
    Constant current mode indicator

Port State Error Defines

aUSBHUB2X4_ERROR_VBUS_OVERCURRENT 0
    VBUS overcurrent error

aUSBHUB2X4_ERROR_OVER_TEMPERATURE 3
    Over temperature error

aUSBHub2X4_ERROR_DISCHARGE 4
    Discharge error

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::*USBClass* **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 Types

enum **PORT_ID**
Port ID

Values:

enumerator **kPORT_ID_0**

enumerator **kPORT_ID_1**

enumerator **kPORT_ID_2**

enumerator **kPORT_ID_3**

enumerator **kPORT_ID_UP0**

typedef enum *aMTMIOSerial*::*PORT_ID* **PORT_ID_t**
Port ID

Public Members

HubClass **hub**
Hub Class

Acroname::BrainStem::*AppClass* **app**[aMTMIOSERIAL_NUM_APPS]
App Class

Acroname::BrainStem::*DigitalClass* **digital**[aMTMIOSERIAL_NUM_DIGITALS]
Digital Class

Acroname::BrainStem::*I2CClass* **i2c**[aMTMIOSERIAL_NUM_I2C]
I2C Class

Acroname::BrainStem::*UARTClass* **uart**[aMTMIOSERIAL_NUM_UART]
UART Class

Acroname::BrainStem::*PointerClass* **pointer**[aMTMIOSERIAL_NUM_POINTERS]
Pointer Class

Acroname::BrainStem::*RailClass* **rail**[aMTMIOSERIAL_NUM_RAILS]
Rail Class

Acroname::BrainStem::*RCServoClass* **servo**[aMTM_STEM_NUM_SERVOS]
RC Servo Class

Acroname::BrainStem::*SignalClass* **signal**[aMTMIOSERIAL_NUM_SIGNALS]
Signal Class

Acroname::BrainStem::*StoreClass* **store**[aMTMIOSERIAL_NUM_STORES]
Store Class

Acroname::BrainStem::*SystemClass* **system**
System Class

Acroname::BrainStem::*TemperatureClass* **temperature**
Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aMTMIOSERIAL_NUM_TIMERS]
Timer Class

Acroname::BrainStem::*USBClass* **usb**
USB Class

class **HubClass** : public Acroname::BrainStem::*USBSystemClass*
Hub class implementation for use with MTMIOSerial.

Defines

aMTMIOSERIAL_MODULE_BASE_ADDRESS 8
MTM-IO-Serial module number

aMTMIOSERIAL_NUM_APPS 4
Number of App instances available

aMTMIOSERIAL_NUM_DIGITALS 8
Number of Digital instances available

aMTMIOSERIAL_NUM_I2C 1
Number of I2C instances available

aMTMIOSERIAL_NUM_POINTERS 4
Number of Pointer instances available

aMTMIOSERIAL_NUM_RAILS 3
Number of Rail instances available

aMTMIOSERIAL_5VRAIL 0
Rail: 5v Rail specifier

aMTMIOSERIAL_ADJRAIL1 1
Rail: Adjustable Rail 0 specifier

aMTMIOSERIAL_ADJRAIL2 2
Rail: Adjustable Rail 1 specifier

aMTMIOSERIAL_MAX_MICROVOLTAGE 5000000
Rail: Max voltage in microvolts

aMTMIOSERIAL_MIN_MICROVOLTAGE 1800000
Rail: Min voltage in microvolts

aMTMIOSERIAL_NUM_SERVOS 8
Number of RC Servo instances available

aMTMIOSERIAL_NUM_SIGNALS 5
Number of Signal instances available

aMTMIOSERIAL_NUM_OUTPUT_SIGNALS 4
Signal: Number of output signal instances available

aMTMIOSERIAL_NUM_INPUT_SIGNALS 5
    Signal: Number of input signal instances available

aMTMIOSERIAL_NUM_STORES 2
    Number of Store instances available

aMTMIOSERIAL_NUM_INTERNAL_SLOTS 12
    Store: Number of internal slots instances available

aMTMIOSERIAL_NUM_RAM_SLOTS 1
    Store: Number of RAM slot instances available

aMTMIOSERIAL_NUM_TIMERS 8
    Number of Timer instances available

aMTMIOSERIAL_NUM_UART 4
    Number of UART instances available

aMTMIOSERIAL_NUM_USB 1
    Number of USB instances available

aMTMIOSERIAL_NUM_USB_PORTS 4
    Number of USB ports available

aMTMIOSERIAL_NUM_PORTS 5
    Number of Ports available

aMTMIOSERIAL_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

aMTMIOSERIAL_USB_VBUS_ENABLED 0
USB VBUS current state

aMTMIOSERIAL_USB2_DATA_ENABLED 1
USB2 data current state

aMTMIOSERIAL_USB_ERROR_FLAG 19

Error indicator for this channel

(see 'Port Errors' below)

aMTMIOSERIAL_USB2_BOOST_ENABLED 20
USB2 boost current state

Port State Error Defines

aMTMIOSERIAL_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

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

3.3.2 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 **setEnable** (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

3.3.3 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.

Return values

- **aErrNone** – success.
- **aErrTimeout** – The request timed out waiting to start execution.
- **aErrConnection** – No active link connection.
- **aErrNotFound** – the app reflex was not found or not enabled on the module.

aErr **execute** (const uint32_t appParam, uint32_t *returnVal, const uint32_t msTimeout = 1000)

Execute the app reflex on the module. Wait for a return from the reflex execution for msTimoue milliseconds. This method will block for up to msTimeout.

Parameters

- **appParam** – The app parameter handed to the reflex.
- **returnVal** – The return value filled in from the result of executing the reflex routine.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

Return values

- **aErrNone** – success.
- **aErrTimeout** – The request timed out waiting for a response.
- **aErrConnection** – No active link connection.
- **aErrNotFound** – the app reflex was not found or not enabled on the module.

3.3.4 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

3.3.5 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 – The configuration to be applied

- Digital Input: digitalConfigurationInput = 0
- Digital Output: digitalConfigurationOutput = 1
- RCServo Input: digitalConfigurationRCServoInput = 2
- RCServo Output: digitalConfigurationRCServoOutput = 3
- High Z State: digitalConfigurationHiZ = 4
- Digital Input: digitalConfigurationInputPullUp = 0
- Digital Input: digitalConfigurationInputNoPull = 4
- Digital Input: digitalConfigurationInputPullDown = 5

Return values

aErrConfiguration – Entity does not support this configuration.

Returns

Returns *common entity* return values

aErr **getConfiguration** (uint8_t *configuration)

Get the digital configuration.

Parameters

configuration – - Current configuration of the digital entity.

Returns

Returns *common entity* return values

aErr **setState** (const uint8_t state)

Set the logical state.

Parameters

state – The state to be set. 0 is logic low, 1 is logic high.

Returns

Returns *common entity* return values

aErr **getState** (uint8_t *state)

Get the state.

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

3.3.6 Entity Class

class **EntityClass**

EntityClass: The *EntityClass* is the base class for interacting with BrainStem UEI entities. All BrainStem UEI classes inherit from *EntityClass*. Advanced users may use *EntityClass* to extend BrainStem functionality specific to their needs.

Subclassed by *Acroname::BrainStem::AnalogClass*, *Acroname::BrainStem::AppClass*, *Acroname::BrainStem::ClockClass*, *Acroname::BrainStem::DigitalClass*, *Acroname::BrainStem::EqualizerClass*, Acroname::BrainStem::EthernetClass, *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*

Public Functions

EntityClass(void)

Constructor.

virtual **~EntityClass**(void)

Destructor.

void **init**(*Module* *pModule, const uint8_t command, const uint8_t index)

init.

Initialize the entity class.

Parameters

- **pModule** – The BrainStem module object.

- **command** – The command of the UEI.
- **index** – The index of the UEI entity.

aErr **callUEI** (const uint8_t option)

A callUEI is a setUEI that has no data length.

Parameters

option – An option for the UEI.

Returns

Returns *common entity* return values

aErr **setUEI8** (const uint8_t option, const uint8_t byteValue)

Set a byte value.

Parameters

- **option** – The option for the UEI.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI8** (const uint8_t option, const uint8_t param, const uint8_t byteValue)

Set a byte value with a subindex.

Parameters

- **option** – The option for the UEI.
- **param** – of the option.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **getUEI8** (const uint8_t option, uint8_t *byteValue)

Get a byte value.

Parameters

- **option** – The option for the UEI.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **getUEI8** (const uint8_t option, const uint8_t param, uint8_t *byteValue)

Get a byte value with a parameter.

Parameters

- **option** – The option for the UEI.
- **param** – The parameter.
- **byteValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI16** (const uint8_t option, const uint16_t shortValue)

Set a 2-byte value.

Parameters

- **option** – The option for the UEI.
- **shortValue** – The value.

Returns

Returns *common entity* return values

aErr **setUEI16** (const uint8_t option, const uint8_t param, const uint16_t shortValue)

Set a 2-byte value with a parameter.

Parameters

- **option** – The option for the UEI.

- **`param`** – The parameter.
- **`shortValue`** – The value.

Returns
Returns *common entity* return values

aErr **`getUEI16`** (const uint8_t option, uint16_t *shortValue)

Get a 2-byte value.

Parameters
- **`option`** – The option for the UEI.
- **`shortValue`** – The value.

Returns
Returns *common entity* return values

aErr **`getUEI16`** (const uint8_t option, const uint8_t param, uint16_t *shortValue)

Get a 2-byte value with a parameter.

Parameters
- **`option`** – The option for the UEI.
- **`param`** – The parameter.
- **`shortValue`** – The value.

Returns
Returns *common entity* return values

aErr **`setUEI32`** (const uint8_t option, const uint32_t intValue)

Set a 4-byte value.

Parameters
- **`option`** – The option for the UEI.
- **`intValue`** – The value.

Returns
Returns *common entity* return values

aErr **`setUEI32`** (const uint8_t option, const uint8_t subIndex, const uint32_t intValue)

Set a 4-byte value, with a subindex parameter.

Parameters
- **`option`** – The option for the UEI.
- **`subIndex`** – The subindex to set.
- **`intValue`** – The value.

Returns
Returns *common entity* return values

aErr **`getUEI32`** (const uint8_t option, uint32_t *intValue)

Get a 4-byte value.

Parameters
- **`option`** – The option for the UEI.
- **`intValue`** – The 4 byte value

Returns
Returns *common entity* return values

aErr **`getUEI32`** (const uint8_t option, const uint8_t param, uint32_t *intValue)

Get a 4-byte value with parameter.

Parameters
- **`option`** – The option for the UEI.
- **`param`** – The parameter.
- **`intValue`** – The 4 byte value

Returns
Returns *common entity* return values

aErr **setUEIBytes** (const uint8_t option, const uint8_t *bufPtr, const size_t bufLen)

Set a multi-byte value.

Parameters

- **option** – The option for the UEI.
- **bufPtr** – The pointer to a data buffer
- **bufLen** – The length of the data buffer

Returns

Returns *common entity* return values

aErr **getUEIBytes** (const uint8_t option, uint8_t *buf, const size_t bufLength, size_t *unloadedLength)

Unloads UEI Bytes data as byte data

Parameters

- **option** – The option for the UEI.
- **buf** – Start of where data should be stored..
- **bufLength** – Size of the buffer
- **unloadedLength** – Amount of data unloaded (in bytes)

Returns

Returns *common entity* return values

aErr **getUEIBytesCheck** (size_t *unloadedLength, const size_t valueSize)

Parameters

- **unloadedLength** – Amount of data unloaded (in bytes)
- **valueSize** – The base type size in this array

Returns

Returns *common entity* return values

uint8_t **getIndex** (void) const

Get the UEI entity index.

Returns

The 1 byte index of the UEI entity.

aErr **drainUEI** (const uint8_t option)

Drain all packets matching this UEI from the packet fifo.

This functionality is useful in rare cases where packet synchronization is lost and a valid return packet is not accessible.

aErr **setStreamEnabled** (uint8_t enabled)

Enables streaming for all possible option codes within the cmd and index the entity was created for.

Parameters

enabled – The state to be applied. 0 = Disabled; 1 = enabled

Returns

Returns *common entity* return values

aErr **registerOptionCallback** (const uint8_t option, const bool enable, *Link*::*streamCallback_t* cb, void *pRef)

Registers a callback function based on a specific option code. Option code applies to the cmd and index of the called API.

Parameters

- **option** – option to filter by (supports Wildcards)
- **enable** – True - installs/updates callback and ref; False - uninstalls callback
- **cb** – Callback to be executed when a new packet matching the criteria is received.
- **pRef** – Pointer to user reference for use inside the callback function.

Returns
aErrNotFound - Item not found (uninstalling only)

Returns
aErrNone - success

aErr **getStreamStatus**(*Link*::*StreamStatusEntry_t* *buffer, const size_t bufferLength, size_t *unloadedSize)

Gets all available stream values associated with the cmd and index of the called API. Keys can be decoded with *Link::getStreamKeyElement*.

Parameters
- **buffer** – Buffer of user allocated memory to be filled with stream data
- **bufferLength** – Number of elements the buffer can hold.
- **unloadedSize** – Number of elements that were placed in the buffer

Returns
aErrParam if status or unloadedSize is null

Returns
aErrResource - if the link is not valid

Returns
aErrNone - success

Public Static Functions

static uint8_t **getUEIBytesSequence**(const *aPacket* *packet)

Parameters
packet – UEI packet to be checked/filtered.

Returns
The sequence number of the byte.

static bool **getUEIBytesContinue**(const *aPacket* *packet)

Parameters
packet – UEI packet to be checked/filtered.

Returns
True - Continue bit is set (more packets to come); False - Continue bit is not set (first or last packet).

static uint8_t **sUEIBytesFilter**(const *aPacket* *packet, const void *ref)

Filter function for UEI Bytes calls. Exposed for unit-testing purposes only.

Parameters
- **packet** – UEI packet to be checked/filtered.
- **ref** – Opaque reference handle

group **EntityReturnValues**

Common *EntityClass* Return Values.

- *aErrNone* - Action completed successfully.
- *aErrTimeout* - Request timed out without response.
- *aErrConnection* - No active link.

3.3.7 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.

3.3.8 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 readLength, uint8_t *buffer)

Read from a device on this I2C bus.

Parameters

- **address** – - The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **readLength** – - The length of the data to read in bytes.
- **buffer** – - The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAINSTEM_MAXPACKETBYTES - 5

Returns

Returns *common entity* return values

aErr **write** (const uint8_t address, const uint8_t bufferLength, const uint8_t *buffer)

Write to a device on this I2C bus.

Parameters

- **address** – - The I2C address (7bit <XXXX-XXX0>) of the device to write.
- **bufferLength** – - The length of the data to write in bytes.
- **buffer** – - The data to send to the device, This array should be no larger than aBRAINSTEM_MAXPACKETBYTES - 5

Returns

Returns *common entity* return values

aErr **setPullup** (const 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

3.3.9 Link Class

class **Link**

LinkClass: The *Link* class provides an interface to a BrainStem link. The link is used to create interfaces to modules on a BrainStem network. The link represents a connection to the BrainStem network from a host computer. The link is always associated with a transport (e.g.: USB, Ethernet, etc.) and a link module, but there are several ways to make this association.

a. The link can be fully specified with a transport and module serial number

b. The link can be created by searching a transport and connecting to the first module found.

Calling connect on a link will start a connection to the module based on The link specification. Calling disconnect will disconnect the link from the the current connection.

Public Types

enum **STREAM_PACKET**

Enumeration of stream packet types.

Values:

enumerator **kSTREAM_PACKET_UNKNOWN**

enumerator **kSTREAM_PACKET_U8**

enumerator **kSTREAM_PACKET_U16**

enumerator **kSTREAM_PACKET_U32**

enumerator **kSTREAM_PACKET_BYTES**

enumerator **kSTREAM_PACKET_SUBINDEX_U8**

enumerator **kSTREAM_PACKET_SUBINDEX_U16**

enumerator **kSTREAM_PACKET_SUBINDEX_U32**

enumerator **kSTREAM_PACKET_LAST**

enum **STREAM_KEY**

Enumeration for element types within a stream key.

Values:

enumerator **STREAM_KEY_MODULE_ADDRESS**

enumerator **STREAM_KEY_CMD**

enumerator **STREAM_KEY_OPTION**

enumerator **STREAM_KEY_INDEX**

enumerator **STREAM_KEY_SUBINDEX**

enumerator **STREAM_KEY_LAST**

typedef enum Acroname::BrainStem::*Link*::*STREAM_PACKET* **STREAM_PACKET_t**

Enumeration of stream packet types.

typedef std::function<uint8_t(const *aPacket* *packet, void *pRef)> **streamCallback_t**

Function signature for streaming callbacks.

Param packet
reference to streaming packet

Param pRef
User provided reference

Return
non zero error code on failure. Return value is not currently used.

typedef struct Acroname::BrainStem::*Link*::*StreamStatusEntry* **StreamStatusEntry_t**

StreamStatusEntry structure - It contains members of streaming entries in the form of key value pairs. Keys are comprised of the devices module address, command, option, index, and subindex API values.

typedef enum Acroname::BrainStem::*Link*::*STREAM_KEY* **STREAM_KEY_t**

Enumeration for element types within a stream key.

Public Functions

aErr **getConfig**(aEtherConfig *config)

Gets the links current aEther configuration

Parameters

config – Pointer to the configuration to be filled

Returns

aErrNone on success; aErrParam if config is NULL

aErr **setConfig**(const aEtherConfig config)

Sets the links aEther configuration. Configuration must be applied BEFORE connecting

Parameters

config – Configuration to be applied

Returns

aErrNone on success. aErrPermission if the module is currently connected.

Link(const *linkSpec* linkSpecifier, const char *name = "Link")

Link Constructor. Takes a fully specified *linkSpec* pointer and creates a link instance with this specifier information.

Parameters

- **linkSpecifier** – The connection details for a specific module.
- **name** – A name for the link to be created. This name can be used to reference the link during later interactions.

Link(const char *name = "Link")

Link constructor without a specifier will most likely use the discoverAndConnect call to create a connection to a link module.

Parameters

name – A name for the link to be created.

~Link(void)

Destructor.

aErr **discoverAndConnect**(const *linkType* type, const uint32_t serialNumber = 0, const uint8_t model = 0)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will be made.

Parameters

- **type** – Transport on which to search for available BrainStem link modules. See the *transport* enum for supported transports.
- **serialNumber** – Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.
- **model** – Acroname model number for the device.

Return values

- **aErrBusy** – if the module is already in use.
- **aErrParam** – if the transport type is undefined.
- **aErrNotFound** – if the module cannot be found or if no modules found.
- **aErrNone** – If the connect was successful.

aErr **connect**(void)

Connect to a link with a fully defined specifier.

Return values

- **aErrBusy** – if the module is running, starting or stopping. Try again in a bit.
- **aErrDuplicate** – If the module is already connected and running.
- **aErrConnection** – If there was an error with the connection. User needs to disconnect, then reconnect.
- **aErrConfiguration** – If the link has an invalid *linkSpec*.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr **connectThroughLinkModule** (*Link* &link)

Connect using a pre-existing link. This member function will connect to the same BrainStem used by given *Link*. If a link module is found on the specified transport, a connection will be made.

Parameters

link – - Reference to the link to be used.

Return values

- **aErrInitialization** – If the referenced link does not exist yet.
- **aErrConnection** – If the connection could not be made.
- **aErrConfiguration** – If the device or connection is in properly configured.
- **aErrNone** – if the connect was successful.

bool **isConnected** (void)

Check to see if a module is connected. isConnected looks for a connection to an active module.

Returns

true: connected, false: not connected.

linkStatus **getStatus** (void)

Check the status of the module connection.

Returns

linkStatus (see aLink.h for status values)

aErr **disconnect** (void)

Disconnect from the BrainStem module.

Return values

- **aErrResource** – If the there is no valid connection.
- **aErrConnection** – If the disconnect failed, due to a communication issue.
- **aErrNone** – If the disconnect was successful.

aErr **reset** (void)

Reset The underlying link stream.

Return values

- **aErrResource** – If the there is no valid connection.
- **aErrConnection** – If the reset failed, due to a communication issue.
- **aErrNone** – If the reset was successful.

const char ***getName** (void)

Accessor for link Name. Returns a pointer to the string representing the link. This string is part of the link, and will be destroyed with it. If you need access to the link name beyond the life of the link, then copy the char* returned.

Returns

Pointer to character array containing the name of the link.

aErr **getLinkSpecifier** (*linkSpec* *spec)

Accessor for current link specification.

Parameters

`spec` – - an allocated empty link spec reference.

Returns

aErrNotFound - If no *linkSpec* set for current link.

aErr **setLinkSpecifier**(const *linkSpec* linkSpecifier)

Accessor Set current link specification.

Parameters

`linkSpecifier` – - The specifier that will replace the current spec.

Returns

aErrBusy - If link is currently connected.

aErr **getModuleAddress**(uint8_t *address)

Gets the module address of the module the link is connected too. A zero is returned if no module can not be determined or if the link is not connected.

aErr **sendUEI**(const *uei* packet)

Sends a BrainStem protocol UEI packet on the link. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

`packet` – The command UEI packet to send.

Return values

- `aErrConnection` – link not connected.
- `aErrParam` – data too long or short.
- `aErrPacket` – invalid module address.
- `aErrNone` – success.

aErr **sendUEI**(const *uei* packet, const uint8_t subindex)

Sends a BrainStem protocol UEI packet on the link where the packet contains a subindex. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- `packet` – The command UEI packet to send.
- `subindex` – The subindex of the command option.

Return values

- `aErrConnection` – link not connected.
- `aErrParam` – data too long or short.
- `aErrPacket` – invalid module address.
- `aErrNone` – success.

aErr **receiveUEI**(const uint8_t module, const uint8_t command, const uint8_t option, const uint8_t index, *uei* *packet)

Awaits receipt of the first available matching UEI packet from the link. The first four arguments describe the packet to wait for. When successful, the supplied uei ref is filled with the received UEI. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- `module` – The module address.
- `command` – The command.
- `option` – The uei option.
- `index` – The index of the uei entity.
- `packet` – The uei packet reference to be filled on success.

Return values

- `aErrConnection` – link not connected.
- `aErrPacket` – invalid module address.

- **aErrTimeout** – no packet available.
- **aErrNone** – success.

aErr **receiveUEI** (const uint8_t module, const uint8_t command, const uint8_t option, const uint8_t index, *uei* *packet, aPacketMatchPacketProc proc)

Awaits receipt of the first available matching UEI packet from the link. The first four arguments and proc describe the packet to wait for. When successful, the supplied uei ref is filled with the received UEI. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- **module** – The module address.
- **command** – The command.
- **option** – The uei option.
- **index** – The index of the uei entity.
- **packet** – The uei packet reference to be filled on success.
- **proc** – The callback used for determining a matching packet.

Return values

- **aErrConnection** – link not connected.
- **aErrPacket** – invalid module address.
- **aErrTimeout** – no packet available.
- **aErrNone** – success.

aErr **dropMatchingUEIPackets** (const uint8_t module, const uint8_t command, const uint8_t option, const uint8_t index)

Drops all existing queued packets that match. from the link. The arguments describe the packets to be matched This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

Parameters

- **module** – The module address.
- **command** – The command.
- **option** – The uei option.
- **index** – The index of the uei entity.

Return values

- **aErrConnection** – link not connected.
- **aErrPacket** – invalid module address.
- **aErrNone** – success.

aErr **sendPacket** (const uint8_t module, const uint8_t command, const uint8_t length, const uint8_t *data)

Sends a raw BrainStem protocol packet on the link. where the length does not include the module or the command. address byte and can be 0 to aBRAINSTEM_MAXPACKETBYTES - 1. This is an advanced interface, please see the relevant section of the reference manual for more information about BrainStem Packet protocol.

Parameters

- **module** – The address of the destination module.
- **command** – The length of the data being sent.
- **length** – The length of the data being sent.
- **data** – The data to send.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – data too long or short.
- **aErrPacket** – invalid module address.
- **aErrNone** – success.

aErr **receivePacket** (const uint8_t module, const uint8_t *match, uint8_t *length, uint8_t *data)

Awaits receipt of the first available matching raw BrainStem protocol packet from the link where the length does not include the module or command bytes and can be zero. The provided module and match array are compared to packets available and the first match is returned. The supplied data pointer must point to at least aBRAINSTEM_MAXPACKETBYTES - 1 bytes. When successful, the data is filled in with the packet data not including the module and command and the length pointer is updated with the length of the returned data.

This is an advanced interface, please see the relevant section of the reference manual for more information about BrainStem Packet protocol.

Parameters

- **module** – The module address.
- **match** – A byte array of the values to match for received packets.
- **length** – The length of the match data on entry and length of the returned data filled on success.
- **data** – The data filled on success.

Return values

- **aErrConnection** – link not connected.
- **aErrPacket** – invalid module address.
- **aErrTimeout** – no packet available.
- **aErrNone** – success.

aErr **loadStoreSlot**(const uint8_t module, const uint8_t store, const uint8_t slot, const uint8_t *pData, const size_t length)

Loads data into a BrainStem Slot. See the relevant section of the BrainStem reference for information about BrainStem Slots and Stores.

Parameters

- **module** – - *Module* address.
- **store** – - BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** – - The Slot within the Brainstem store to place the data.
- **pData** – - Pointer to a buffer containing the data to load.
- **length** – - The length in bytes of the data buffer to write.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this call was unable to successfully complete.
- **aErrNone** – success.

aErr **unloadStoreSlot**(const uint8_t module, const uint8_t store, const uint8_t slot, uint8_t *pData, const size_t dataLength, size_t *pNRead)

Unloads data from a BrainStem Slot. If there are no read.

reference for information about BrainStem Slots and Stores.

Parameters

- **module** – - *Module* address.
- **store** – - BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** – - The Slot within the Brainstem store to place the data.
- **pData** – - Pointer to a buffer with dataLength space in bytes that will be filled by the call.
- **dataLength** – - Expected length of the data, and at most the size of the pData buffer.

- **pNRead** – - The number of bytes actually read.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this call was unable to successfully complete.
- **aErrOverrun** – The read would overrun the buffer, i.e there is more data in the slot than the buffer can handle.
- **aErrNone** – success.

aErr **storeSlotSize**(const uint8_t module, const uint8_t store, const uint8_t slot, size_t *size)

Returns the current size of the data loaded in the slot specified.

Parameters

- **module** – - *Module* address.
- **store** – - BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** – - The Slot within the Brainstem store to place the data.
- **size** – - size in bytes of the data stored in the slot.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this request was unable to successfully complete.
- **aErrNone** – success.

aErr **storeSlotCapacity**(const uint8_t module, const uint8_t store, const uint8_t slot, size_t *capacity)

Returns the maximum data capacity of the slot specified.

Parameters

- **module** – - *Module* address.
- **store** – - BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** – - The Slot within the Brainstem store to place the data.
- **capacity** – - size in bytes of the data stored in the slot.

Return values

- **aErrConnection** – link not connected.
- **aErrParam** – invalid module address.
- **aErrCancel** – The write process is closing and this request was unable to successfully complete.
- **aErrNone** – success.

aErr **enableStream**(const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const bool enable)

Enables streaming for the supplied criteria.

Parameters

- **moduleAddress** – Address to filter on.
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - Enables streaming; False - disables streaming

aErr **isLinkStreaming**(const uint8_t moduleAddress, uint8_t *enabled)

Determines if the module is actively streaming. Does not indicate what is streaming, only if streaming is currently active.

Parameters
- **moduleAddress** – The devices module address.
- **enabled** – Variable to be populated.

Returns
Returns *common entity* return values

aErr **registerStreamCallback**(const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const bool enable, *streamCallback_t* cb, void *pRef)

Registers a callback function based on a specific module, cmd, option, and index.

Parameters
- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - installs/updates callback and ref; False - uninstalls callback
- **cb** – Callback to be executed when a new packet matching the criteria is received.
- **pRef** – Pointer to user reference for use inside the callback function.

Returns
aErrNotFound - Item not found (uninstalling only)

Returns
aErrNone - success

aErr **getStreamValue**(const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const uint8_t subindex, uint32_t *value)

Gets stream value based on the search criteria

Parameters
- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)

Returns
aErrStreamStale if the value has not been updated since the last read.

Returns
aErrNotFound if no such stream element exists.

Returns
aErrNone - success

aErr **getStreamStatus**(const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const uint8_t subindex, *StreamStatusEntry_t* *buffer, const size_t bufferLength, size_t *unloadedSize)

Gets all available stream values based on the search criteria.

Parameters
- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **subindex** – subindex to filter by (supports Wildcards)
- **buffer** – Buffer of user allocated memory to be filled with stream data Note: *Link::getStreamKeyElement* should be used to decode the keys
- **bufferLength** – Number of elements the buffer can hold.
- **unloadedSize** – Number of elements that were placed in the buffer

Returns
aErrParam if status or unloadedSize is null

Returns
aErrNone - success

std::vector<uint64_t> **filterActiveStreamKeys**(const uint8_t moduleAddress, const uint8_t cmd, const uint8_t option, const uint8_t index, const uint8_t subindex, const bool acquireLock)

Provides a list of active stream keys based on the supplied criteria. Exposed for unit-testing purposes only.

Parameters
- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **acquireLock** – Option to acquire mutex before getting list elements.

Returns
List of keys meeting the search criteria

aErr **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 *vpCBRef, const uint32_t networkInterface = LOCALHOST_IP_ADDRESS)

Discover is called with a specified transport to search for link modules on that transport. The callback is called with a fully filled in specifier for any link module found. The sDiscover returns aErrNone if the discovery process is successful, regardless of if any links are found. An error is only returned if the link discovery process fails. Discovery can take some time. The callback will occur in the same thread context as this routine call.

Parameters

- **type** – Transport to search for available BrainStem link modules on. See the *transport* enum for supported transports.
- **cbLinkFound** – Process that is called when a module is discovered.
- **vpCBRef** – This is passed to cbLinkFound when a module is discovered.

Return values

- **aErrNone** – on success.
- **aErrNotFound** – if no devices were found.

static inline *bContinueSearch* **sFindAll** (const *linkSpec* *spec, bool *bSuccess, void *vpCBRef)

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.
- **vpCBRef** – Reference pointer to the std::list that was passed in.

Return values

- **true** – Caller should continue to call this function.
- **false** – Caller should stop calling this function.

static inline *aErr* **sDiscover** (const *linkType* type, list<*linkSpec*> *devices, const uint32_t networkInterface = LOCALHOST_IP_ADDRESS)

Discover is called with a specified transport to search for link modules on that transport. The devices list is filled with device specifiers. sDiscover returns aErrNone if the discovery process is successful, regardless of whether any links are found. An error is only returned if the link discovery process fails. Discovery can take some time.

Parameters

- **type** – Transport to search for available BrainStem link modules on. See the *transport* enum for supported transports.
- **devices** – an empty list of specifiers that will be filled in.

Return values

- **aErrNotFound** – if no devices were found.
- **aErrNone** – on success.

static bool **getStreamPacketType** (const *aPacket* *packet, *STREAM_PACKET_t* *type)

Decodes the streaming packet type from a provided packet.

Parameters

- **packet** – - The packet to be interrogated.
- **type** – - variable to be populated. Filled with kSTREAM_PACKET_UNKNOWN on failure.

Returns

true on success; false on failure.

static bool **isSubindexType** (*STREAM_PACKET_t* type)

Helper function for indicating wether 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 wether the value is a stream packet. Stream "Packets" encompass all STREAM_PACKET_t valid elements.

Parameters

packet – UEI stream packet to be checked.

Returns

Wether the packet is a stream sample or not

static bool **isStreamSample**(const *aPacket* *packet)

Convenience function to determine wether 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

Wether the packet is a stream sample or not

static *aErr* **getStreamSample**(const *aPacket* *packet, uint64_t *timestamp = NULL, uint32_t *value = NULL, uint8_t *subindex = NULL)

Convenience function to unpack the stream samples timestamp and value. Calling isStreamSample prior is not required as this function will verify the packet type.

Parameters

- **packet** – UEI stream packet to be unpacked.
- **timestamp** – Variable to be filled with stream sample timestamp. (optional)
- **value** – Variable to be filled with the stream sample (optional). May require casting to signed value depending on the cmd/option code.

Return values

- **aErrPacket** – Not a stream packet
- **aErrUnknown** – Unknown decoding issue.
- **aErrNone** – success.

static void **getTimestampParts**(const uint64_t timestamp, uint32_t *seconds, uint32_t *uSeconds)

Helper function for extracting the parts of a timestamp.

Parameters

- **timestamp** – - Value acquired from *Link::getStreamSample*
- **seconds** – - Seconds element from timestamp. Refers to the seconds since firmware boot.
- **uSeconds** – - Micro second element from the timestamp. Refers to the micro seconds from firmware boot. Micro seconds rolls over to seconds. Value range: 0-99999

static bool **linkStreamFilter**(const *aPacket* *packet, void *ref)

Filter function for Streaming packets. This is used internally whenever streaming is enabled. Exposed for unit-testing purposes only.

Parameters

- **packet** – UEI stream packet to be checked/filtered.

- **ref** – Opaque reference handle

struct **StreamStatusEntry**

StreamStatusEntry structure - It contains members of streaming entries in the form of key value pairs. Keys are comprised of the devices module address, command, option, index, and subindex API values.

Public Members

uint64_t **key**

The stream key.

uint32_t **value**

The value associated with the key

3.3.10 Module Class

class **Module**

ModuleClass: The *Module* class provides a generic interface to a BrainStem hardware module. The *Module* class is the parent class for all BrainStem modules. Each module inherits from *Module* and implements its hardware specific features.

Subclassed by a40PinModule, aMTMDAQ1, *aMTMDAQ2*, *aMTMIOSerial*, *aMTMLoad1*, *aMTMPM1*, *aMTMRelay*, *aMTMStemModule*, aPD3M, *aUSBCSwitch*, aUSBCSwitchPro, *aUSBHub2x4*, *aUSBHub3c*, *aUSBHub3p*

Public Functions

Module (const uint8_t address, const uint8_t model = 0)

Constructor. Implicitly creates a link object with no specifier. Most often objects created with this constructor will use linkDiscoverAndConnect to find and connect to a module.

Parameters

- **address** – The BrainStem network address of the module. The default address (or base address for modules that support address offsets) is defined in each module's "Defs.h" header.
- **model** – Acroname model number.

virtual **~Module** (void)

Destructor.

aErr **connect** (const *linkType* type, const uint32_t serialNum)

Connect using the current link specifier.

Parameters

- **type** – - Transport on which to search for available BrainStem link modules. See the *transport* enum for supported transports.
- **serialNum** – - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Return values

- **aErrBusy** – if the module is already in use.

- **aErrParam** – if the type is incorrect or serialNum is not specified
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr **connectFromSpec** (const *linkSpec* linkSpecifier)

Connect to a link with a fully defined specifier.

Parameters

linkSpecifier – - Connect to module with specifier.

Return values

- **aErrInitialization** – If there is currently no link object.
- **aErrBusy** – If the link is currently connected.
- **aErrParam** – if the specifier is incorrect.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr **discoverAndConnect** (*linkType* type, const uint32_t serialNum = 0)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will

Parameters

- **type** – - Transport on which to search for available BrainStem link modules. See the *transport* enum for supported transports.
- **serialNum** – - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Return values

- **aErrBusy** – if the module is already in use.
- **aErrParam** – if the transport type is undefined.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

aErr **connectThroughLinkModule** (*Module* *pModule)

Connect using link from another *Module*. This member function will connect to the same BrainStem used by given *Module*. If a link module is found on the specified transport, a connection will be made

Parameters

pModule – - Pointer to a valid *Module* class object.

Return values

- **aErrParam** – if the module is undefined.
- **aErrNone** – if the connect was successful.

bool **isConnected** (void)

Is the link connected to the BrainStem *Module*.

linkStatus **getStatus** (void)

Check the status of the BrainStem module connection.

Returns

linkStatus (see aLink.h for status values)

aErr **disconnect** (void)

Disconnect from the BrainStem module.

Return values

- **aErrResource** – If the there is no valid connection.
- **aErrConnection** – If the disconnect failed, due to a communication issue.
- **aErrNone** – If the disconnect was successful.

aErr **reconnect**()

Reconnect using the current link specifier.

Return values

- **aErrBusy** – if the module is already in use.
- **aErrParam** – if the specifier is incorrect.
- **aErrNotFound** – if the module cannot be found.
- **aErrNone** – If the connect was successful.

Link ***getLink**(void) const

Get the current link object.

Returns

The link associated with the module.

aErr **getConfig**(aEtherConfig *config)

Gets the links current network configuration

Parameters

config – Variable to be filled with the config

Returns

aErrNone on success. aErrNotReady if the module does not have a link aErrParam if config is NULL

aErr **setConfig**(const aEtherConfig config)

Sets the links network configuration. Configuration must be applied BEFORE connecting

Parameters

config – Configuration to be applied

Returns

aErrNone on success. aErrPermission if the module is currently connected. aErrNotReady if the module does not have a link

uint8_t **getModuleAddress**(void) const

Accessor to get the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

Returns

The module address.

void **setModuleAddress**(const uint8_t address)

Accessor to set the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

Parameters

address – The module address.

aErr **getLinkSpecifier**(*linkSpec* *spec)

Get linkSpecifier

Parameters

spec – - allocated linkspec struct will be filled with spec.

Returns

aErrNone - If the module does not have a spec.

aErr **getBuild**(uint32_t *build)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Parameters

build – Variable to be filled with build.

aErr **hasUEI**(const uint8_t command, const uint8_t option, const uint8_t index, const uint8_t flags)

Queries the module to determine if it implements a UEI. Each UEI has a command, option or variant, index and flag. The hasUEI method queries for a fully specified UEI. Returns aErrNone if the variation is supported and an appropriate error if not. This call is blocking for up to the nMSTimeout period.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **option** – The option or variant of the command.
- **index** – The entity index.
- **flags** – The flags (ueiOPTION_SET or ueiOPTION_GET).

Return values

- **aErrNone** – The module supports this command and access flags.
- **aErrMode** – The module supports the command but not the access flag.
- **aErrNotFound** – The module does not support the command, option, or index.
- **aErrTimeout** – The request timed out without a response.
- **aErrConnection** – There is no active link

aErr **classQuantity**(const uint8_t command, uint8_t *count)

Queries the module to determine how many entities of the specified class are implemented by the module. Zero is a valid return value. For example, calling classQuantity with the command parameter of cmdANALOG would return the number of analog entities implemented by the module.

Parameters

- **command** – One of UEI commands (cmdXXX).
- **count** – When the request is successful count is updated with the number of entities found.

Return values

- **aErrNone** – Success.
- **aErrTimeout** – The request timed out without a response.
- **aErrConnection** – There is no active link.

aErr **subClassQuantity**(const uint8_t command, const uint8_t index, uint8_t *count)

Queries the module to determine how many subclass entities of the specified class are implemented by the module for a given entity index. This is used for entities which may be 2-dimensional. E.g. cmdMUX subclasses are the number of channels supported by a particular mux type (index); as a specific example, a module may support 4 UART channels, so subClassQuantity(cmdMUX, aMUX_UART…) could return 4. Zero is a valid return value.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **count** – The number of subclasses found.

Return values

- **aErrNone** – Success.
- **aErrTimeout** – The request timed out waiting for response.
- **aErrConnection** – There is no active link.

aErr **entityGroup**(const uint8_t command, const uint8_t index, uint8_t *group)

Queries the module the group assigned to an entity and index. Entities groups are used to specify when certain hardware features are fundamentally related. E.g. certain hardware modules may have some digital pins associated with an adjustable voltage rail; these digitals would be in the same group as the rail. Zero is the default group.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **group** – Upon success, group is filled with the entities group value.

Return values

- **aErrNone** – Success.
- **aErrTimeout** – The request timed out without response.
- **aErrConnection** – There is no active link.

aErr **debug**(const uint8_t *pData, const uint8_t length)

Sends a debug packet to the module containing the provided data. Modules receiving debug packets simply echo the packet back to the sender. If the round-trip is successful, the reply data will match the data sent. This method returns aErrNone when successful, if not successful, an appropriate error is returned.

Parameters

- **pData** – A pointer to an array of data to be sent in the debug packet.
- **length** – The length of the data array.

Return values

- **aErrNone** – Success.
- **aErrTimeout** – Timeout occurred without response.
- **aErrConnection** – No active link exists.

void **setNetworkingMode**(const bool mode)

Sets the networking mode of the module object. By default the module object is configure to automatically adjust its address based on the devices current module address. So that, if the device has a software or hardware offset it will still be able to communication with the device. If advanced networking is required the auto networking mode can be turned off.

Parameters

mode – True/1 for Auto Networking, False/0 for manual networking

3.3.11 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 on or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

Public Functions

MuxClass (void)

Constructor.

~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 chanel 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

3.3.12 PDChannelLogger

Warning: doxygenclass: Cannot find class "Acroname::Utilities::PDChannelLogger" in doxygen xml output for project "BrainStem" from directory: doxml/xml

3.3.13 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 an 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 transferLength)
Transfer data to the store.

Parameters
transferLength – The length of the data transfer.

Returns
All possible standard UEI return values.

aErr **initiateTransferFromStore**(uint8_t transferLength)
Transfer data from the store.

Parameters
transferLength – 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.

3.3.14 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 getDataHSEnabled.

Parameters
: **enable** – 1 = Data enabled; 0 = Data disabled.

Returns
: Returns *common entity* return values

aErr **setDataHS1Enabled**(const uint8_t enable)

Enables or disables the Hight Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

Parameters
: **enable** – 1 = Enable data; 0 = Disable data.

Returns
: Returns *common entity* return values

aErr **getDataHS2Enabled**(uint8_t *enable)

Gets the current enable value of the High Speed B side (HSB) data lines.: Sub-component of getDataHSEnabled.

Parameters
: **enable** – 1 = Data enabled; 0 = Data disabled.

Returns
: Returns *common entity* return values

aErr **setDataHS2Enabled**(const uint8_t enable)

Enables or disables the Hight Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

Parameters
: **enable** – 1 = Enable data; 0 = Disable data.

Returns
: Returns *common entity* return values

aErr **getDataSSEnabled**(uint8_t *enable)

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled.

Parameters
: **enable** – 1 = Data enabled; 0 = Data disabled.

Returns
: Returns *common entity* return values

aErr **setDataSSEnabled**(const uint8_t enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Parameters
: **enable** – 1 = Enable data; 0 = Disable data.

Returns
: Returns *common entity* return values

aErr **getDataSS1Enabled**(uint8_t *enable)

Gets the current enable value of the Super Speed A side (SSA) data lines.: Sub-component of getDataSSEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSS1Enabled**(const uint8_t enable)

Enables or disables the Super Speed (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 he 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 "Devices" reference for details.
Returns
Returns *common entity* return values

aErr **getMode** (uint32_t *mode)

Gets current mode of the port

Parameters
`mode` – Bit mapped value representing the ports mode. See "Devices" reference for details.

Returns
Returns *common entity* return values

aErr **setMode** (const uint32_t mode)

Sets the mode of the port

Parameters
`mode` – Port mode to be set. See "Devices" documentation for details.

Returns
Returns *common entity* return values

aErr **getErrors** (uint32_t *errors)

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

Parameters
`errors` – Bit mapped field representing the current errors of the ports

Returns
Returns *common entity* return values

aErr **getCurrentLimit** (uint32_t *limit)

Gets the current limit of the port.

Parameters
`limit` – Variable to be filled with the limit in microAmps (uA).

Returns
Returns *common entity* return values

aErr **setCurrentLimit** (const uint32_t limit)

Sets the current limit of the port.

Parameters
`limit` – Current limit to be applied in microAmps (uA).

Returns
Returns *common entity* return values

aErr **getCurrentLimitMode** (uint8_t *mode)

Gets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters
`mode` – Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **setCurrentLimitMode** (const uint8_t mode)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

Parameters
`mode` – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **getAvailablePower**(uint32_t *power)

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Parameters

power – Variable to be filled with the available power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getAllocatedPower**(int32_t *power)

Gets the currently allocated power This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Parameters

power – Variable to be filled with the allocated power in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPowerLimit**(uint32_t *limit)

Gets the user defined power limit for the port.

Parameters

limit – Variable to be filled with the power limit in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **setPowerLimit**(const uint32_t limit)

Sets a user defined power limit for the port.

Parameters

limit – Power limit to be applied in milli-watts (mW).

Returns

Returns *common entity* return values

aErr **getPowerLimitMode**(uint8_t *mode)

Gets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

mode – Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setPowerLimitMode**(const uint8_t mode)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

Parameters

mode – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getName**(uint8_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setName** (uint8_t *buffer, const size_t bufferLength)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getCCCurrentLimit** (uint8_t *value)

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations.

Parameters

value – Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A

Returns

Returns *common entity* return values

aErr **setCCCurrentLimit** (const uint8_t value)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations.

Parameters

value – Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A

Returns

Returns *common entity* return values

aErr **getDataHSRoutingBehavior** (uint8_t *mode)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataHSRoutingBehavior** (const uint8_t mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getDataSSRoutingBehavior** (uint8_t *mode)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters

mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **setDataSSRoutingBehavior** (const uint8_t mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters
mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **getVbusAccumulatedPower** (int32_t *milliwatthours)

Gets the Vbus Accumulated Power

Parameters
milliwatthours – The accumuled 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

aErr **getCC1State**(uint16_t *value)
Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Parameters
value – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:
- None = 0 = portCC1State_None
- Invalid = 1 = portCC1State_Invalid
- Rp (default) = 2 = portCC1State_RpDefault
- Rp (1.5A) = 3 = portCC1State_Rp1p5
- Rp (3A) = 4 = portCC1State_Rp3p0
- Rd = 5 = portCC1State_Rd
- Ra = 6 = portCC1State_Ra
- Managed by controller = 7 = portCC1State_Managed
- Unknown = 8 = portCC1State_Unknown

Returns
Returns *common entity* return values

aErr **getCC2State**(uint16_t *value)
Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Parameters
value – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:
- None = 0 = portCC2State_None
- Invalid = 1 = portCC2State_Invalid
- Rp (default) = 2 = portCC2State_RpDefault
- Rp (1.5A) = 3 = portCC2State_Rp1p5
- Rp (3A) = 4 = portCC2State_Rp3p0
- Rd = 5 = portCC2State_Rd
- Ra = 6 = portCC2State_Ra
- Managed by controller = 7 = portCC2State_Managed
- Unknown = 8 = portCC2State_Unknown

Returns
Returns *common entity* return values

3.3.15 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 bufferLength, size_t *unloadedLength)

Gets all Power Data Objects (PDOs). Equivalent to calling *PowerDeliveryClass::getPowerDataObject()* on all partners, power roles, and index's.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled The order of which is:
  - Rules 1-7 Local Source
  - Rules 1-7 Local Sink
  - Rules 1-7 Partner Source
  - Rules 1-7 Partner Sink.
- **bufferLength** – Length of the buffer to be 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 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 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

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 filed 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.

3.3.16 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. It has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

Public Functions

RailClass(void)

Constructor.

~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 **`setEnable`** (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 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 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 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 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 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 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

3.3.17 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 **setEnable** (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, “getPostion” 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

3.3.18 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 **setEnable** (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 **getEnable** (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

3.3.19 Signal Class

See the *Signal Entity* for generic information.

class **SignalClass** : public Acroname::BrainStem::*EntityClass*

SignalClass: Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Public Functions

SignalClass (void)

Constructor.

~SignalClass (void)

Destructor.

void **init** (*Module* *pModule, const uint8_t index)

Initialize the class.

Parameters

- **pModule** – The module.
- **index** – The index.

aErr **setEnable** (const uint8_t enable)

Enable/Disable the signal output.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr **getEnable** (uint8_t *enable)

Get the Enable/Disable of the signal.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr **setInvert** (const uint8_t invert)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert – 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

3.3.20 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 *buffer, const uint16_t bufferLength)

Load the slot.

Parameters

- **slot** – The slot number.
- **buffer** – The data.
- **bufferLength** – The data length.

Returns

Returns *common entity* return values

aErr **unloadSlot** (const uint8_t slot, const size_t bufferLength, uint8_t *buffer, size_t *unloadedLength)

Unload the slot data.

Parameters

- **buffer** – Byte array that the unloaded data will be placed into.
- **bufferLength** – - 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 filed with the locked state.

Returns
Returns *common entity* return values

aErr **setSlotLocked**(const uint8_t slot, const uint8_t lock)

Sets the locked state of the slot Allows for write protection on a slot.

Parameters
- **slot** – The slot number
- **lock** – state to be set.

Returns
Returns *common entity* return values

3.3.21 System Class

class **SystemClass** : public Acroname::BrainStem::*EntityClass*

SystemClass: The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc.

Public Functions

SystemClass(void)
: Constructor.

virtual **~SystemClass**(void)
: Destructor.

void **init**(*Module* *pModule, const uint8_t index)
: Initialize the 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 **setLEDMaxBrightness** (const uint8_t brightness)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Parameters
brightness – Brightness value relative to 255

Returns
Returns *common entity* return values

aErr **getLEDMaxBrightness** (uint8_t *brightness)

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum).

Parameters
brightness – Brightness value relative to 255

Returns
Returns *common entity* return values

aErr **setBootSlot** (const uint8_t slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

Parameters
slot – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

Returns
Returns *common entity* return values

aErr **getBootSlot**(uint8_t *slot)

Get the store slot which is mapped when the module boots.

Parameters

slot – The slot number in aSTORE_INTERNAL that is mapped after the module boots.

Returns

Returns *common entity* return values

aErr **getVersion**(uint32_t *build)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

Parameters

build – The build version date code.

aErr **getBuild**(uint32_t *build)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Parameters

build – Variable to be filled with build.

aErr **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. aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

Returns

Returns *common entity* return values

aErr **`logEvents`** (void)

Saves system log events to a slot defined by the module (usually ram slot 0).

Returns

Returns *common entity* return values

aErr **`getUptime`** (uint32_t *uptimeCounter)

Get the module's accumulated uptime in minutes

Parameters

`uptimeCounter` – The module's accumulated uptime in minutes.

Returns

Returns *common entity* return values

aErr **`getTemperature`** (int32_t *temperature)

Get the module's current temperature in micro-C

Parameters

`temperature` – The module's system temperature in micro-C

Returns

Returns *common entity* return values

aErr **`getMinimumTemperature`** (int32_t *minTemperature)

Get the module's minimum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

Parameters

`minTemperature` – The module's minimum system temperature in micro-C

Returns

Returns *common entity* return values

aErr **`getMaximumTemperature`** (int32_t *maxTemperature)

Get the module's maximum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

Parameters

`maxTemperature` – The module's maximum system temperature in micro-C

Returns

Returns *common entity* return values

aErr **`getInputVoltage`** (uint32_t *inputVoltage)

Get the module's input voltage.

Parameters

`inputVoltage` – The module's input voltage reported in microvolts.

Returns

Returns *common entity* return values

aErr **`getInputCurrent`** (uint32_t *inputCurrent)

Get the module's input current.

Parameters

`inputCurrent` – The module's input current reported in microamps.

Returns

Returns *common entity* return values

aErr **getModuleHardwareOffset** (uint8_t *offset)

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

Parameters

offset – The module address offset.

Returns

Returns *common entity* return values

aErr **setModuleSoftwareOffset** (const uint8_t address)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

aErr **getModuleSoftwareOffset** (uint8_t *address)

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

aErr **getRouterAddressSetting** (uint8_t *address)

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

Parameters

address – The address for the module. Value must be even from 0-254.

Returns

Returns *common entity* return values

aErr **routeToMe** (const uint8_t 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 bufferLength, 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
- **bufferLength** – Length of the buffer to be filed
- **unloadLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setInputPowerBehaviorConfig** (uint32_t *buffer, const size_t bufferLength)

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getName** (uint8_t *buffer, const size_t bufferLength, size_t *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
- **bufferLength** – Length of the buffer to be filed
- **unloadLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setName** (uint8_t *buffer, const size_t bufferLength)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **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

3.3.22 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.

3.3.23 Timer Class

class **TimerClass** : public Acroname::BrainStem::*EntityClass*

TimerClass: The Timer Class provides access to a simple scheduler. The timer can set to fire only once, or to repeat at a certain interval. Additionally, a timer entity can execute custom Reflex routines upon firing.

Public Functions

TimerClass(void)

Constructor.

~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.

Return values

aErrNone – Action completed successfully.

Returns

Returns *common entity* return values

3.3.24 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 **setEnable**(const uint8_t bEnabled)

Enable the UART channel.

Parameters

bEnabled – true: enabled, false: disabled.

Returns
Returns *common entity* return values

aErr **getEnable**(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

3.3.25 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, SuperSpeed (3.0) only.

Parameters
channel – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setPowerEnable**(const uint8_t channel)

Enable only the power line for a port without changing the state of the data lines.

Parameters
channel – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setPowerDisable**(const uint8_t channel)

Disable only the power line for a port without changing the state of the data lines.

Parameters
channel – The USB sub channel.

Returns
Returns *common entity* return values

aErr **getPortCurrent**(const uint8_t channel, int32_t *microamps)

Get the current through the power line for a port.

Parameters
- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns
Returns *common entity* return values

aErr **getPortVoltage**(const uint8_t channel, int32_t *microvolts)

Get the voltage on the power line for a port.

Parameters
- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in microvolts (1 == 1e-6V).

Returns
Returns *common entity* return values

aErr **getHubMode** (uint32_t *mode)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

Parameters
mode – The USB hub mode.

Returns
Returns *common entity* return values

aErr **setHubMode** (const uint32_t mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

Parameters
mode – The USB hub mode.

Returns
Returns *common entity* return values

aErr **clearPortErrorStatus** (const uint8_t channel)

Clear the error status for the given port.

Parameters
channel – The port to clear error status for.

Returns
Returns *common entity* return values

aErr **getUpstreamMode** (uint8_t *mode)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

Parameters
mode – The Upstream port mode.

Returns
Returns *common entity* return values

aErr **setUpstreamMode** (const uint8_t mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0, usbUpstreamModePort1, and usbUpstreamModeNone.

Parameters
mode – The Upstream port mode.

Returns
Returns *common entity* return values

aErr **getUpstreamState** (uint8_t *state)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

Parameters
state – The Upstream port state.

Returns
Returns *common entity* return values

aErr **setEnumerationDelay** (const uint32_t ms_delay)

Set the inter-port enumeration delay in milliseconds.

Parameters
ms_delay – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns
Returns *common entity* return values

aErr **getEnumerationDelay** (uint32_t *ms_delay)

Get the inter-port enumeration delay in milliseconds.

Parameters

ms_delay – Millisecond delay in 100mS increments (100, 200, 300 etc.)

Returns

Returns *common entity* return values

aErr **setPortCurrentLimit** (const uint8_t channel, const uint32_t microamps)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a stem.system.save() call.

Parameters

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

Returns

Returns *common entity* return values

aErr **getPortCurrentLimit** (const uint8_t channel, uint32_t *microamps)

Get the current limit for the port.

Parameters

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

Returns

Returns *common entity* return values

aErr **setPortMode** (const uint8_t channel, const uint32_t mode)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode 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** – State to be set
  - Disabled: 0
  - Enabled: 1

Returns
Returns *common entity* return values

aErr **getCC1Enable** (const uint8_t channel, uint8_t *pEnable)

Get Enable/Disable on the CC1 line.

Parameters
- **channel** – USB channel.
- **pEnable** – State to be filled
  - Disabled: 0
  - Enabled: 1

Returns
Returns *common entity* return values

aErr **setCC2Enable** (const uint8_t channel, const uint8_t bEnable)

Set Enable/Disable on the CC2 line.

Parameters
- **channel** – USB channel.
- **bEnable** – State to be filled
  - Disabled: 0

- Enabled: 1

Returns
Returns *common entity* return values

aErr **getCC2Enable**(const uint8_t channel, uint8_t *pEnable)

Get Enable/Disable on the CC1 line.

Parameters
- **channel** – - USB channel.
- **pEnable** – - State to be filled
  - Disabled: 0
  - Enabled: 1

Returns
Returns *common entity* return values

aErr **getCC1Current**(const uint8_t channel, int32_t *microamps)

Get the current through the CC1 for a port.

Parameters
- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns
Returns *common entity* return values

aErr **getCC2Current**(const uint8_t channel, int32_t *microamps)

Get the current through the CC2 for a port.

Parameters
- **channel** – The USB sub channel.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

Returns
Returns *common entity* return values

aErr **getCC1Voltage**(const uint8_t channel, int32_t *microvolts)

Get the voltage of CC1 for a port.

Parameters
- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns
Returns *common entity* return values

aErr **getCC2Voltage**(const uint8_t channel, int32_t *microvolts)

Get the voltage of CC2 for a port.

Parameters
- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns
Returns *common entity* return values

aErr **setSBUEnable**(const uint8_t channel, const uint8_t bEnable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

Parameters
- **channel** – The USB sub channel.
- **bEnable** – The state to be set
  - Disabled: 0
  - Enabled: 1

Returns
Returns *common entity* return values

aErr **getSBUEnable**(const uint8_t channel, uint8_t *pEnable)

Get the Enable/Disable status of the SBU

Parameters

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of the SBU

Returns

Returns *common entity* return values

aErr **setCableFlip**(const uint8_t channel, const uint8_t bEnable)

Set Cable flip. This will flip SBU, CC and SS data lines.

Parameters

- **channel** – The USB sub channel.
- **bEnable** – The state to be set The state to be set
  - Disabled: 0
  - Enabled: 1

Returns

Returns *common entity* return values

aErr **getCableFlip**(const uint8_t channel, uint8_t *pEnable)

Get Cable flip setting.

Parameters

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of cable flip.

Returns

Returns *common entity* return values

aErr **setAltModeConfig**(const uint8_t channel, const uint32_t configuration)

Set USB Alt Mode Configuration.

Parameters

- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

Returns

Returns *common entity* return values

aErr **getAltModeConfig**(const uint8_t channel, uint32_t *configuration)

Get USB Alt Mode Configuration.

Parameters

- **channel** – The USB sub channel
- **configuration** – The USB configuration for the given channel.

Returns

Returns *common entity* return values

aErr **getSBU1Voltage**(const uint8_t channel, int32_t *microvolts)

Get the voltage of SBU1 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns

Returns *common entity* return values

aErr **getSBU2Voltage**(const uint8_t channel, int32_t *microvolts)

Get the voltage of SBU2 for a port.

Parameters

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

Returns
Returns *common entity* return values

3.3.26 USBSystem Class

class **USBSystemClass** : public Acroname::BrainStem::*EntityClass*

USBSystem Class: The USBSystem class provides high level control of the lower level Port Class.

Subclassed by *aMTMIOSerial::HubClass*, *aUSBHub2x4::HubClass*, *aUSBHub3c::HubClass*, *aUSBHub3p::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 bufferLength, size_t *unloadedLength)

Gets the current mode of all ports with a single call. Equivalent to calling *PortClass*:getMode() on each port.

Parameters
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

Returns
Returns *common entity* return values

aErr **setModeList** (uint32_t *buffer, const size_t bufferLength)

Sets the mode of all ports with a single call. Equivalent to calling *PortClass::setMode()* on each port

Parameters
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns
Returns *common entity* return values

aErr **getStateList** (uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the state for all ports with a single call. Equivalent to calling *PortClass::getState()* on each port.

Parameters
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

Returns
Returns *common entity* return values

aErr **getPowerBehavior**(uint8_t *behavior)

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system. i.e. What happens when requested power greater than available power.

Parameters

behavior – Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setPowerBehavior**(const uint8_t behavior)

Sets the behavior of how available power is managed. i.e. What happens when requested power is greater than available power.

Parameters

behavior – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **getPowerBehaviorConfig**(uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

Returns

Returns *common entity* return values

aErr **setPowerBehaviorConfig**(uint32_t *buffer, const size_t bufferLength)

Sets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

Parameters

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns

Returns *common entity* return values

aErr **getDataRoleBehavior**(uint8_t *behavior)

Gets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters

behavior – Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns

Returns *common entity* return values

aErr **setDataRoleBehavior**(const uint8_t behavior)

Sets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

Parameters

behavior – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **getDataRoleBehaviorConfig**(uint32_t *buffer, const size_t bufferLength, size_t *unloadedLength)

Gets the current data role behavior configuration Certain data role behaviors use a list of ports to determine priority host priority.

Parameters
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

Returns
Returns *common entity* return values

aErr **setDataRoleBehaviorConfig**(uint32_t *buffer, const size_t bufferLength)

Sets the current data role behavior configuration Certain data role behaviors use a list of ports to determine host priority.

Parameters
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufferLength** – Length of the buffer to be transferred.

Returns
Returns *common entity* return values

aErr **getSelectorMode**(uint8_t *mode)

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters
mode – Variable to be filled with the selector mode

Returns
Returns *common entity* return values

aErr **setSelectorMode**(const uint8_t mode)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Parameters
mode – Mode to be set.

Returns
Returns *common entity* return values

aErr **resetEntityToFactoryDefaults**(void)

Resets the *USBSystemClass* Entity to it factory default configuration.

aErr **getUpstreamHS**(uint8_t *port)

Gets the USB HighSpeed upstream port.

Parameters
port – The current upstream port.

Returns
Returns *common entity* return values

aErr **setUpstreamHS**(const uint8_t port)

Sets the USB HighSpeed upstream port.

Parameters
port – The upstream port to set.

Returns
Returns *common entity* return values

aErr **getUpstreamSS**(uint8_t *port)

Gets the USB SuperSpeed upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstreamSS**(const uint8_t port)

Sets the USB SuperSpeed upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getOverride**(uint32_t *overrides)

Gets the current enabled overrides

Parameters

overrides – Bit mapped representation of the current override configuration.

Returns

Returns *common entity* return values

aErr **setOverride**(const uint32_t overrides)

Sets the current enabled overrides

Parameters

overrides – Overrides to be set in a bit mapped representation.

Returns

Returns *common entity* return values

aErr **setDataHSMaxDatarate**(const uint32_t datarate)

Sets the USB HighSpeed Max datarate

Parameters

datarate – Maximum datarate for the USB HighSpeed signals.

Returns

Returns *common entity* return values

aErr **getDataHSMaxDatarate**(uint32_t *datarate)

Gets the USB HighSpeed Max datarate

Parameters

datarate – Current maximum datarate for the USB HighSpeed signals.

Returns

Returns *common entity* return values

aErr **setDataSSMaxDatarate**(const uint32_t datarate)

Sets the USB SuperSpeed Max datarate

Parameters

datarate – Maximum datarate for the USB SuperSpeed signals.

Returns

Returns *common entity* return values

aErr **getDataSSMaxDatarate**(uint32_t *datarate)

Gets the USB SuperSpeed Max datarate

Parameters

datarate – Current maximum datarate for the USB SuperSpeed signals.

Returns

Returns *common entity* return values

3.4 C API Reference

3.4.1 aDefs.h

group **aDefs**

Acroname Specific Universal Defines and includes.

The C-Interface requires some specific defines for cross platform compatibility. The *aDefs.h* file contains those defines and includes that are necessary at a global level across platforms.

Things like a cross platform way to specify line endings, safe c-string copy and concatenation operations, and boolean typedefs when they are not defined by default.

We rely on the following std headers:

- assert.h
- stddef.h
- stdint.h
- stdbool.h
- string.h
- stdio.h
- stdlib.h

aSHOWERR(msg, error) (printf("Error in File; %s on line; %d, %s: %d\n", __FILE__, __LINE__, msg, error))

A macro that will emit an error on stdout including file and line number when compiled without NDEBUG flag. When NDEBUG is defined it emits nothing.

OS_NEW_LN "\n"

A macro containing the appropriate line ending characters for a given platform \r\n on windows and \n elsewhere.

aStringCopySafe(d, l, s) strncpy((d), (s), (l))

A macro that maps to platform specific safe string copy. Parameters are;.

- d: destination
- l: length
- s: source

aStringCatSafe(d, l, s) strncat((d), (s), (l))

A macro that maps to platform specific safe string concatination. 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 acroname specific semantic define for a pointer to a chunk of memory.

const char ***aDefs_GetModelName**(const int modelNum)

Returns a printable model string.

3.4.2 aDiscovery.h

group **aDiscovery**

Link Discovery Interface.

aDiscovery.h provides an interface for locating BrainStem modules accross multiple transports. It provides a way to find all modules for a give transport as well as specific modules by serial number, or first found.

enum **linkType**

Enum *linkType*.

The linkType enum specifies the connection transport type.

Values:

enumerator **INVALID**

- Undefined link type.

enumerator **USB**

- USB link type.

enumerator **TCPIP**

- TCPIP link type.

enumerator **SERIAL**

- Serial link type.

enumerator **AETHER**

- aEther 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:

- **buadrate** - *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 typdef 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 *vpCBRef, const uint32_t networkInterface)

Function *aDiscovery_EnumerateModules*.

Enumerates the discoverable modules for the given link type. Takes a *aDiscoveryModuleFoundProc* which will determine when to stop the enumeration.

Parameters

- **type** – The trasport type on which search for devices. Valid *linkType* "linktypes" are accepted
- **cbFound** – The *aDiscoveryModuleFoundProc* to call for each module found.
- **vpCBRef** – The vpRef passed into the callback.
- **networkInterface** – Defines the network interface to use when multiple are present. A value of 0 or LOCALHOST_IP_ADDRESS will result in local searches only. Values other than this will have firewall implications.

Returns

Returns the number of modules found.

linkSpec ***aDiscovery_FindModule** (const *linkType* type, const uint32_t serialNum, const uint32_t networkInterface)

Function *aDiscovery_FindModule*.

Finds the module with the given serial number on the given transport type.

Parameters

- **type** – The trasport type on which search for devices. Valid *linkType* "linktypes" are accepted
- **serialNum** – The serial number of the Module to find.
- **networkInterface** – Defines the network interface to use when multiple are present. A value of 0 or LOCALHOST_IP_ADDRESS will result in local searches only. Values other than this will have firewall implications.

Returns

A pointer to the *linkSpec* for the requested module if found or NULL otherwise. This call Allocates memory that must be freed by a call to *aLinkSpec_Destroy*.

linkSpec ***aDiscovery_FindFirstModule** (const *linkType* type, const uint32_t networkInterface)

Function *aDiscovery_FindFirstModule*.

Finds the first module found on the given transport.

Parameters

- **type** – The transport type on which search for devices. Valid *linkType* “linktypes” are accepted
- **networkInterface** – Defines the network interface to use when multiple are present. A value of 0 or LOCALHOST_IP_ADDRESS will result in local searches only. Values other than this will have firewall implications.

Returns

A pointer to the *linkSpec* for the requested module if found or NULL otherwise. This call Allocates memory that must be freed by a call to *aLinkSpec_Destroy*.

LinkSpec Functions

linkSpec ***aLinkSpec_Create**(const *linkType* type)

Function *aLinkSpec_Create*.

Creates a *linkSpec* object with transport set to the given type.

Parameters

type – The transport type on which search for devices. Valid *linkType* “linktypes” are accepted

Returns

A pointer to the *linkSpec* for the requested module or NULL if there was an error allocating memory. This call Allocates memory that must be freed by a call to *aLinkSpec_Destroy*.

aErr **aLinkSpec_Destroy**(*linkSpec* **spec)

Function *aLinkSpec_Destroy*.

Destroys and clears the referenced *linkSpec*.

Parameters

spec – A pointer to the *linkSpec* pointer previously allocated.

Returns

aErrNone on success or an error if there was an error encountered deallocating the *linkSpec*.

3.4.3 Error Codes

enum **aErr**

The *aErr* enum lists the possible error codes for library calls. BrainStem commands generally return a set of unified Error codes. The API tries to be consistent and return these errors from every interaction with the stem.

Values:

enumerator **aErrNone**

0 - Success, no error.

enumerator **aErrMemory**

1 - Memory allocation.

enumerator **aErrParam**

2 - Invalid parameter.

enumerator **aErrNotFound**

3 - Not found.

enumerator **aErrFileNameLength**

4 - File name too long.

enumerator **aErrBusy**

5 - Resource busy.

enumerator **aErrIO**

6 - Input/Output error.

enumerator **aErrMode**

7 - Invalid Mode.

enumerator **aErrWrite**

8 - Write error.

enumerator **aErrRead**

9 - Read error.

enumerator **aErrEOF**

10 - End of file.

enumerator **aErrNotReady**

11 - Not ready, no bytes available.

enumerator **aErrPermission**

12 - Insufficient permissions.

enumerator **aErrRange**

13 - Value out of range.

enumerator **aErrSize**

14 - Invalid Size.

enumerator **aErrOverrun**

15 - Buffer/queue overrun.

enumerator **aErrParse**

16 - Parse error.

enumerator **`aErrConfiguration`**

17 - Configuration error.

enumerator **`aErrTimeout`**

18 - Timeout occurred.

enumerator **`aErrInitialization`**

19 - Initialization error.

enumerator **`aErrVersion`**

20 - Invalid version.

enumerator **`aErrUnimplemented`**

21 - Functionality unimplemented.

enumerator **`aErrDuplicate`**

22 - Duplicate request.

enumerator **`aErrCancel`**

23 - Cancelation 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.

3.4.4 aFile.h

group **aFile**

Platform Independent File Access Interface.

aFile.h provides a platform independent interface for opening, reading, and writing files.

typedef void ***aFileRef**

Typedef *aFileRef* Opaque reference to a file handle.

enum **aFileMode**

Enum *aFileMode*.

Represents whether the file is to be opened in read or write mode.

Values:

enumerator **aFileModeReadOnly**

File read mode.

enumerator **aFileModeWriteOnly**

File write mode.

enumerator **aFileModeAppend**

File write mode from end of current file.

enumerator **aFileModeUnknown**

File in unknown mode.

enum **aFileSeekMode**

Enum *aFileSeekMode*.

Represents the seek start location.

Values:

enumerator **aSeekStart**

Perform a seek from the beginning of the file.

enumerator **aSeekCurrent**

Perform a seek from the current location.

enumerator **aSeekEnd**

Perform a seek from the end of the file.

bool **aFile_Exists**(const char *pFilename)

Does the File Exist.

Checks for the existence of a file at filename.

Parameters

pFilename – path to file.

Returns

bool True if file exists, false otherwise.

aFileRef **aFile_Open**(const char *pFilename, const *aFileMode* eMode)

Open a File.

Opens the file Given in pFilename with the given fileMode eMode.

Parameters

- **pFilename** – path to file.
- **eMode** – Open the file for Reading or Writing.

Returns

aFileRef on success or NULL on failure.

aErr **aFile_Close**(*aFileRef* *fileRef)

Close an open file.

Close an open file. The fileRef is set to NULL on success.

Parameters

fileRef – Pointer to the handle to the open file.

Return values

- **aErrNone** – Success.
- **aErrParam** – invalid file reference.

Returns

Function returns aErr values.

aErr **aFile_Read**(*aFileRef* fileRef, uint8_t *pBuffer, const size_t nLength, size_t *pActuallyRead)

Read from an open file.

Read from an open file.

Parameters

- **fileRef** – The handle to the open file.
- **pBuffer** – The data buffer to read into.

- **nLength** – The length of the read buffer.
- **pActuallyRead** – The Number of bytes actually read from the file.

Return values

- **aErrNone** – Success.
- **aErrMode** – The file is not readable.
- **aErrIO** – An error occured 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 occured 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 occured 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 occured 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 priviledges.
- **aErrNotFound** – if the file cannot be located.

Returns

Function returns aErr values.

3.4.5 aLink.h

group **aLink**

BrainStem Link Interface.

aLink.h provides the interface for creating and maintaining the link to a BrainStem module, and the BrainStem network. It includes facilities for starting and stopping links, as well as sending and receiving BrainStem protocol packets.

typedef uint32_t **aLinkRef**

Typedef *aLinkRef* Opaque reference to a BrainStem link.

Typedef for aLinkRef for an opaque reference to BrainStem Link.

enum **linkStatus**

Represents the current state of the BrainStem link.

Values:

enumerator **STOPPED**

Link currently stopped.

enumerator **INITIALIZING**

Starting communication with module.

enumerator **RUNNING**

Link running.

enumerator **STOPPING**

Link is in the process of stopping.

enumerator **SYNCING**

Packet framing lost re-syncing.

enumerator **INVALID_LINK_STREAM**

Link stream provided is not valid.

enumerator **IO_ERROR**

Communication error occured on link, could not resync.

enumerator **RESETTING**

Resetting the link connection

enumerator **UNKNOWN_ERROR**

Something really bad happened, but we couldn't determin what.

aLinkRef **aLink_CreateUSB** (const uint32_t serialNumber)

Create a USB BrainStem link reference.

Creates a reference to a USB BrainStem link. The linkStream is now maintained by the BrainStem link. If the link already exists, the use count for that link will be incremented and the linkRef for that entry will be returned.

Links created with this procedure must use aLink_Destroy to properly dispose of the link reference and associated connections.

Parameters

serialNumber – Unique identifier of the device.

Returns

aLinkRef identifier if successful or 0 otherwise.

aLinkRef **aLink_CreateTCPIP** (const uint32_t 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 succesful completion of the Destroy call.

Returns
aStreamRef Return value will always be NULL. The return value has been left for backwards compatability.

aErr **aLink_Reset** (const *aLinkRef* linkRef)

Reset a connection to a BrainStem module.

Stop the active connection to the BrainStem if the Link contains a valid stream Reference, and clear out the communication buffers, and restart the link.

Parameters
linkRef – A valid LinkRef.

Return values

- **aErrNone** – the call completed successfully, a subsequent call to aLink_GetStatus should return the current state of the link.
- **aErrParam** – No valid LinkRef provided.

Returns
Function returns aErr values.

linkStatus **aLink_GetStatus** (const *aLinkRef* linkRef)

Return the current status of the BrainStem link.

Return the current status of the BrainStem link.

Parameters
linkRef – A valid LinkRef.

Returns
linkStatus See the possible linkStatus values.

aPacket ***aLink_GetPacket** (const *aLinkRef* linkRef)

Return the first packet in the Link incomming FIFO.

Return the first packet in the Link incomming 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 incomming FIFO.

Return the first packet in the Link incomming 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 incomming FIFO.

Return the first packet matched by proc in the Link incomming FIFO. This call is non blocking and returns immediatly.

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 incomming FIFO.

Return the first packet matched by proc in the Link incomming 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 incomming FIFO.

Drain all matching packets from the incomming FIFO. This call does not block.

Parameters

- **linkRef** – A valid LinkRef.
- **proc** – The callback used for determining a matching packet.
- **vpRef** – A resource passed to the callback proc.

Returns

aPacket Returns the first packet that is matched by proc or NULL.

aErr **aLink_PutPacket**(const *aLinkRef* linkRef, const *aPacket* *packet)

Put a packet into the outgoingBackend link FIFO.

Put a packet into the outgoingBackend link FIFO.

Parameters

- **linkRef** – A valid LinkRef.
- **packet** – A BrainStem packet.

Return values

- **aErrNone** – Call successfully added the packet.
- **aErrParam** – Invalid LinkRef or packet.
- **aErrResource** – Unable to create memory for packet in FIFO.

Returns

Function returns aErr values.

3.4.6 aMutex.h

group **aMutex**

Platform Independent Synchronization Primitive.

aMutex.h Provides a platform independent synchronization mechanism. The link interface and the packe 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 occured.

Return values

- **aErrNone** – - Successfully aquired the lock.
- **aErrParam** – - If the MutexRef was invalid.

Returns

Function returns aErr values.

Returns

aErrDuplicate - If a specific error occured locking the mutex.

aErr **aMutex_TryLock** (*aMutexRef* mutex)

Mutex TryLock.

Non Blocking attempt to Lock the mutex. The call will return immediately with aErrBusy if another process or thread owns the lock.

Return values

- **aErrNone** – - Successfully aquired the lock.
- **aErrParam** – - If the MutexRef was invalid.
- **aErrBusy** – - If the lock was already in use.

Returns

Function returns aErr values.

aErr **aMutex_Unlock** (*aMutexRef* mutex)

Mutex Unlock.

Relenquish the lock on the mutex.

Return values

- **aErrNone** – - Successfully unlocked mutex.
- **aErrParam** – - If the MutexRef was invalid.
- **aErrPermission** – - If the lock is owned by another thread.

Returns

Function returns aErr values.

3.4.7 aPacket.h

group **aPacket**

BrainStem Packet.

aPacket.h Provides and 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 destoryed.
- **aErrParam** – - The packetRef is invalid.

Returns

Function returns aErr values.

3.4.8 aProtocoldefs.h

group **aProtocoldefs**

BrainStem Protocol Definitions.

aProtocoldefs.h Provides protocol and BrainStem specific defines for entities, communication, and protocol specifics.

aBRAINSTEM_MAXPACKETBYTES 28

8 Bytes - Packet protocol payload maximum.

group **UEI_Defines**

UEI and Command support for C/C++ and Reflex languages.

Defines

ueiSPECIFIER_INDEX_MASK 0x1F

0x1F - Mask bits for Index on index byte.

ueiSPECIFIER_RETURN_MASK 0xE0

0xE0 - Mask bits for Return value on index byte.

ueiSPECIFIER_RETURN_HOST 0x20

1 << 5 - Specifier Bit for UEI response to host.

ueiSPECIFIER_RETURN_I2C 0x40

2 << 5 - Specifier Bit for UEI response to Module over I2C.

ueiSPECIFIER_RETURN_VM 0x60

3 << 5 - Specifier Bit for UEI response to VM on module.

ueiREPLY_ERROR 0x80

1 << 7 - Error flag on response in index byte.

ueiREPLY_STREAM 0x40

1 << 6 - Stream flag on response in index byte.

ueiOPTION_GET 0x40

0x40 - Option byte code for UEI Get request.

ueiOPTION_VAL 0x00

0x00 - Option byte code for UEI Val response.

ueiOPTION_SET 0x80

0x80 - Option byte code for UEI Set request.

ueiOPTION_ACK 0xC0

0xC0 - Option byte code for UEI Ack response.

ueiOPTION_MASK 0x3F

0x3F - Mask for getting command option from option byte.

ueiOPTION_OP_MASK 0xC0

0xC0 - Mask for getting Operation Get/Set/Val/Ack

ueiBYTES_CONTINUE 0x80

ueiBYTES_CONTINUE_MASK 0x7F

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` 30
: **30** - System input power behavior option code

`systemInputPowerBehaviorConfig` 31
: **31** - System input power behavior config option code

`systemName` 32
: **32** - System name option code

`systemPowerLimit` 33
: **33** - System power limit option code

`systemPowerLimitMax` 34
: **34** - System power limit max option code

`systemPowerLimitState` 35
: **35** - System power limit state option code

`systemResetEntityToFactoryDefaults` 36
: **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.

`systemLEDMaxBrightness` 42
: **42** - System LED Brightness option code

systemBuild 43
    43 - Firmware build 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** - Select Mux Mode configuration

muxConfig_default 0

muxConfig_splitMode 1

muxConfig_channelpriority 2

muxSplit 5
: **5** - Configure the data and power signals when in split mode

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 changing functionality

usbPortMode_passive 3

3 - Electrical pasthrough 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.

3.4.9 aStream.h

group **aStream**

Platform Independent Stream Abstraction.

aStream.h provides a platform independent stream abstraction for common I/O streams. Provides facilities for creating and destroying as well as writing and reading from streams.

typedef void ***aStreamRef**

Typedef *aStreamRef* Opaque reference to stream primitive.

group **StreamCallbacks**

Typedefs

typedef *aErr* (***aStreamGetProc**)(uint8_t *pData, void *ref)

Typedef *aStreamGetProc*.

This callback is defined to read one byte from the concrete stream implementation.

Param pData
The data Buffer to fill.

Param ref
Opaque reference to concrete stream implementation.

Retval aErrNone
Successfully read the byte.

Retval aErrNotReady
No bytes in stream to read.

Retval aErrEOF
Reached the end of the stream.

Retval aErrIO
An error encountered reading from stream.

Return
Function returns aErr values.

typedef *aErr* (***aStreamPutProc**)(const uint8_t *pData, void *ref)

Typedef *aStreamPutProc*.

This callback is defined to write one byte to the concrete stream implementation.

Param pData
The data Buffer to write.

Param ref
opaque reference to concrete stream implementation.

Retval aErrNone
Successfully wrote the byte.

Retval aErrIO
An error encountered reading from stream.

Return
Function returns aErr values.

typedef *aErr* (***aStreamDeleteProc**)(void *ref)

Typedef *aStreamDeleteProc*.

This callback is defined to destroy the concrete stream implementation.

Param ref
opaque reference to concrete stream implementation.

Retval aErrNone
Successfully destroyed.

Retval aErrParam
Invalid ref.

Return
Function returns aErr values.

typedef *aErr* (***aStreamWriteProc**)(const uint8_t *pData, const size_t nSize, void *ref)

Typedef *aStreamWriteProc*. (Optional)

Optional multi-byte write for efficiency, not required..

Param pData
The pointer to the data to write to the stream.

Param nSize
The size of the data buffer in bytes.

Param ref
Opaque reference to concrete stream implementation.

Retval aErrNone
Successfully destroyed.

Retval aErrIO
An error encountered reading from stream.

Return
Function returns aErr values.

enum **aBaudRate**

Enum *aBaudRate*.

Accepted serial stream baudrates.

Values:

enumerator **aBAUD_2400**
2400 baud

enumerator **aBAUD_4800**
4800 baud

enumerator **aBAUD_9600**
9600 baud

enumerator **aBAUD_19200**
19,200 baud

enumerator **aBAUD_38400**
38,400 baud

enumerator **aBAUD_57600**
57,600 baud

enumerator **aBAUD_115200**
115,200 baud

enumerator **aBAUD_230400**
230,400 buad

enum **aSerial_Bits**
Enum *aSerial_Bits*.

The accepted number of serial bits per byte.

Values:

enumerator **aBITS_8**
8 bits

enumerator **aBITS_7**
7 bits

enum **aSerial_Stop_bits**
Enum *aSerial_Stop_bits*.

The accepted number of serial stop bits.

Values:

enumerator **aSTOP_BITS_1**
1 stop bit

enumerator **aSTOP_BITS_2**

2 stop bits

aStreamRef **aStream_Create** (*aStreamGetProc* getProc, *aStreamPutProc* putProc, *aStreamWriteProc* writeProc, *aStreamDeleteProc* deleteProc, const void *procRef)

Base Stream creation procedure.

Creates a Stream Reference.

Parameters

- **getProc** – - Callback for reading bytes from the underlying stream.
- **putProc** – - Callback for writing bytes to the underlying stream.
- **writeProc** – - Optional callback for optimized writing of multiple bytes.
- **deleteProc** – - Callback for safe destruction of underlying resource.
- **procRef** – - opaque reference to the underlying resource,

Returns

Function returns aStreamRef on success and NULL on error.

aErr **aStream_CreateFileInput** (const char *pFilename, *aStreamRef* *pStreamRef)

Create a file input stream.

Creates a file input stream.

Parameters

- **pFilename** – - The filename and path of the file to read from.
- **pStreamRef** – - The resulting stream accessor for the input file.

Return values

- **aErrNone** – - Successful creation.
- **aErrNotFound** – - The file to read was not found.
- **aErrIO** – - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateFileOutput** (const char *pFilename, *aStreamRef* *pStreamRef)

Create a file output stream.

Creates a file output stream.

Parameters

- **pFilename** – - The filename and path of the file to write to.
- **pStreamRef** – - The resulting stream accessor for the output file.

Return values

- **aErrNone** – - Successful creation.
- **aErrIO** – - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateSerial** (const char *pPortName, const *aBaudRate* nBaudRate, const bool parity, const *aSerial_Bits* bits, const *aSerial_Stop_bits* stop, *aStreamRef* *pStreamRef)

Create a serial communication stream.

Creates a serial stream.

Parameters

- **pPortName** – - The portname of the serial device.
- **nBaudRate** – - The baudrate to connect to the device at.
- **parity** – - Whether serial parity is enabled.
- **bits** – - The number of bits per serial byte.
- **stop** – - The number of stop bits per byte.
- **pStreamRef** – - The resulting stream accessor for the serial device.

Return values

- **aErrNone** – - Successful creation.
- **aErrConnection** – - The connection was unsuccessful.
- **aErrIO** – - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateSocket** (const uint32_t address, const uint16_t port, *aStreamRef* *pStreamRef)

Create a TCP/IP socket stream.

Creates a TCP/IP socket stream.

Parameters

- **address** – - The IP4 address of the connection.
- **port** – - The TCP port to connect to.
- **pStreamRef** – - The resulting stream accessor for the TCP connection.

Return values

- **aErrNone** – - Successful creation.
- **aErrConnection** – - The connection was unsuccessful.
- **aErrIO** – - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateMemory** (const aMemPtr pMemory, const size_t size, *aStreamRef* *pStreamRef)

Create a stream accessor for a block of memory.

Creates a stream accessor for a block of allocated memory. Reads and Writes like any other stream. The memory stream does not make a copy of the memory and doesn't free it but rather provides a stream layer to access it.

Parameters

- **pMemory** – - a pointer to a block of memory.
- **size** – - The size of the block in bytes.

- **pStreamRef** – - The resulting stream accessor for the memory block.

Return values

- **aErrNone** – - Successful creation.
- **aErrParam** – - The memory block is invalid.
- **aErrIO** – - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStream_CreateUSB** (const uint32_t serialNum, *aStreamRef* *pStreamRef)

Create a stream to a USB device.

Creates a BrainStem link stream to a USB based module.

Parameters

- **serialNum** – - The BrainStem serial number.
- **pStreamRef** – - The resulting stream accessor for the BrainStem module.

Return values

- **aErrNone** – - Successful creation.
- **aErrNotFound** – - The brainstem device was not found.
- **aErrIO** – - A communication error occurred.

Returns

Function returns aErr values.

aErr **aStreamBuffer_Create** (const size_t nIncSize, *aStreamRef* *pBufferStreamRef)

Create a stream buffer.

Creates a stream buffer.

StreamBuffers are typically used to aggregate a bunch of output into a single pile of bytes. This pile can then be checked for size or accessed as a single block of bytes using the aStreamBuffer_Get call. Finally, these bytes can then be read back out of the buffer until it is empty when it will report an error of aErrEOF. While this stream is thread-safe for different threads doing reads and writes, it is not the best candidate for managing a pipe between threads. Use the aStream_CreatePipe in that scenario as it can be filled and emptied over and over which is typically the use case for cross-thread pipes.

Parameters

- **nIncSize** – - The Increment size to expand the buffer by when it becomes full.
- **pBufferStreamRef** – - The buffer stream resulting from the call.

Return values

- **aErrNone** – - The buffer was successfully created.
- **aErrResource** – - The resources were not available to create the buffer.

Returns

Function returns aErr values.

aErr **aStreamBuffer_Get** (*aStreamRef* bufferStreamRef, 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

- **`bufferStreamRef`** – - 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 occurred creating the logging stream.

aErr **aStream_Read** (*aStreamRef* streamRef, uint8_t *pBuffer, const size_t length)

Read a byte array record from a stream.

Read a byte array record from a stream.

Parameters

- **streamRef** – - The reference to the stream to read from.
- **pBuffer** – - byte array buffer to read into.
- **length** – - the length of the read buffer.

Return values

- **aErrNone** – - Successful read.
- **aErrMode** – - The streamRef is not readable.
- **aErrIO** – - An error occurred reading the data.

Returns

Function returns aErr values.

aErr **aStream_Write** (*aStreamRef* streamRef, const uint8_t *pBuffer, const size_t length)

Write a byte array to a Stream.

Write a byte array to a Stream.

Parameters

- **streamRef** – - The reference to the stream to write to.
- **pBuffer** – - byte array to write out to the stream.
- **length** – - the byte array length

Return values

- **aErrNone** – - Successful write.
- **aErrMode** – - The streamRef is not writable.
- **aErrIO** – - An error occurred writing the data.

Returns
Function returns aErr values.

aErr **aStream_ReadRecord** (*aStreamRef* streamRef, uint8_t *pBuffer, size_t *lengthRead, const size_t maxLength, const uint8_t *recordTerminator, const size_t terminatorLength)

Read a byte array record from a stream with a record terminator.

Read a byte array record from a stream with a record terminator.

Parameters

- **streamRef** – - The reference to the stream to read from.
- **pBuffer** – - Byte array buffer to read into.
- **lengthRead** – - The length of the read buffer.
- **maxLength** – - The Maximum record length.
- **recordTerminator** – - The byte array representing the record terminator.
- **terminatorLength** – - The length of the record terminator.

Return values
aErrNone – - Successful read.

Returns
Function returns aErr values.

Returns
aErrMode - The streamRef is not readable.

Returns
aErrIO - An error occurred reading the data.

aErr **aStream_WriteRecord** (*aStreamRef* streamRef, const uint8_t *pBuffer, const size_t bufferLength, const uint8_t *recordTerminator, const size_t terminatorLength)

Write a byte array with a record terminator to a Stream.

Write a byte array with a record terminator to a Stream.

Parameters

- **streamRef** – - The reference to the stream to write to.
- **pBuffer** – - byte array to write out to the stream.
- **bufferLength** – - the byte array length
- **recordTerminator** – - the byte array representing the record terminator
- **terminatorLength** – - the length of the record terminator.

Return values

- **aErrNone** – - Successful write.
- **aErrMode** – - The streamRef is not writable.
- **aErrIO** – - An error occurred writing the data.

Returns
Function returns aErr values.

aErr **aStream_ReadCString** (*aStreamRef* streamRef, char *pBuffer, const size_t maxLength)

Read a null terminated string from Stream.

Read a null terminated string from Stream.

Parameters

- **streamRef** – - The reference to the stream to read from.
- **pBuffer** – - Character array buffer to read into.
- **maxLength** – - The maximum length of the string.

Return values

- **aErrNone** – - Successful read.
- **aErrMode** – - The streamRef is not readable.
- **aErrIO** – - An error occurred reading the data.

Returns
Function returns aErr values.

aErr **aStream_WriteCString** (*aStreamRef* streamRef, const char *pBuffer)

Write a null terminated string.

Write a null terminated string.

Parameters

- **streamRef** – - The reference to the stream to write to.
- **pBuffer** – - character array to write.

Return values

- **aErrNone** – - Successful write.
- **aErrMode** – - The streamRef is not writable.
- **aErrIO** – - An error occurred writing the data.

Returns
Function returns aErr values.

aErr **aStream_ReadCStringRecord** (*aStreamRef* streamRef, char *pBuffer, const size_t maxLength, const char *recordTerminator)

Read a null terminated string with a record terminator to pBuffer.

Read a null terminated string with a record terminator to pBuffer.

Parameters

- **streamRef** – - The reference to the stream to read to.
- **pBuffer** – - character array to read to.
- **maxLength** – - The maximum number of characters to read.
- **recordTerminator** – - The record terminator to read to.

Return values

- **aErrNone** – - Successful read.

- **aErrMode** – - The streamRef is not readable.
- **aErrIO** – - An error occurred reading the data.

Returns
Function returns aErr values.

aErr **aStream_WriteCStringRecord**(*aStreamRef* streamRef, const char *pBuffer, const char *recordTerminator)

Write a null terminated string with a record terminator to the stream.

Write a null terminated string with a record terminator to the stream.

Parameters

- **streamRef** – - The reference to the stream to be written to.
- **pBuffer** – - Null terminated string to write to the stream.
- **recordTerminator** – - The record terminator to write after the contents.

Return values

- **aErrNone** – - Successful write.
- **aErrMode** – - The streamRef is not writable.
- **aErrIO** – - An error occurred writing the data.

Returns
Function returns aErr values.

aErr **aStream_Flush**(*aStreamRef* inStreamRef, *aStreamRef* outStreamRef)

Flush contents of inStream into outStream.

Flush the entire current content of the instream into the outstream.

Parameters

- **inStreamRef** – - The reference to the stream to be flushed into the outstream.
- **outStreamRef** – - The reference to the stream instream is flushed into.

Return values

- **aErrNone** – - Successful Flush.
- **aErrMode** – - The outstream is not writable or instream is not readable.
- **aErrIO** – - An error occurred flushing the data.

Returns
Function returns aErr values.

aErr **aStream_Destroy**(*aStreamRef* *pStreamRef)

Destroy a Stream.

Safely destroy a stream and deallocate the associated resources.

Parameters
pStreamRef – - A pointer to a pointer of a valid streamRef. The StreamRef will be set to NULL on successful destruction of the stream.

Return values

- **aErrNone** – - The stream was successfully destroyed.
- **aErrParam** – - If the streamRef is invalid.

Returns
Function returns aErr values.

3.4.10 aTime.h

group **aTime**

Basic Time procedures Sleep and Get process tics.

aTime.h provides a platform independent interface for millisecond sleep, and for getting process tics.

unsigned long **aTime_GetMSTicks** (void)

Get the current tick count in milliseconds.

This call returns a number of milliseconds. Depending on the platform, this can be the number of milliseconds since the last boot, or from the epoc start. As such, this call should not be used as an external reference clock. It is accurate when used as a differential, i.e. internal, measurement only.

Returns
unsigned long number of milliseconds elapsed.

aErr **aTime_MSSleep** (const unsigned long msTime)

Sleep the current process for msTime milliseconds.

Sleeps the current process. This is not an active sleep, there are no signals which will “wake” the process.

Parameters
msTime – Milliseconds to sleep.

Return values

- **aErrNone** – The call returned successfully.
- **aErrUnknown** – Um unknown what went wrong.

Returns
Function returns aErr values.

3.4.11 aUEI.h

group **aUEI**

UEI Utilities.

aUEI.h Provides structs and utilities for working with UEIs.

enum **dataType**

Typedef Enum *dataType*.

UEI datatype

Values:

enumerator **aUEI_VOID**

Void datatype.

enumerator **aUEI_BYTE**

Char datatype.

enumerator **aUEI_SHORT**

Short datatype.

enumerator **aUEI_INT**

Int datatype.

enumerator **aUEI_BYTES**

Bytes datatype.

struct **uei**

Typedef Struct *uei*.

UEI data struct.

Public Members

uint8_t **module**

Module address.

uint8_t **command**

Command code.

uint8_t **option**

option code & UEI operation.

uint8_t **specifier**

Entity index & response specifier.

uint8_t **byteVal**

Char value union member.

uint16_t **shortVal**

Short value union member.

uint32_t **intVal**

Int value union member.

dataType **type**

Union dataType.

uint16_t **aUEI_RetrieveShort** (const uint8_t *p)

Retrieve a short from a UEI.

Parameters
: **p** – Pointer to byte array containing short.

Returns
: uint16_t The short value.

void **aUEI_StoreShort** (uint8_t *p, uint16_t v)
: Store a short in a UEI.

  Parameters

  - **p** – Pointer to uei shortVal.
  - **v** – Short value to store.

uint32_t **aUEI_RetrieveInt** (const uint8_t *p)
: Retrieve an Int from a UEI.

  Parameters
  : **p** – Pointer to byte array containing the Int.

  Returns
  : uint32_t The integer value.

void **aUEI_StoreInt** (uint8_t *p, uint32_t v)
: Store an Int in a UEI.

  Parameters

  - **p** – Pointer to the IntVal of a UEI.
  - **v** – The value to store.

3.4.12 aVersion.h

group **aVersion**
: Library version interface.

  aVersion.h Provides version information for the BrainStem2 library.

aVERSION_MAJOR 2
: Major revision level of library.

  Major revision bumps will break compatibility with existing versions and may introduce protocol changes or other fundamental differences.

aVERSION_MINOR 11
: Minor revision level of library.

  Minor revisions should largly be compatible, however new features may be added with a minor revision change.

aVERSION_PATCH 1
: 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 formated version number.

Parameters

- **build** – The packed version number returned from the system.getVersion call.
- **string** – The string to fill with the version string.
- **len** – The length of the filled string, not longer than MAX_VERSION_STRING.

bool **aVersion_IsLegacyFormat**(uint32_t build)

Check if the given build version is of the legacy packing format.

Parses the revision format from the given uint32.

Parameters

build – The packed version number returned from the system.getVersion call.

Returns

Whether the revision format is the old packing format.

uint8_t **aVersion_GetMajor** (void)

Return the major revision number.

Returns

The major revision number.

uint8_t **aVersion_GetMinor** (void)

Return the minor revision number.

Returns

The minor revision number.

uint32_t **aVersion_GetPatch** (void)

Return the revision patch number.

Returns

The revision patch number.

const char ***aVersion_GetString** (void)

Return a human readable version string.

Returns

char* human readable version string.

bool **aVersion_IsAtLeast** (const uint8_t major, const uint8_t minor, const uint8_t patch)

Check that the current software version is at least major.minor.patch.

Parameters

- **major** – The major revision level.
- **minor** – The minor revision.
- **patch** – The patch level.

Returns

True when current version is at least what is given, false otherwise

char ***aVersion_GetFeatureList** (void)

Get an array of the features the library supports.

Returns

an array of c strings describing the features the library supports.

void **aVersion_DestroyFeatureList** (char **featureList)

Destroy the feature list.

Parameters

featureList – pointer to featurelist.

3.4.13 PortMapping.h

group **PortMapping**

BrainStem Port Mapping Interface.

PortMapping.h provides an interface for usb descriptor information of devices downstream of Acroname hub products.

enum **PORT_SPEED**

Port speed enumeration

Values:

enumerator **kPORT_SPEED_UNKNOWN**

kPORT_SPEED_UNKNOWN (0)

enumerator **kPORT_SPEED_LOW**

kPORT_SPEED_LOW (1)

enumerator **kPORT_SPEED_FULL**

kPORT_SPEED_FULL (2)

enumerator **kPORT_SPEED_HIGH**

kPORT_SPEED_HIGH (3)

enumerator **kPORT_SPEED_SUPER**

kPORT_SPEED_SUPER (4)

enumerator **kPORT_SPEED_SUPER_PLUS**

kPORT_SPEED_SUPER_PLUS (5)

struct **DeviceNode**

Device Node Structure - Contains information linking the downstream device to the Acroname Hub.

Public Members

uint32_t **hubSerialNumber**

Serial number of the Acroname hub where the device was found.

uint8_t **hubPort**

Port of the Acroname hub where the device was found.

uint16_t **idVendor**

Manufactures Vendor ID of the downstream device.

uint16_t **idProduct**

Manufactures Product ID of the downstream device.

PORT_SPEED_t **speed**

The devices downstream device speed.

char **productName**[255]

USB string descriptor

char **serialNumber**[255]
USB string descriptor

char **manufacturer**[255]
USB string descriptor

aErr **getDownstreamDevices** (DeviceNode_t *buffer, uint32_t bufferLength, uint32_t *devicesFound)
Gets downstream device USB information for all Acroname hubs.

Parameters

- **buffer** – Pointer to the start of a list/array to be used by the function.
- **bufferLength** – Size of the list/array in DeviceNode_t's, not bytes.)
- **devicesFound** – The number of DeviceNode_t's that were populated.

Return values

- **aErrNone** – on success
- **aErrParam** – Passed in values are not valid. (NULL, size etc).
- **aErrMemory** – No more room in the list
- **aErrNotFound** – No Acroname devices were found.

3.5 RESTful API Reference

Welcome to the BrainStem RESTful API reference documentation. This interface provides access to the BrainStem protocol over HTTP as well as additional information about system versions, attached USB devices, and bulk data transfers.

Use of the RESTful API requires either *BrainD*, or one of the applications built on BrainD, such as *ControlRoom*.

Receiving information from the RESTful interface occurs through an HTTP GET request. Sending or setting information occurs through a PUT interface.

The current version of the RESTful interface is `v1`. Changes that do not break a RESTful API will not constitute a version change. Removing features or making significant changes will result in a version change. Deploying breaking changes may be necessary for legal, performance, or security reasons. When a new version is released, the current version will remain supported for one year and may become unavailable anytime after this period.

3.5.1 API Version v1

- `GET http://<IPADDRESS>:<PORT>/api/v1/<ENDPOINT>/<PARAMETERS>`
- `PUT http://<IPADDRESS>:<PORT>/api/v1/<ENDPOINT>/<PARAMETERS> {BODY}`

Table 4: Endpoint Struture Definition

Parameter	Request Type	Description
IPADDRESS	GET, PUT	IP Address of BrainD server hosting RESTful endpoints.
PORT	GET, PUT	Port that BrainD server is listening for connections. Default is `9005`.
ENDPOINT	GET, PUT	RESTful API endpoint to access.
PARAMETERS	GET, PUT	Command parameters for the given ENDPOINT.
BODY	PUT	HTTP `PUT` payload for the given ENDPOINT and PARAMETERS.

The available options for the `ENDPOINT` parameter are listed below:

Table 5: Available Endpoints

Service	Description
acronameDevices	Enumeration of attached Acroname devices
acronameDevicesState	Bulk Endpoint containing telemetry about all attached Acroname devices
brainstem	Raw access to BrainStem API for a device
devices	List of all USB devices connected to the host computer
version	Software Versions

The response body for all RESTful API calls will be encapsulated in a *transaction envelope*. This envelope will provide error information, request context, and other useful fields about the request and the response payload for the specific API call.

JSON Schema Version 1

acronameDevices Endpoint

The `acronameDevices` endpoint enumerates all of the attached Acroname devices on the host system, and provides information about their connection parameters, software licenses, and module information.

There are no input parameters to this endpoint. It will return a JSON array of devices, or error, encapsulated in a *transaction envelope*.

JSON Schema for Response Object

The following tables list a JSON Schema[92] for the response object.

Download the raw JSON Schema file for this response: `acroname_devices_response.json`

Object Fields of Acroname Devices Response

Name	Type	Mandatory	Description
`acronameDevices`	Array<*Object*>	Yes	An array containing all Acroname devices in the system.

Object Fields of `acronameDevices[]`

Name	Type	Mandatory	Description
`type`	*v1-type*	Yes	Type of connection to device
`serialNumber`	*v1-serialNumber*	Yes	Acroname Serial Number
`module`	*v1-uint32*	Yes	Brainstem Module address of this device
`router`	*v1-uint32*	No	BrainStem network router address
`routerSerialNumber`	*v1-serialNumber*	No	BrainStem network router serial number
`model`	*v1-model*	Yes	Model Number of device (see aDefs.c for allowed values)
`entitlements`	Array<*Object*>	No	An array containing all entitlements for this specific device

[92] https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

Object Fields of `acronameDevices[].entitlements[]`

Name	Type	Mandatory	Description
`tag`	*v1-tag*	Yes	The entitlement tag associated with this entitlement
`version`	*v1-uint32*	Yes	Version field of the specified entitlement

v1-type

Name	Value
Type	string
description	Enumeration of all possible USB speeds
Enumerations	`INVALID`, `USB`, `TCPIP`, `SERIAL`, `NETWORK`

v1-uint32

Name	Value
Type	integer
description	Unsigned 32-bit integer
minimum	0
maximum	4294967295

v1-serialNumber

Name	Value
Type	string
description	Representation of an Acroname device serial number
Regular Expression	`^(0x)?[A-Fa-f0-9]{8}$`

v1-tag

Name	Value
Type	string
description	Representation of the entitlement tag

v1-model

Name	Value
Type	string
description	Enumeration of all supported Acroname devices.
Enumerations	`USBStem`, `EtherStem`, `MTMIOSerial`, `MTMPM1`, `MTMEtherStem`, `MTMUSBStem`, `USBHub2x4`, `MTMRelay`, `USBHub3p`, `MTMDAQ1`, `USBCSwitch`, `MTMDAQ2`, `MTMLoad1`, `USBHub3c`, `Unknown`

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevices
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/acronameDevices')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:


```
{
    "timestamp": "2023-09-20T17:11:54.092Z",
    "request": {
        "endpointName": "/api/v1/acronameDevices",
        "parameters": {}
    },
    "response": {
        "acronameDevices": [
            {
                "type": "USB",
                "serialNumber": "3C43352C",
                "module": 6,
                "model": "USBHub3c",
                "entitlements": [
                    {
                        "tag": "CONTROL",
                        "version": 0
                    }
                ]
            },
            {
                "type": "USB",
                "serialNumber": "F7D9AFB6",
                "module": 6,
                "model": "USBHub3p",
```


(continues on next page)

(continued from previous page)

```
                    "entitlements": [
                        {
                            "tag": "CONTROL",
                            "version": 0
                        }
                    ]
                }
            ]
        }
}
```

acronameDevicesState Endpoint

The `acronameDevicesState` endpoint provides telemetry and state information for all of the attached Acroname devices on the host system. It provides USB connection information, custom names, and device version information for each port on the device.

There are no input parameters to this endpoint. It will return a JSON array of devices, or error, encapsulated in a *transaction envelope*.

JSON Schema for Response Object

The following table lists a JSON Schema[93] for the response object.

Download the raw JSON Schema file for this response: `acroname_devices_state_response.json`

Object Fields of Acroname Devices State Response

Name	Type	Mandatory	Description
`timestamp`	*v1-timestamp*	Yes	Time and date that this object was created.
`sequence`	*v1-uint32*	Yes	Sequence counter that increments on each message sent.
`brainstemVersion`	*v1-version*	Yes	Version of the BrainStem library for this object.
`acronameDevices`	Array<*Object*>	Yes	Array containing all the available Acroname devices on the system.

[93] https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

Object Fields of `acronameDevices[]`

Name	Type	Mandatory	Description
`serialNumber`	*v1-serialNumber*	Yes	Acroname Serial Number
`firmwareVersion`	*v1-version*	Yes	Firmware version that is running on the device.
`model`	*v1-model*	Yes	Model Name of the device.
`customName`	string	No	If set, a user-defined name for the device.
`upstreamPort`	*v1-uint8*	No	Currently active upstream port index.
`upstreamPortSelectionIsAutomatic`	boolean	Yes	Is upstream port selected automatically.
`ports`	Array<*Object*>	No	Object representing a single port on a device.

Object Fields of `acronameDevices[].ports[]`

Name	Type	Mandatory	Description
`index`	*v1-uint8*	Yes	Physical Port Number on device. This may include upstream ports, control ports, etc.
`physicalName`	string	Yes	Description of the port type and number.
`customName`	string	No	If set, a user-defined name for the port.
`speed`	string	No	Current data speed of the port.
`roles`	Array<*v1-dataRole*>	Yes	All supported data roles of the port.
`currentDataRole`	*v1-dataRole*	Yes	Current data role of the port.
`enabled`	boolean	No	Whether or not the port is enabled.
`voltage`	*v1-portParam*	No	Voltage measurement from this port.
`current`	*v1-portParam*	No	Current measurement from this port.

v1-timestamp

Name	Value
Type	string
description	String containing a ISO8601 timestamp (YYYY-MM-DDTHH:MM:SS.mmmZ
Regular Expression	`^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{3,6}Z$`

v1-serialNumber

Name	Value
Type	string
description	Hexadecimal representation of an Acroname device.
Regular Expression	`^(0x)?[A-Fa-f0-9]{8}$`

v1-version

Name	Value
Type	*Object*
description	Object representing a <major>.<minor>.<patch> semantic version.

Name	Type	Mandatory	Description
`major`	*v1-uint8*	Yes	
`minor`	*v1-uint8*	Yes	
`patch`	*v1-uint8*	Yes	

v1-model

Name	Value
Type	string
description	Enumeration of all supported Acroname devices.
Enumerations	`USBStem`, `EtherStem`, `MTMIOSerial`, `MTMPM1`, `MTMEtherStem`, `MTMUSBStem`, `USBHub2x4`, `MTMRelay`, `USBHub3p`, `MTMDAQ1`, `USBCSwitch`, `MTMDAQ2`, `MTMLoad1`, `USBHub3c`, `Unknown`

v1-dataRole

Name	Value
Type	string
description	Enumeration of all supported data roles for a port.
Enumerations	`Disabled`, `Upstream`, `Downstream`, `Control`, `Unknown`

v1-portParam

Name	Value
Type	*Object*
description	Object containing a data parameter with units and scale values.

Name	Type	Mandatory	Description
`rawValue`	number	Yes	Unformatted return value from the device.
`value`	number	Yes	If appropriate, a user-friendly formatted value.
`units`	*v1-units*	Yes	If appropriate, a user-friendly unit string.

v1-uint8

Name	Value
Type	integer
description	Unsigned 8-bit integer
minimum	0
maximum	255

v1-uint32

Name	Value
Type	integer
description	Unsigned 32-bit integer
minimum	0
maximum	4294967295

v1-units

Name	Value
Type	string
description	Allowed and recognized units for a value.
Enumerations	`volts`, `millivolts`, `microvolts`, `amperes`, `milliamperes`, `microamperes`, `coulombs`, `millicoulombs`, `microcoulombs`, `seconds`, `milliseconds`, `microseconds`, `nanoseconds`, `kilowatts`, `watts`, `milliwatts`, `microwatts`, `kilowatt-hours`, `watt-hours`, `milliwatt-hours`, `microwatt-hours`

v1-aErr

Name	Value
Type	string
description	Enumeration of all possible error results
Enumerations	`aErrNone`, `aErrMemory`, `aErrParam`, `aErrNotFound`, `aErrFileNameLength`, `aErrBusy`, `aErrIO`, `aErrMode`, `aErrWrite`, `aErrRead`, `aErrEOF`, `aErrNotReady`, `aErrPermission`, `aErrRange`, `aErrSize`, `aErrOverrun`, `aErrParse`, `aErrConfiguration`, `aErrTimeout`, `aErrInitialization`, `aErrVersion`, `aErrUnimplemented`, `aErrDuplicate`, `aErrCancel`, `aErrPacket`, `aErrConnection`, `aErrIndexRange`, `aErrShortCommand`, `aErrInvalidEntity`, `aErrInvalidOption`, `aErrResource`, `aErrMedia`, `aErrAsyncReturn`, `aErrOperation`, `aErrUnknown`

v1-error

Name	Value
Type	*Object*
description	Error containing an acroname error

Name	Type	Mandatory	Description
`errorCode`	*v1-aErr*	Yes	Acroname error code
`errorMessage`	string	No	A human-readable message to describe the context of the error.

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevicesState
```

Python

```python
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/acronameDevicesState')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:


```json
{
    "timestamp": "2023-09-20T17:24:24.428Z",
    "request": {
        "endpointName": "/api/v1/acronameDevicesState",
        "parameters": {}
    },
    "response": {
        "timestamp": "2023-09-20T17:24:24.108Z",
        "sequence": 2563,
        "brainstemVersion": {
            "major": 2,
            "minor": 10,
            "patch": 0
        },
        "acronameDevices": [
            {
                "serialNumber": "0x3C43352C",
                "firmwareVersion": {
                    "major": 2,
                    "minor": 10,
                    "patch": 0
                },
                "model": "USBHub3c",
                "upstreamPort": 0,
                "upstreamPortSelectionIsAutomatic": true,
                "ports": [
                    {
                        "index": 0,
                        "physicalName": "Port 0",
                        "speed": "5 Gbps",
                        "roles": [
                            "Upstream",
                            "Downstream"
                        ],
                        "currentDataRole": "Upstream",
                        "enabled": true,
                        "voltage": {
                            "value": 5.207519,
                            "units": "volts",
                            "rawValue": 5207519
                        },
```


(continues on next page)

(continued from previous page)

```
                        "current": {
                            "value": 0.00061,
                            "units": "amperes",
                            "rawValue": 610
                        }
                    },
                    {
                        "index": 1,
                        "physicalName": "Port 1",
                        "speed": "Unknown",
                        "roles": [
                            "Upstream",
                            "Downstream"
                        ],
                        "currentDataRole": "Downstream",
                        "enabled": true,
                        "voltage": {
                            "value": 0.0,
                            "units": "volts",
                            "rawValue": 0
                        },
                        "current": {
                            "value": 0.000305,
                            "units": "amperes",
                            "rawValue": 305
                        }
                    },
                    <additional ports ...>
                ]
            }
        ]
    }
}
```

devices Endpoint

The `devices` endpoint provides a list of all USB devices attached to the host system, including device information, port path, capabilities, and mappings to Acroname devices.

There are no input parameters to this endpoint. It will return a JSON array of devices, encapsulated in a *transaction envelope*.

JSON Schema for Response Object

The following table lists a JSON Schema[94] for the response object.

Download the raw JSON Schema file for this response: `devices_response.json`

[94] https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

Object Fields of Devices Response

Name	Type	Mandatory	Description
`usb`	Array<*Object*>	Yes	An array containing all USB devices in the system

Object Fields of `usb[]`

Name	Type	Mandatory	Description
`vendorId`	*v1-vidPid*	Yes	Vendor Identifier
`productId`	*v1-vidPid*	Yes	Product Identifier
`speedActual`	*v1-speed*	Yes	Current enumerated speed
`maxSpeed`	*v1-speed*	Yes	Maximum supported speed
`serialNumber`	*v1-name*	Yes	Vendor-provided serial number
`manufacturer`	*v1-name*	Yes	Vendor-provided manufacturer name
`productName`	*v1-name*	Yes	Vendor-provided product name
`deviceClass`	*v1-classCode*	Yes	Device Class Code
`interfaces`	Array<*Objec*	Yes	List of all interfaces provided by device
`portPath`	Array<*v1-uint8*>	Yes	List of the logical IDs of each parent node, starting with the controller ID and followed by the port ID for each parent device.
`acronameDevice`	*Object*	No	If an Acroname device is detected in this device's parent tree, information about the Acroname device.
`bulkCapability`	*v1-capability*	Yes	Capability information for Bulk Transfers
`isochronousCapability`	*v1-capability*	Yes	Capability information for Isochronous Transfers
`interruptCapability`	*v1-capability*	Yes	Capability information for Interrupt Transfers

Object Fields of `usb[].interfaces[]`

Name	Type	Mandatory	Description
`interfaceIndex`	*v1-uint8*	Yes	Interface Index within this device
`entryIndex`	*v1-uint8*	Yes	Alternate Setting Index within this interface
`kClass`	*v1-classCode*	Yes	USB-IF class code for this interface
`kSubclass`	*v1-classCode*	Yes	USB-IF subclass code for this interface, qualified by the kClass code
`kProtocol`	*v1-uint8*	Yes	USB-IF protocol code for this interface, qualified by the kClass and kSubclass codes

Object Fields of `usb[].acronameDevice`

Name	Type	Mandatory	Description
`serialNumber`	*v1-serialNumber*	Yes	Acroname Serial Number
`port`	*v1-uint8*	Yes	Physical port number on the Acroname device
`distance`	*v1-uint8*	Yes	Number of parent tiers between Acroanme device and this device

v1-vidPid

Name	Value
Type	string
description	Representation of a USB Device or Vendor ID
Regular Expression	`^(0x)?[A-Fa-f0-9]{4}$`

v1-serialNumber

Name	Value
Type	string
description	Representation of an Acroname device serial number
Regular Expression	`^(0x)?[A-Fa-f0-9]{8}$`

v1-speed

Name	Value
Type	string
description	Enumeration of all possible USB speeds
Enumerations	`Unknown`, `1.5 Mbps`, `12 Mbps`, `480 Mbps`, `5 Gbps`, `10 Gbps`

v1-name

Name	Value
Type	string
description	USB-IF Compliant Name Field
Maximum # of Glyphs	`126`

v1-classCode

Name	Value
Type	integer
description	Enumeration of all possible USB device/interface classes (https://vovkos.github.io/doxyrest/samples/libusb/enum_libusb_class_code.html)
Enumerations	`0`, `1`, `2`, `3`, `5`, `6`, `7`, `8`, `9`, `10`, `11`, `13`, `14`, `15`, `220`, `224`, `254`, `255`

v1-uint8

Name	Value
Type	integer
description	Unsigned 8-bit integer
minimum	0
maximum	255

v1-capability

Name	Value
Type	*Object*
description	Description for a capability of a device

Name	Type	Mandatory	Description
`isDoing`	boolean	Yes	Whether the device is currently using this capability
`isCapable`	boolean	Yes	Whether the device is capable of using this capability

v1-aErr

Name	Value
Type	string
description	Enumeration of all possible error results
Enumerations	`aErrNone`, `aErrMemory`, `aErrParam`, `aErrNotFound`, `aErrFileNameLength`, `aErrBusy`, `aErrIO`, `aErrMode`, `aErrWrite`, `aErrRead`, `aErrEOF`, `aErrNotReady`, `aErrPermission`, `aErrRange`, `aErrSize`, `aErrOverrun`, `aErrParse`, `aErrConfiguration`, `aErrTimeout`, `aErrInitialization`, `aErrVersion`, `aErrUnimplemented`, `aErrDuplicate`, `aErrCancel`, `aErrPacket`, `aErrConnection`, `aErrIndexRange`, `aErrShortCommand`, `aErrInvalidEntity`, `aErrInvalidOption`, `aErrResource`, `aErrMedia`, `aErrAsyncReturn`, `aErrOperation`, `aErrUnknown`

v1-error

Name	Value
Type	*Object*
description	Error containing an acroname error

Name	Type	Mandatory	Description
`errorCode`	*v1-aErr*	Yes	Acroname error code
`errorMessage`	string	No	A human-readable message to describe the context of the error.

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/devices
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/devices')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:


```
{
    "timestamp": "2023-09-20T17:36:15.359Z",
    "request": {
        "endpointName": "/api/v1/devices",
        "parameters": {}
    },
    "response": {
        "usb": [
            {
                "vendorId": "0x067B",
                "productId": "0x2303",
                "speedActual": "12 Mbps",
                "maxSpeed": "12 Mbps",
                "productName": "USB-Serial Controller",
                "serialNumber": "",
                "manufacturer": "Prolific Technology Inc.",
                "deviceClass": 0,
                "interfaces": [
                    {
                        "interfaceIndex": 0,
                        "entryIndex": 0,
                        "kClass": 255,
                        "kSubclass": 0,
                        "kProtocol": 0
                    }
                ],
                "portPath": [
                    4,
                    3,
                    1
                ],
                "acronameDevice": {
                    "serialNumber": "0xF7D9AFB6",
                    "port": 0,
                    "distance": 0
                },
                "bulkCapability": {
                    "isDoing": true,
                    "isCapable": true
                },
                "isochronousCapability": {
```


(continues on next page)

(continued from previous page)

```
                    "isDoing": false,
                    "isCapable": false
                },
                "interruptCapability": {
                    "isDoing": true,
                    "isCapable": true
                }
            },
            {
                "vendorId": "0x2188",
                "productId": "0x0747",
                "speedActual": "5 Gbps",
                "maxSpeed": "5 Gbps",
                "productName": "Card Reader ",
                "serialNumber": "000000000010",
                "manufacturer": "CalDigit",
                "deviceClass": 0,
                "interfaces": [
                    {
                        "interfaceIndex": 0,
                        "entryIndex": 0,
                        "kClass": 8,
                        "kSubclass": 6,
                        "kProtocol": 80
                    }
                ],
                "portPath": [
                    4,
                    8
                ],
                "bulkCapability": {
                    "isDoing": true,
                    "isCapable": true
                },
                "isochronousCapability": {
                    "isDoing": false,
                    "isCapable": false
                },
                "interruptCapability": {
                    "isDoing": false,
                    "isCapable": false
                }
            }
        ]
    }
}
```

brainstem Endpoint

This reference builds on understanding of the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation:

- *BrainStem Overview*
- *BrainStem Terminology*
- *Getting Started with the BrainStem.*

BrainStem GET commands are implemented with HTTP `GET` calls, and SET commands are implemented with HTTP `PUT` calls. The entity and operation fields are mapped directly to BrainStem API calls, minus the `get` and `set` prefixes on the operations.

The parameters for the command are given as slash-separated path entries:

- `GET http://<IPADDRESS>:<PORT>/api/v1/brainstem/<SERIALNUM>/<ENTITY>/<INDEX>/<COMMAND>`
- `GET http://<IPADDRESS>:<PORT>/api/v1/brainstem/<SERIALNUM>/<ENTITY>/<INDEX>/<COMMAND> {BODY}`

Table 6: BrainStem Request Parameters

Parameter	Request Type	Description
IPADDRESS	GET, PUT	IP Address of BrainD server hosting RESTful endpoints.
PORT	GET, PUT	Port that BrainD server is listening for connections. Default is `9005`.
SERIALNUM	GET, PUT	Serial Number of the BrainStem device being accessed
ENTITY	GET, PUT	*BrainStem Entity*, e.g. `system`, `temperature`, `digital`.
INDEX	GET, PUT	Entity Index
COMMAND	GET, PUT	Operation to perform on entity without `get` or `set` prefix, e.g. `inputvoltage`, `currentlimit`, `name`.
BODY	PUT	HTTP `PUT` payload, matching *PUT Request JSON Schema*

Note: The following BrainStem Entities are unsupported at this time: `app`, `i2c`, `powerDelivery`

Examples

GET Command

Bash

```
curl http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/inputvoltage
```

Python

```
import requests
import json
```

(continues on next page)

(continued from previous page)

```
response = requests.get('http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/
↪inputvoltage')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:


```
{
    "timestamp": "2023-09-20T17:53:32.480Z",
    "request": {
        "endpointName": "/api/v1/brainstem/3C43352C/system/0/inputvoltage",
        "parameters": {}
    },
    "response": {
        "value": 22974139,
        "rawValue": 22974139
    }
}
```


SET Command

Bash

```
curl -X PUT -d '{"value": "New Port Name"}' http://127.0.0.1:9005/api/v1/brainstem/
↪3C43352C/port/2/name
```

Python

```
import requests
import json

payload = {}
payload['value'] = 'New Port Name'

response = requests.put('http://127.0.0.1:9005/api/v1/brainstem/3C43352C/port/2/name',
↪ json=payload)
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:


```
{
    "timestamp": "2023-09-20T17:55:15.134Z",
    "request": {
        "endpointName": "/api/v1/brainstem/3C43352C/port/2/name",
        "parameters": {
            "value": "New Port Name"
        }
    },
    "response": {}
}
```

GET Command with no response

Bash

```
curl http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/save
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/save')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
{
    "timestamp": "2023-09-20T17:57:05.245Z",
    "request": {
        "endpointName": "/api/v1/brainstem/3C43352C/system/0/save",
        "parameters": {}
    },
    "response": {}
}
```

JSON Schema for GET Response Object

The following table lists a JSON Schema[95] for the response object.

Download the raw JSON Schema file for this response: `get_brainstem_response.json`

Object Fields of Get Brainstem Response

Name	Type	Mandatory	Description
`rawValue`	*v1-rawValue*	Yes	
`value`	string *or* number	No	If appropriate, a user-friendly formatted value.
`units`	*v1-units*	No	

[95] https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

v1-aErr

Name	Value
Type	string
description	Enumeration of all possible error results
Enumerations	aErrNone, aErrMemory, aErrParam, aErrNotFound, aErrFileNameLength, aErrBusy, aErrIO, aErrMode, aErrWrite, aErrRead, aErrEOF, aErrNotReady, aErrPermission, aErrRange, aErrSize, aErrOverrun, aErrParse, aErrConfiguration, aErrTimeout, aErrInitialization, aErrVersion, aErrUnimplemented, aErrDuplicate, aErrCancel, aErrPacket, aErrConnection, aErrIndexRange, aErrShortCommand, aErrInvalidEntity, aErrInvalidOption, aErrResource, aErrMedia, aErrAsyncReturn, aErrOperation, aErrUnknown

v1-units

Name	Value
Type	string
description	Allowed and recognized units for a value.
Enumerations	volts, millivolts, microvolts, amperes, milliamperes, microamperes, coulombs, millicoulombs, microcoulombs, seconds, milliseconds, microseconds, nanoseconds, kilowatts, watts, milliwatts, microwatts, kilowatt-hours, watt-hours, milliwatt-hours, microwatt-hours

v1-uint32

Name	Value
Type	integer
description	Unsigned 32-bit integer
minimum	0
maximum	4294967295

v1-rawValue

Name	Value
Type	*v1-uint32* *or* Array<*v1-uint32*>
description	Unformatted return value from the device.

v1-error

Name	Value
Type	*Object*
description	Error containing an acroname error

Name	Type	Mandatory	Description
`errorCode`	*v1-aErr*	Yes	Acroname error code
`errorMessage`	string	No	A human-readable message to describe the context of the error.

JSON Schema for PUT Request Object

The following table lists a JSON Schema[96] for the response object.

Download the raw JSON Schema file for this response: `put_brainstem_payload_v1.json`

Object Fields of Put Brainstem Payload (v1)

Name	Type	Mandatory	Description
`value`	integer *or* string *or* boolean *or* Array<integer *or* string>	Yes	Value to set on the Device.

[96] https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

JSON Schema for PUT Response Object

The following table lists a JSON Schema[97] for the response object.

Download the raw JSON Schema file for this response: `put_brainstem_response.json`

Object Fields of Put Brainstem Response

Name	Type	Mandatory	Description
`errorCode`	*v1-aErr*	Yes	Acroname error code
`errorMessage`	string	No	A human-readable message to describe the context of the error.
- or -			
`thisMessageLeftIntentionallyBlank`	null	No	

v1-aErr

Name	Value
Type	string
description	Enumeration of all possible error results
Enumerations	`aErrNone`, `aErrMemory`, `aErrParam`, `aErrNotFound`, `aErrFileNameLength`, `aErrBusy`, `aErrIO`, `aErrMode`, `aErrWrite`, `aErrRead`, `aErrEOF`, `aErrNotReady`, `aErrPermission`, `aErrRange`, `aErrSize`, `aErrOverrun`, `aErrParse`, `aErrConfiguration`, `aErrTimeout`, `aErrInitialization`, `aErrVersion`, `aErrUnimplemented`, `aErrDuplicate`, `aErrCancel`, `aErrPacket`, `aErrConnection`, `aErrIndexRange`, `aErrShortCommand`, `aErrInvalidEntity`, `aErrInvalidOption`, `aErrResource`, `aErrMedia`, `aErrAsyncReturn`, `aErrOperation`, `aErrUnknown`

v1-error

Name	Value
Type	*Object*
description	Error containing an acroname error

[97] https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

Name	Type	Mandatory	Description
`errorCode`	*v1-aErr*	Yes	Acroname error code
`errorMessage`	string	No	A human-readable message to describe the context of the error.

version Endpoint

The `version` endpoint shows version information for various software components, including BrainStem version, BrainD version, and build information.

There are no input parameters to this endpoint. It will return a JSON object as described below. This endpoint does not use a transaction envelope. Contents are returned exactly as described in the schema below.

JSON Schema for Response Object

The following table lists a JSON Schema[98] for the response object.

Download the raw JSON Schema file for this response: `get_version_response.json`

Object Fields of Get Version Response

Name	Type	Mandatory	Description
`braind`	*v1-buildInfo*	Yes	Version and Build Information for the braind daemon.
`brainstem`	*v1-buildInfo*	Yes	Version and Build Information for the BrainStem library.
`controlRoom`	*v1-buildInfo*	No	Version and Build Information for the Control Room application.

v1-version

Name	Value
Type	*Object*
description	Object representing a <major>.<minor>.<patch> semantic version.

98 https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

Name	Type	Mandatory	Description
`major`	*v1-uint8*	Yes	
`minor`	*v1-uint8*	Yes	
`patch`	*v1-uint8*	Yes	

v1-buildInfo

Name	Value
Type	*Object*
description	Version and Build Information for an arbitrary software component.

Name	Type	Mandatory	Description
`version`	*v1-version*	Yes	
`buildDate`	string	No	Date and Time that this software component was built.
`buildHash`	string	No	Hexadecimal Git commit ID that was used for this build.

v1-uint8

Name	Value
Type	integer
description	Unsigned 8-bit integer
minimum	0
maximum	255

Examples

Bash

```
curl http://127.0.0.1:9005/api/v1/version
```

Python

```
import requests
import json
response = requests.get('http://127.0.0.1:9005/api/v1/version')
json_data = response.json()
print(json.dumps(json_data, indent=4))
```

The output will be similar to the following:

```
{
    "braind": {
        "version": {
            "major": 1,
            "minor": 0,
            "patch": 1
        },
        "buildDate": "Wed Aug 23 13:35:12 2023",
        "buildHash": "b1ba5c6153b7fbbb65c868b65c8ac70bb3d7b7dd"
    },
    "brainstem": {
        "version": {
            "major": 2,
            "minor": 10,
            "patch": 0
        },
        "buildDate": "2023-09-11T17:14:25Z",
        "buildHash": "7af9c07e44601d6987fe3dab0ea201a13609683e"
    }
}
```

Transaction Envelope

Most RESTful Endpoints respond with this JSON object, where `response` is populated by an endpoint-specific schema. The Transaction Envelope schema enables providing error information, request context, and other useful fields about the request and response payload for the specific API call.

The *Acroname Devices*, *Acroname Devices State endpoint* , *Devices*, and *BrainStem* endpoints use this schema as an envelope.

The *Version* endpoint does not use this envelope.

JSON Schema for Transaction Envelope

The following table lists a JSON Schema[99] .

Download the raw JSON Schema file: `transaction_envelope.json`

[99] https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-01

Object Fields of BrainD V1 Transaction Envelope

Name	Type	Mandatory	Description
`request`	*Object*	Yes	
`response`	*v1-error* *or* boolean *or* object *or* array *or* number *or* string	Yes	
`developerNotes`	Array<null>	No	
`timestamp`	*v1-timestamp*	Yes	When the call completed, in ISO8601 time.

Object Fields of `request`

Name	Type	Mandatory	Description
`endpointName`	string	Yes	Path-section of the URL
`params`	*Object*	No	Parameters decoded from the HTTP body.

Object Fields of `request.params`

Name	Type	Mandatory	Description

v1-aErr

Name	Value
Type	string
description	Enumeration of all possible error results
Enumerations	`aErrNone`, `aErrMemory`, `aErrParam`, `aErrNotFound`, `aErrFileNameLength`, `aErrBusy`, `aErrIO`, `aErrMode`, `aErrWrite`, `aErrRead`, `aErrEOF`, `aErrNotReady`, `aErrPermission`, `aErrRange`, `aErrSize`, `aErrOverrun`, `aErrParse`, `aErrConfiguration`, `aErrTimeout`, `aErrInitialization`, `aErrVersion`, `aErrUnimplemented`, `aErrDuplicate`, `aErrCancel`, `aErrPacket`, `aErrConnection`, `aErrIndexRange`, `aErrShortCommand`, `aErrInvalidEntity`, `aErrInvalidOption`, `aErrResource`, `aErrMedia`, `aErrAsyncReturn`, `aErrOperation`, `aErrUnknown`

v1-error

Name	Value
Type	*Object*
description	Error containing an acroname error

Name	Type	Mandatory	Description
`errorCode`	*v1-aErr*	Yes	Acroname error code
`errorMessage`	string	No	A human-readable message to describe the context of the error.

v1-timestamp

Name	Value
Type	string
description	String containing a ISO8601 timestamp (YYYY-MM-DDTHH:MM:SS.mmmZ
Regular Expression	`^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{3,6}Z$`

Examples

These are examples of payloads for endpoints that use a transaction envelope. These examples closely follow the examples on every endpoint's reference page, but include the envelope for verbosity.

Acroname Devices

The success-case of `response` is detailed in *Acroname Devices endpoint* page.

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevices
```

A reasonable output in the expected case:

```
{
    "timestamp": "2023-09-20T17:11:54.092Z",
    "request": {
        "endpointName": "/api/v1/acronameDevices",
        "parameters": {}
    },
    "response": {
        "acronameDevices": [
```

(continues on next page)

(continued from previous page)

```
            {
                "type": "USB",
                "serialNumber": "3C43352C",
                "module": 6,
                "model": "USBHub3c",
                "entitlements": [
                    {
                        "tag": "CONTROL",
                        "version": 0
                    }
                ]
            },
            {
                "type": "USB",
                "serialNumber": "F7D9AFB6",
                "module": 6,
                "model": "USBHub3p",
                "entitlements": [
                    {
                        "tag": "CONTROL",
                        "version": 0
                    }
                ]
            }
        ]
    }
}
```

The Acroname Device endpoint does not emit errors. But if it did, it might look something like:

```
{
    "timestamp": "2023-10-05T18:34:30.200Z",
    "request": {
        "endpointName": "/api/v1/acronameDevices",
        "parameters": {}
    },
    "response": {
        "errorCode": "aErrUnknown",
        "errorMessage": "Unhandled exception while gathering device names."
    }
}
```

Acroname Devices State

The success-case of `response` is detailed in *Acroname Devices State endpoint* page.

Bash

```
curl http://127.0.0.1:9005/api/v1/acronameDevicesState
```

A reasonable output in the expected case:

```
{
    "timestamp": "2023-09-20T17:24:24.428Z",
    "request": {
```

(continues on next page)

(continued from previous page)

```
        "endpointName": "/api/v1/acronameDevicesState",
        "parameters": {}
    },
    "response": {
        "timestamp": "2023-09-20T17:24:24.108Z",
        "sequence": 2563,
        "brainstemVersion": {
            "major": 2,
            "minor": 10,
            "patch": 0
        },
        "acronameDevices": [
            {
                "serialNumber": "0x3C43352C",
                "firmwareVersion": {
                    "major": 2,
                    "minor": 10,
                    "patch": 0
                },
                "model": "USBHub3c",
                "upstreamPort": 0,
                "upstreamPortSelectionIsAutomatic": true,
                "ports": [
                    {
                        "index": 0,
                        "physicalName": "Port 0",
                        "speed": "5 Gbps",
                        "roles": [
                            "Upstream",
                            "Downstream"
                        ],
                        "currentDataRole": "Upstream",
                        "enabled": true,
                        "voltage": {
                            "value": 5.207519,
                            "units": "volts",
                            "rawValue": 5207519
                        },
                        "current": {
                            "value": 0.00061,
                            "units": "amperes",
                            "rawValue": 610
                        }
                    },
                    {
                        "index": 1,
                        "physicalName": "Port 1",
                        "speed": "Unknown",
                        "roles": [
                            "Upstream",
                            "Downstream"
                        ],
                        "currentDataRole": "Downstream",
                        "enabled": true,
                        "voltage": {
                            "value": 0.0,
```

(continues on next page)

(continued from previous page)

```
                            "units": "volts",
                            "rawValue": 0
                        },
                        "current": {
                            "value": 0.000305,
                            "units": "amperes",
                            "rawValue": 305
                        }
                    },
                    <additional ports ...>
                ]
            }
        ]
    }
}
```

The *Acroname Devices State endpoint* emits an error in the case where licenses were not found.

```
{
    "timestamp": "2023-10-05T18:57:49.317Z",
    "request": {
        "endpointName": "/api/v1/acronameDevicesState",
        "parameters": {}
    },
    "response": {
        "errorCode": "aErrPermission",
        "errorMessage": "The call is not allowed, check permissions and licenses associated with this call"
    }
}
```

Devices

The success-case of `response` is detailed in the *Devices endpoint* page.

Bash

```
curl http://127.0.0.1:9005/api/v1/devices
```

A reasonable output in the expected case:

```
{
    "timestamp": "2023-09-20T17:36:15.359Z",
    "request": {
        "endpointName": "/api/v1/devices",
        "parameters": {}
    },
    "response": {
        "usb": [
            {
                "vendorId": "0x067B",
                "productId": "0x2303",
                "speedActual": "12 Mbps",
                "maxSpeed": "12 Mbps",
                "productName": "USB-Serial Controller",
```

(continues on next page)

(continued from previous page)


```
                "serialNumber": "",
                "manufacturer": "Prolific Technology Inc.",
                "deviceClass": 0,
                "interfaces": [
                    {
                        "interfaceIndex": 0,
                        "entryIndex": 0,
                        "kClass": 255,
                        "kSubclass": 0,
                        "kProtocol": 0
                    }
                ],
                "portPath": [
                    4,
                    3,
                    1
                ],
                "acronameDevice": {
                    "serialNumber": "0xF7D9AFB6",
                    "port": 0,
                    "distance": 0
                },
                "bulkCapability": {
                    "isDoing": true,
                    "isCapable": true
                },
                "isochronousCapability": {
                    "isDoing": false,
                    "isCapable": false
                },
                "interruptCapability": {
                    "isDoing": true,
                    "isCapable": true
                }
            },
            {
                "vendorId": "0x2188",
                "productId": "0x0747",
                "speedActual": "5 Gbps",
                "maxSpeed": "5 Gbps",
                "productName": "Card Reader ",
                "serialNumber": "000000000010",
                "manufacturer": "CalDigit",
                "deviceClass": 0,
                "interfaces": [
                    {
                        "interfaceIndex": 0,
                        "entryIndex": 0,
                        "kClass": 8,
                        "kSubclass": 6,
                        "kProtocol": 80
                    }
                ],
                "portPath": [
                    4,
                    8
```


(continues on next page)

(continued from previous page)

```
                ],
                "bulkCapability": {
                    "isDoing": true,
                    "isCapable": true
                },
                "isochronousCapability": {
                    "isDoing": false,
                    "isCapable": false
                },
                "interruptCapability": {
                    "isDoing": false,
                    "isCapable": false
                }
            }
        ]
    }
}
```

The *Devices endpoint* emits an error in the case where licenses were not found.

```
{
    "timestamp": "2023-10-05T19:11:51.113Z",
    "request": {
        "endpointName": "/api/v1/devices",
        "parameters": {}
    },
    "response": {
        "errorCode": "aErrPermission",
        "errorMessage": "The call is not allowed, check permissions and licenses associated with this call"
    }
}
```

BrainStem

The success-case of `response` is detailed in *BrainStem Endnpoint* page.

Bash

```
curl http://127.0.0.1:9005/api/v1/brainstem/3C43352C/system/0/inputvoltage
```

A reasonable output in the expected case:

```
{
    "timestamp": "2023-09-20T17:53:32.480Z",
    "request": {
        "endpointName": "/api/v1/brainstem/3C43352C/system/0/inputvoltage",
        "parameters": {}
    },
    "response": {
        "value": 22974139,
        "rawValue": 22974139
    }
}
```

The *BrainStem endpoint* will emit errors specific to malformed requests, or hardware exceptions.

Example: Invalid `value` field supplied

```
{
    "timestamp": "2023-10-05T20:11:55.218Z",
    "request": {
        "endpointName": "/api/v1/brainstem/F2C3B5AC/port/5/Enabled/",
        "parameters": {
            "value": "□"
        }
    },
    "response": {
        "errorCode": "aErrParse",
        "errorMessage": "Value can not be converted to an integer value."
    }
}
```

Example: 8-port hardare does not support index 9

```
{
    "timestamp": "2023-10-05T20:13:16.954Z",
    "request": {
        "endpointName": "/api/v1/brainstem/F2C3B5AC/port/9/Enabled/",
        "parameters": {
            "value": "1"
        }
    },
    "response": {
        "errorCode": "aErrUnimplemented",
        "errorMessage": "Functionality unimplemented"
    }
}
```

3.6 .NET API Reference

3.6.1 BrainStem2 CLI Types

enum class `Acroname::BrainStem2CLI::`**`aErr`**

Values:

enumerator **`aErrNone`**

0 - Success, no error.

enumerator **`aErrMemory`**

1 - Memory allocation.

enumerator **`aErrParam`**

2 - Invalid parameter.

enumerator **`aErrNotFound`**

3 - Not found.

enumerator **`aErrFileNameLength`**

4 - File name too long.

enumerator **`aErrBusy`**

5 - Resource busy.

enumerator **`aErrIO`**

6 - Input/Output error.

enumerator **`aErrMode`**

7 - Invalid Mode.

enumerator **`aErrWrite`**

8 - Write error.

enumerator **`aErrRead`**

9 - Read error.

enumerator **`aErrEOF`**

10 - End of file.

enumerator **`aErrNotReady`**

11 - Not ready, no bytes available.

enumerator **`aErrPermission`**

12 - Insufficient permissions.

enumerator **aErrRange**

13 - Value out of range.

enumerator **aErrSize**

14 - Invalid Size.

enumerator **aErrOverrun**

15 - Buffer/queue overrun.

enumerator **aErrParse**

16 - Parse error.

enumerator **aErrConfiguration**

17 - Configuration error.

enumerator **aErrTimeout**

18 - Timeout occurred.

enumerator **aErrInitialization**

19 - Initialization error.

enumerator **aErrVersion**

20 - Invalid version.

enumerator **aErrUnimplemented**

21 - Functionality unimplemented.

enumerator **aErrDuplicate**

22 - Duplicate request.

enumerator **aErrCancel**

23 - Cancelation occured, 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 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.

3.6.2 Analog Class

class **AnalogClass** : public EntityClass

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.

Public Functions

AnalogClass()

Constructors.

~AnalogClass()

Destructor.

!AnalogClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.

- **index** – The cmdANALOG index to be addressed.

aErr **getValue**(unsigned short %value)

Get the raw ADC output value in bits.

Note: Not all modules 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

3.6.3 App Class

class **AppClass** : public EntityClass

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()

Constructors.

~AppClass()

Destructor.

!AppClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdAPP index to be addressed.

aErr **execute**(const unsigned int appParam)

Execute the app reflex on the module. 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 msTimoue 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.

3.6.4 Clock Class

class **ClockClass** : public EntityClass

ClockClass. Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

Note: Clock time must be reset if power to the BrainStem module is lost.

Public Functions

ClockClass()

Constructors.

~ClockClass()

Destructor.

!ClockClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters
- **pModule** – The module.
- **index** – The cmdCLOCK index to be addressed.

aErr **getYear**(unsigned short %year)

Get the four digit year value (0-4095).

Parameters
year – Get the year portion of the real-time clock value.

Returns
Returns *common entity* return values

aErr **setYear**(const unsigned short year)

Set the four digit year value (0-4095).

Parameters
`year` – Set the year portion of the real-time clock value.
Returns
Returns *common entity* return values

aErr **getMonth**(unsigned char %month)

Get the two digit month value (1-12).
Parameters
`month` – The two digit month portion of the real-time clock value.
Returns
Returns *common entity* return values

aErr **setMonth**(const unsigned char month)

Set the two digit month value (1-12).
Parameters
`month` – The two digit month portion of the real-time clock value.
Returns
Returns *common entity* return values

aErr **getDay**(unsigned char %day)

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).
Parameters
`day` – The two digit day portion of the real-time clock value.
Returns
Returns *common entity* return values

aErr **setDay**(const unsigned char day)

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

3.6.5 Digital Class

class **DigitalClass** : public EntityClass

The *DigitalClass* is the interface to digital entities on BrainStem modules. Digital entities have the following 5 possabilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

Public Functions

DigitalClass()
Constructors.

~DigitalClass()
Destructor.

!DigitalClass ()
Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)
Initializes the class. Should only be called when manually creating classes.
Parameters
- **pModule** – The module.
- **index** – The cmdDIGITAL index to be addressed.

aErr **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 avaliable 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

3.6.6 Equalizer Class

class **EqualizerClass** : public EntityClass

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

Public Functions

EqualizerClass ()

Constructors.

~EqualizerClass (void)

Destructor.

!EqualizerClass (void)

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdEQUALIZER index to be addressed.

aErr **setReceiverConfig** (const unsigned char channel, const unsigned char config)

Sets the receiver configuration for a given channel.

Parameters

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

Returns

Returns *common entity* return values.

aErr **getReceiverConfig** (const unsigned char channel, unsigned char %config)

Gets the receiver configuration for a given channel.

Parameters

- **channel** – The equalizer receiver channel.
- **config** – Configuration of the receiver.

Returns

Returns *common entity* return values.

aErr **setTransmitterConfig** (const unsigned char config)

Sets the transmitter configuration

Parameters

config – Configuration to be applied to the transmitter.

Returns

Returns *common entity* return values.

aErr **getTransmitterConfig** (unsigned char %config)

Gets the transmitter configuration

Parameters

config – Configuration of the Transmitter.

Returns

Returns *common entity* return values.

3.6.7 I2C Class

class **I2CClass** : public EntityClass

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()

Constructor.

~I2CClass()

Destructor.

!I2CClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdI2C index to be addressed.

aErr **read**(const unsigned char address, const unsigned char 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

3.6.8 Module Class

class **ModuleClass**

ModuleClass. Provides a generic interface to a BrainStem hardware module. The Module class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

Public Functions

ModuleClass(const unsigned char address, const unsigned char model)
Constructors.

~ModuleClass()
Destructor.

!ModuleClass ()
Finalizer.

BrainStem2CLI::*aErr* **discoverAndConnect**(m_linkType transport)
A discovery-based connect. This member function will attempt to connect to the first BrainStem module found. If this module does not match the object type the connection will fail.

Parameters
transport – - Transport on which to search for available BrainStem link modules. See the m_linkType "transport" enum for supported transports.
Returns
aErr::aErrBusy - if the module is already in use.

Returns
aErr::aErrParam - if the transport type is undefined.

Returns
aErr::aErrNotFound - if the module cannot be found.

Returns
aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **`discoverAndConnect`**(m_linkType transport, unsigned int serialNum)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will

Parameters
- **`transport`** – - Transport on which to search for available BrainStem link modules. See the m_linkType "transport" enum for supported transports.
- **`serialNum`** – - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Returns
aErr::aErrBusy - if the module is already in use.

Returns
aErr::aErrParam - if the transport type is undefined.

Returns
aErr::aErrNotFound - if the module cannot be found.

Returns
aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **`connectFromSpec`**(m_linkSpec spec)

Connect to a link with a fully defined specifier.

Parameters
`spec` – - Connect to module with specifier.

Returns
aErr::aErrInitialization - If there is currently no link object.

Returns
aErr::aErrBusy - If the link is currently connected.

Returns
aErr::aErrParam - if the specifier is incorrect.

Returns
aErr::aErrNotFound - if the module cannot be found.

Returns
aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **`connect`**(m_linkType transport)

Connect using the current link specifier.

Parameters
`transport` – - Transport on which to search for available BrainStem link modules. See the m_linkType "transport" enum for supported transports.

Returns
aErr::aErrBusy - if the module is already in use.

Returns
aErr::aErrParam - if the type is incorrect or serialNum is not specified

Returns
aErr::aErrNotFound - if the module cannot be found.

Returns
aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **connect** (m_linkType transport, unsigned int serialNum)

Connect using the current link specifier.

Parameters
- **transport** – - Transport on which to search for available BrainStem link modules. See the m_linkType "transport" enum for supported transports.
- **serialNum** – - Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

Returns
aErr::aErrBusy - if the module is already in use.

Returns
aErr::aErrParam - if the type is incorrect or serialNum is not specified

Returns
aErr::aErrNotFound - if the module cannot be found.

Returns
aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **connectThroughLinkModule** (BrainStem2CLI::*ModuleClass* ^module)

Connect using link from another Module. This member function will connect to the same BrainStem used by given Module. If a link module is found on the specified transport, a connection will

Parameters
module – - Pointer to a valid Module class object.

Returns
aErr::aErrParam - if the module is undefined.

Returns
aErr::aErrNone - if the connect was successful.

BrainStem2CLI::*aErr* **disconnect** ()

Disconnect from the BrainStem module.

Returns
aErr::aErrResource - If the there is no valid connection.

Returns
aErr::aErrConnection - If the disconnect failed, due to a communication issue.

Returns
aErr::aErrNone If the disconnect was successful.

BrainStem2CLI::*aErr* **reconnect** ()

Reconnect using the current link specifier.

Returns
aErr::aErrBusy - if the module is already in use.

Returns
aErr::aErrParam - if the specifier is incorrect.

Returns
aErr::aErrNotFound - if the module cannot be found.

Returns
aErr::aErrNone If the connect was successful.

bool **isConnected** ()

Is the link connected to the BrainStem Module.

BrainStem2CLI::linkStatus **getStatus** ()

Check the status of the module connection.

Returns
linkStatus (see aLink.h for status values)

BrainStem2CLI::*aErr* **getConfig**(BrainStem2CLI::aEtherConfig ^config)

Gets the links current aEther configuration

Parameters
config – Variable to be filled with the config

Returns
aErrNone on success. aErrNotReady if the module does not have a link aErrParam if config is NULL

cli::array<unsigned int> ^**getIPv4Interfaces**(void)

Populates a list with all of the available IPv4 Interfaces.

Returns
An array of network interfaces.

BrainStem2CLI::*aErr* **setConfig**(BrainStem2CLI::aEtherConfig ^config)

Sets the links aEther configuration. Configuration must be applied BEFORE connecting

Parameters
config – Configuration to be applied

Returns
aErrNone on success. aErrPermission if the module is currently connected. aErrNotReady if the module does not have a link

void **setModuleAddress**(const unsigned char address)

Accessor to set the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

Parameters
address – The module address.

unsigned char **getModuleAddress**()

Accessor to get the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

Returns
The module address.

BrainStem2CLI::aErr getBuild (unsigned int% build)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Parameters
build – Variable the build will be placed in.

BrainStem2CLI::*aErr* **hasUEI**(const unsigned char command, const unsigned char option, const unsigned char index, const unsigned char flags)

Queries the module to determine if it implements a UEI. Each UEI has a command, option or variant, index and flag. The hasUEI method queries for a fully specified UEI. Returns aErrNone if the variation is supported and an appropriate error if not. This call is blocking for up to the nMSTimeout period.

Parameters
- **command** – One of the UEI commands (cmdXXX).
- **option** – The option or variant of the command.
- **index** – The entity index.
- **flags** – The flags (ueiOPTION_SET or ueiOPTION_GET).

BrainStem2CLI::aErr classQuantity (const unsigned char command, unsigned char% count)

Queries the module to determine how many entities of the specified class are implemented by the module. Zero is a valid return value. For example, calling classQuantity with the command parameter of cmdANALOG would return the number of analog entities implemented by the module.

Parameters

- **command** – One of UEI commands (cmdXXX).
- **count** – When the request is successful count is updated with the number of entities found.

BrainStem2CLI::aErr subClassQuantity (const unsigned char command, const unsigned char index, unsigned char% count)

Queries the module to determine how many subclass entities of the specified class are implemented by the module for a given entity index. This is used for entities which may be 2-dimensional. E.g. cmdMUX subclasses are the number of channels supported by a particular mux type (index); as a specific example, a module may support 4 UART channels, so subClassQuantity(cmdMUX, aMUX_UART...) could return 4. Zero is a valid return value.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **count** – The number of subclasses found.

BrainStem2CLI::aErr entityGroup (const unsigned char command, const unsigned char index, unsigned char% group)

Queries the module the group assigned to an entity and index. Entities groups are used to specify when certain hardware features are fundamentally related. E.g. certain hardware modules may have some digital pins associated with an adjustable voltage rail; these digitals would be in the same group as the rail. Zero is the default group.

Parameters

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **group** – Upon success, group is filled with the entities group value.

void **setNetworkingMode** (const bool mode)

Sets the networking mode of the module object. By default the module object is 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, [Optional] Nullable< unsigned int > networkInterface)

Finds the first module found on the given transport

Parameters

type – The transport type on which to search for devices. Valid m_linkType "linktypes" are accepted.

Returns

If found, the linkspec will be populated with the devices information. Values will be all zeros otherwise.

static BrainStem2CLI::m_linkSpec findModule (BrainStem2CLI::m_linkType type, unsigned int serialNum, [Optional] Nullable< unsigned int > networkInterface)

Finds the module with the given serial number on the given transport type.

Parameters

- **type** – The trasport 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, [Optional] Nullable< unsigned int > networkInterface)

Discover is called with a specified transport to search for link modules on that transport. The devices list is filled with device scecifiers. sDiscover returns aErrNone if the discovery process is successful, regardless of whether any links are found. An error is only returned if the link discovery process fails. Discovery can take some time.

Parameters

- **type** – Transport to search for available BrainStem link modules on. See the m_linkType "transport" enum for supported transports.
- **specList** – an empty list of specifiers that will be filled in.

Returns

aErr::aErrNotFound if no devices were found.

Returns

aErr::aErrNone on success.

3.6.9 Mux Class

class **MuxClass** : public 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 (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).

Public Functions

MuxClass()

Constructors.

~MuxClass()

Destructor.

!MuxClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdMUX index to be addressed.

aErr **getEnable**(unsigned char %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 chanel 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

3.6.10 Pointer Class

class **PointerClass** : public EntityClass

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.

Public Functions

PointerClass()

Constructor.

~PointerClass()

Destructor.

!PointerClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdPOINTER index to be addressed.

aErr **getOffset**(unsigned short %offset)

Get the offset of the pointer

Parameters

offset – The value of the offset.

Returns

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.

3.6.11 Port Class

class **PortClass** : public 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 ()

Constructors.

~PortClass ()

Destructor.

!PortClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdPORT index to be addressed.

aErr **getVbusVoltage** (int %microvolts)

Gets the Vbus Voltage

Parameters

microvolts – The voltage in microvolts (1 == 1e-6V) currently present on Vbus.

Returns

Returns *common entity* return values

aErr **getVbusCurrent** (int %microamps)

Gets the Vbus Current

Parameters

microamps – The current in microamps (1 == 1e-6A) currently present on Vbus.

Returns

Returns *common entity* return values

aErr **getVconnVoltage** (int %microvolts)

Gets the Vconn Voltage

Parameters

microvolts – The voltage in microvolts (1 == 1e-6V) currently present on Vconn.

Returns

Returns *common entity* return values

aErr **getVconnCurrent** (int %microamps)

Gets the Vconn Current

Parameters

microamps – The current in microamps (1 == 1e-6A) currently present on Vconn.

Returns

Returns *common entity* return values

aErr **getPowerMode** (unsigned char %powerMode)

Gets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

powerMode – The current power mode.

Returns

Returns *common entity* return values

aErr **setPowerMode** (const unsigned char powerMode)

Sets the Port Power Mode: Convenience Function of get/setPortMode

Parameters

powerMode – The power mode to be set.

Returns

Returns *common entity* return values

aErr **getEnabled** (unsigned char %enable)

Gets the current enable value of the port.

Parameters
enable – 1 = Fully enabled port; 0 = One or more disabled components.
Returns
Returns *common entity* return values

aErr **setEnabled** (const unsigned char enable)

Enables or disables the entire port.
Parameters
enable – 1 = Fully enable port; 0 = Fully disable port.
Returns
Returns *common entity* return values

aErr **getDataEnabled** (unsigned char %enable)

Gets the current enable value of the data lines.: Sub-component (Data) of getEnabled.
Parameters
enable – 1 = Data enabled; 0 = Data disabled.
Returns
Returns *common entity* return values

aErr **setDataEnabled** (const unsigned char enable)

Enables or disables the data lines. Sub-component (Data) of setEnabled.
Parameters
enable – 1 = Enable data; 0 = Disable data.
Returns
Returns *common entity* return values

aErr **getDataHSEnabled** (unsigned char %enable)

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled.
Parameters
enable – 1 = Data enabled; 0 = Data disabled.
Returns
Returns *common entity* return values

aErr **setDataHSEnabled** (const unsigned char enable)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.
Parameters
enable – 1 = Enable data; 0 = Disable data.
Returns
Returns *common entity* return values

aErr **getDataHS1Enabled** (unsigned char %enable)

Gets the current enable value of the High Speed A side (HSA) data lines. Sub-component of getDataHSEnabled.
Parameters
enable – 1 = Data enabled; 0 = Data disabled.
Returns
Returns *common entity* return values

aErr **setDataHS1Enabled** (const unsigned char enable)

Enables or disables the Hight Speed A side (HSA) data lines. Sub-component of set-DataHSEnabled.
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 Hight Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSSEnabled**(unsigned char %enable)

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSSEnabled**(const unsigned char enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Parameters

enable – 1 = Enable data; 0 = Disable data.

Returns

Returns *common entity* return values

aErr **getDataSS1Enabled**(unsigned char %enable)

Gets the current enable value of the Super Speed A side (SSA) data lines.: Sub-component of getDataSSEnabled.

Parameters

enable – 1 = Data enabled; 0 = Data disabled.

Returns

Returns *common entity* return values

aErr **setDataSS1Enabled**(const unsigned char enable)

Enables or disables the Super Speed (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 he 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 filed
- **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 **getCCCurrentLimit**(unsigned char %value)

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations.

Parameters
value – Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A

Returns
Returns *common entity* return values

aErr **setCCCurrentLimit**(const unsigned char value)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations.

Parameters
value – Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A

Returns
Returns *common entity* return values

aErr **getDataHSRoutingBehavior**(unsigned char %mode)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters
mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **setDataHSRoutingBehavior**(const unsigned char mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters
mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **getDataSSRoutingBehavior**(unsigned char %mode)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters
mode – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **setDataSSRoutingBehavior**(const unsigned char mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Parameters
mode – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

Returns
Returns *common entity* return values

aErr **getVbusAccumulatedPower**(int %milliwatthours)

Gets the Vbus Accumulated Power

Parameters
milliwatthours – The accumuled 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

aErr **getCC1State** (unsigned short %value)
Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.
Parameters
`value` – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:
- None = 0 = portCC1State_None
- Invalid = 1 = portCC1State_Invalid
- Rp (default) = 2 = portCC1State_RpDefault
- Rp (1.5A) = 3 = portCC1State_Rp1p5
- Rp (3A) = 4 = portCC1State_Rp3p0
- Rd = 5 = portCC1State_Rd
- Ra = 6 = portCC1State_Ra
- Managed by controller = 7 = portCC1State_Managed
- Unknown = 8 = portCC1State_Unknown

Returns
Returns *common entity* return values

aErr **getCC2State** (unsigned short %value)
Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Parameters

`value` – Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:

- None = 0 = portCC2State_None
- Invalid = 1 = portCC2State_Invalid
- Rp (default) = 2 = portCC2State_RpDefault
- Rp (1.5A) = 3 = portCC2State_Rp1p5
- Rp (3A) = 4 = portCC2State_Rp3p0
- Rd = 5 = portCC2State_Rd
- Ra = 6 = portCC2State_Ra
- Managed by controller = 7 = portCC2State_Managed
- Unknown = 8 = portCC2State_Unknown

Returns

Returns *common entity* return values

3.6.12 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.

3.6.13 Power Delivery Class

class **PowerDeliveryClass** : public EntityClass

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()

Constructors.

~PowerDeliveryClass(void)

Destructor.

!PowerDeliveryClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdPOWERDELIVERY index to be addressed.

aErr **getConnectionState**(unsigned char %state)

Gets the current state of the connection in the form of an enumeration.

Parameters

state – Pointer to be filled with the current connection state.

Returns

Returns *common entity* return values

aErr **getNumberOfPowerDataObjects**(const unsigned char partner, const unsigned char powerRole, unsigned char %numRules)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

Parameters

- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **numRules** – Variable to be filled with the number of PDOs.

Returns

Returns *common entity* return values

aErr **getPowerDataObject**(const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex, unsigned int %pdo)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

Parameters

- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Variable to be filled with the requested power rule.

Returns

Returns *common entity* return values

aErr **setPowerDataObject**(const unsigned char powerRole, const unsigned char ruleIndex, const unsigned int pdo)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Power Data Object to be set.

Returns

Returns *common entity* return values

aErr **resetPowerDataObjectToDefault**(const unsigned char powerRole, const unsigned char ruleIndex)

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

Parameters

- **powerRole** – Indicates which power role of PD connection is in question.

- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

Returns

Returns *common entity* return values

aErr **getPowerDataObjectList** (unsigned int %buffer, const unsigned int 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 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 filed 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

3.6.14 Rail Class

class **`RailClass`** : public 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()

Constructor.

~RailClass()

Destructor.

!RailClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdRAIL index to be addressed.

aErr **getCurrent**(int %microamps)

Get the rail current.

Parameters

microamps – The current in micro-amps (1 == 1e-6A).

Returns

Returns *common entity* return values

aErr **setCurrentSetpoint**(const int microamps)

Set the rail 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 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 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 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 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

3.6.15 RCServo Class

class **RCServoClass** : public EntityClass

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 ()

Constructor.

~RCServoClass ()

Destructor.

!RCServoClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdRCSERVER index to be addressed.

aErr **setEnable** (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 1ms pulse and 192 = a 2ms pulse.

Returns
Returns *common entity* return values

aErr **getPosition**(unsigned char %position)

Get the position of the servo channel

Parameters
position – The current position of the servo channel. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

Returns
Returns *common entity* return values

aErr **setReverse**(const unsigned char reverse)

Set the output to be reversed on the servo channel

Parameters
reverse – Reverses the value set by "setPosition". 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

3.6.16 Relay Class

class **RelayClass** : public EntityClass

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()

Constructor.

~RelayClass()

Destructor.

!RelayClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters
- **pModule** – The module.

- **index** – The cmdRELAY index to be addressed.

aErr **setEnable** (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

3.6.17 Signal Class

See the *Signal Entity* for generic information.

class **SignalClass** : public EntityClass

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 ()

Constructors.

~SignalClass ()

Destructor.

!SignalClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdSIGNAL index to be addressed.

aErr **setEnable** (const unsigned char enable)

Enable/Disable the signal output.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr **getEnable** (unsigned char %enable)

Get the Enable/Disable of the signal.

Parameters

enable – True to enable, false to disable

Returns

Returns *common entity* return values

aErr **setInvert** (const unsigned char invert)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert – True to invert, false for normal mode.

Returns

Returns *common entity* return values

aErr **getInvert** (unsigned char %invert)

Get the invert status the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Parameters

invert – True to invert, false for normal mode.

Returns

Returns *common entity* return values

aErr **setT3Time** (const 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

3.6.18 Store Class

class **StoreClass** : public EntityClass

StoreClass. The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

Public Functions

StoreClass()

Constructors.

~StoreClass()

Destructor.

!StoreClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdSTORE index to be addressed.

aErr **getSlotState**(const unsigned char slot, unsigned char %state)

Get slot state.

Parameters

- **slot** – The slot number.
- **state** – true: enabled, false: disabled.

Returns
Returns *common entity* return values

aErr **loadSlot**(const unsigned char slot, const unsigned char %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

3.6.19 System Class

class **SystemClass** : public 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()

Constructors.

~SystemClass()

Destructor.

!SystemClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdSYSTEM index to be addressed.

aErr **getModule**(unsigned char %address)

Get the current address the module uses on the BrainStem network.

Parameters

address – The address the module is using on the BrainStem network.

Returns

Returns *common entity* return values

aErr **getModuleBaseAddress**(unsigned char %address)

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

Parameters

address – The address the module is using on the BrainStem network.

Returns

Returns *common entity* return values

aErr **setRouter**(const unsigned char address)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to "drop off" the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

Parameters

address – The router address to be used.

Returns
Returns *common entity* return values

aErr **getRouter**(unsigned char %address)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

Parameters
address – The address.

Returns
Returns *common entity* return values

aErr **setHBInterval**(const unsigned char interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

Parameters
interval – The desired heartbeat delay.

Returns
Returns *common entity* return values

aErr **getHBInterval**(unsigned char %interval)

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

Parameters
interval – The current heartbeat delay.

Returns
Returns *common entity* return values

aErr **setLED**(const unsigned char 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 **setLEDMaxBrightness**(const unsigned char brightness)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Parameters
brightness – Brightness value relative to 255

Returns
Returns *common entity* return values

aErr **getLEDMaxBrightness** (unsigned char %brightness)

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum).

Parameters
brightness – Brightness value relative to 255

Returns
Returns *common entity* return values

aErr **setBootSlot** (const unsigned char slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

Parameters
slot – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

Returns
Returns *common entity* return values

aErr **getBootSlot** (unsigned char %slot)

Get the store slot which is mapped when the module boots.

Parameters
slot – The slot number in aSTORE_INTERNAL that is mapped after the module boots.

Returns
Returns *common entity* return values

aErr **getVersion** (unsigned int %build)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

Parameters
build – The build version date code.

aErr **getBuild** (unsigned int %build)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Parameters
build – Variable to be filled with build.

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.

Note: aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

Returns
Returns aErrTimeout, from *common entity*, on success.

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 occured. 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

aErr **getName**(unsigned char %buffer, const unsigned int bufLength, unsigned int %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 filed
- **unloadLength** – 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 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

3.6.20 Temperature Class

class **TemperatureClass** : public EntityClass

TemperatureClass. This entitiy is only available on certain modules, and provides a temperature reading in microcelsius.

Public Functions

TemperatureClass()

Constructors.

~TemperatureClass()

Destructor.

!TemperatureClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdTEMPERATURE index to be addressed.

aErr **getValue**(int %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

3.6.21 Timer Class

class **TimerClass** : public 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 ()

Constructor.

~TimerClass ()

Destructor.

!TimerClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdTIMER index to be addressed.

aErr **getExpiration** (unsigned int %usecDuration)

Get the currently set expiration time in microseconds. This is not a “live” timer. That is, it shows the expiration time originally set with setExpiration; it does not “tick down” to show the time remaining before expiration.

Parameters

usecDuration – The timer expiration duration in microseconds.

Returns

Returns *common entity* return values

aErr **setExpiration** (const 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.

3.6.22 UART Class

class **UARTClass** : public EntityClass

UARTClass. A UART is a "Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines.

Public Functions

UARTClass()
Constructor.

~UARTClass()
Destructor.

!UARTClass ()
Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)
Initializes the class. Should only be called when manually creating classes.
Parameters
- **pModule** – The module.
- **index** – The cmdUART index to be addressed.

aErr **setEnable**(const unsigned char 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

3.6.23 USB Class

class **USBClass** : public EntityClass

USBClass. The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

Public Functions

USBClass()
Constructors.

~USBClass()
Destructor.

!USBClass ()
Finalizer.

void init (Acroname::BrainStem2CLI::ModuleClass^ module, const unsigned char index)
Initializes the class. Should only be called when manually creating classes.

Parameters
- **pModule** – The module.
- **index** – The cmdUSB index to be addressed.

aErr **setPortEnable**(const unsigned char channel)
Enable both power and data lines for a port.

Parameters
`channel` – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setPortDisable** (const unsigned char channel)

Disable both power and data lines for a port.

Parameters
`channel` – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line.

Parameters
`channel` – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line.

Parameters
`channel` – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setHiSpeedDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters
`channel` – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setHiSpeedDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Parameters
`channel` – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setSuperSpeedDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Parameters
`channel` – The USB sub channel.

Returns
Returns *common entity* return values

aErr **setSuperSpeedDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, 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, usbUpstreamModePort1, and usbUpstreamModeNone

Parameters

mode – The Upstream port mode.

Returns

Returns *common entity* return values

aErr **getUpstreamState**(unsigned char %state)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

Parameters

state – The Upstream port state.

Returns

Returns *common entity* return values

aErr **setEnumerationDelay**(const unsigned int ms_delay)

Set the 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

3.6.24 USBSystem Class

class **`USBSystemClass`** : public EntityClass

USBSystem Class The USBSystem class provides high level control of the lower level Port Class.

Public Functions

USBSystemClass()

Constructor.

~USBSystemClass()

Destructor.

!USBSystemClass ()

Finalizer.

void init (BrainStem2CLI::ModuleClass^ module, const unsigned char index)

Initializes the class. Should only be called when manually creating classes.

Parameters

- **pModule** – The module.
- **index** – The cmdUSBSYSTEM index to be addressed.

aErr **getUpstream**(unsigned char %port)

Gets the upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstream**(const unsigned char port)

Sets the upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getEnumerationDelay**(unsigned int %msDelay)

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Parameters

msDelay – the current inter-port delay in milliseconds.

Returns

Returns *common entity* return values

aErr **setEnumerationDelay**(const unsigned int msDelay)

Sets the inter-port enumeration delay in milliseconds. 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 *PortClass::getDataRole()* on each individual port.

Parameters

roleList – A bit packed representation of the data role for all ports.

Returns

Returns *common entity* return values

aErr **getEnabledList** (unsigned int %enabledList)

Gets the current enabled status of all ports with a single call. Equivalent to calling *PortClass::setEnabled()* on each port.

Parameters

enabledList – Bit packed representation of the enabled status for all ports.

Returns

Returns *common entity* return values

aErr **setEnabledList** (const unsigned int enabledList)

Sets the enabled status of all ports with a single call. Equivalent to calling *PortClass::setEnabled()* on each port.

Parameters

enabledList – Bit packed representation of the enabled status for all ports to be applied.

Returns

Returns *common entity* return values

aErr **getModeList** (unsigned int %buffer, const unsigned int 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 filed
- **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 filed
- **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 filed
- **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 filed
- **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

aErr **getUpstreamHS**(unsigned char %port)

Gets the USB HighSpeed upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstreamHS**(const unsigned char port)

Sets the USB HighSpeed upstream port.

Parameters

port – The upstream port to set.

Returns

Returns *common entity* return values

aErr **getUpstreamSS**(unsigned char %port)

Gets the USB SuperSpeed upstream port.

Parameters

port – The current upstream port.

Returns

Returns *common entity* return values

aErr **setUpstreamSS**(const unsigned char port)

Sets the USB SuperSpeed upstream port.

Parameters

port – The upstream port to set.

Returns
Returns *common entity* return values

aErr **getOverride**(unsigned int %overrides)

Gets the current enabled overrides

Parameters
overrides – Bit mapped representation of the current override configuration.

Returns
Returns *common entity* return values

aErr **setOverride**(const unsigned int overrides)

Sets the current enabled overrides

Parameters
overrides – Overrides to be set in a bit mapped representation.

Returns
Returns *common entity* return values

aErr **getDataHSMaxDatarate**(unsigned int %datarate)

Gets the USB HighSpeed Max datarate

Parameters
datarate – Current maximum datarate for the USB HighSpeed signals.

Returns
Returns *common entity* return values

aErr **setDataHSMaxDatarate**(const unsigned int datarate)

Sets the USB HighSpeed Max datarate

Parameters
datarate – Maximum datarate for the USB HighSpeed signals.

Returns
Returns *common entity* return values

aErr **getDataSSMaxDatarate**(unsigned int %datarate)

Gets the USB SuperSpeed Max datarate

Parameters
datarate – Current maximum datarate for the USB SuperSpeed signals.

Returns
Returns *common entity* return values

aErr **setDataSSMaxDatarate**(const unsigned int datarate)

Sets the USB SuperSpeed Max datarate

Parameters
datarate – Maximum datarate for the USB SuperSpeed signals.

Returns
Returns *common entity* return values

3.7 CCA API Reference

3.7.1 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.

Returns common entity return values

Note: Not all modules are provide 16 useful bits; this value's least significant bits are zero-padded to 16 bits. Refer to the module's datasheet to determine analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 16 bit analog reading with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.
- **index** – The index of the entity in question.

void **analog_getVoltage**(unsigned int *id, struct Result *result, const int index)

Get the scaled micro volt value with reference to ground.

Returns common entity return values

Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.
- **index** – The index of the entity in question.

void **analog_getRange**(unsigned int *id, struct Result *result, const int index)

Get the analog input range.

Returns common entity return values

Parameters

- **`id`** – ID assigned through “module_createStem”
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 8 bit value corresponding to a discrete range option
- **`index`** – The index of the entity in question.

void **`analog_getEnable`**(unsigned int *id, struct Result *result, const int index)

Get the analog output enable status.

Returns common entity return values

Parameters

- **`id`** – ID assigned through “module_createStem”
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 0 if disabled 1 if enabled.
- **`index`** – The index of the entity in question.

void **`analog_setValue`**(unsigned int *id, struct Result *result, const int index, const unsigned short value)

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

Returns common entity return values

Note: Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module’s datasheet to determine analog bit depth and reference voltage.

Parameters

- **`id`** – ID assigned through “module_createStem”
- **`result`** – 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.

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.

Returns common entity return values

Note: Voltage range is dependent on the specific DAC channel range.

Parameters

- **`id`** – ID assigned through “module_createStem”
- **`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.

void **analog_setRange**(unsigned int *id, struct Result *result, const int index, const unsigned char range)

Set the analog input range.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

void **analog_setEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Set the analog output enable state.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **analog_setConfiguration**(unsigned int *id, struct Result *result, const int index, const unsigned char configuration)

Set the analog configuration.

EntityReturnValues "common entity" return values

aErrConfiguration - Entity does not support this configuration.

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **analog_getConfiguration**(unsigned int *id, struct Result *result, const int index)

Get the analog configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
  - Current configuration of the analog entity.
- **index** – The index of the entity in question.

void **analog_setBulkCaptureSampleRate**(unsigned int *id, struct Result *result, const int index, const unsigned int value)

Set the sample rate for this analog when bulk capturing.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

void **analog_getBulkCaptureSampleRate**(unsigned int *id, struct Result *result, const int index)

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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. upon success filled with current sample rate in samples per second (Hertz).
- **index** – The index of the entity in question.

void **analog_setBulkCaptureNumberOfSamples**(unsigned int *id, struct Result *result, const int index, const unsigned int value)

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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. number of samples.
- **index** – The index of the entity in question.

void **analog_initiateBulkCapture**(unsigned int *id, struct Result *result, const int index)

Initiate a BulkCapture on this analog. Captured measurements are stored in the module's RAM store (RAM_STORE) slot 0. Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

Returns common entity return values. When the bulk capture is complete getBulkCaptureState() will return either bulkCaptureFinished or bulkCaptureError.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **analog_getBulkCaptureState**(unsigned int *id, struct Result *result, const int index)

Get the current bulk capture state for this analog.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. the state of bulk capture.
  - Idle: bulkCaptureIdle = 0
  - Pending: bulkCapturePending = 1
  - Finished: bulkCaptureFinished = 2
  - Error: bulkCaptureError = 3
- **index** – The index of the entity in question.

3.7.2 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.

aErrNone success.

aErrTimeout The request timed out waiting to start execution.

aErrConnection No active link connection.

aErrNotFound the app reflex was not found or not enabled on the module.

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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 msTimoue milliseconds. This method will block for up to msTimeout.

aErrNone success.

aErrTimeout The request timed out waiting for a response.

aErrConnection No active link connection.

aErrNotFound the app reflex was not found or not enabled on the module.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure. The return value filled in from the result of executing the reflex routine.
- **index** – The index of the entity in question.
- **appParam** – The app parameter handed to the reflex.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

3.7.3 Clock Entity

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).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Get the year portion of the real-time clock value.
- **index** – The index of the entity in question.

void **clock_setYear**(unsigned int *id, struct Result *result, const int index, const unsigned short year)

Set the four digit year value (0-4095).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **clock_getMonth**(unsigned int *id, struct Result *result, const int index)

Get the two digit month value (1-12).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The two digit month portion of the real-time clock value.
- **index** – The index of the entity in question.

void **clock_setMonth**(unsigned int *id, struct Result *result, const int index, const unsigned char month)

Set the two digit month value (1-12).

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **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.

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).

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The two digit day portion of the real-time clock value.
- **index** – The index of the entity in question.

void **clock_setDay**(unsigned int *id, struct Result *result, const int index, const unsigned char day)

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **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.

void **clock_getHour**(unsigned int *id, struct Result *result, const int index)

Get the two digit hour value (0-23).

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The two digit hour portion of the real-time clock value.
- **index** – The index of the entity in question.

void **clock_setHour**(unsigned int *id, struct Result *result, const int index, const unsigned char hour)

Set the two digit hour value (0-23).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **clock_getMinute**(unsigned int *id, struct Result *result, const int index)

Get the two digit minute value (0-59).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The two digit minute portion of the real-time clock value.
- **index** – The index of the entity in question.

void **clock_setMinute**(unsigned int *id, struct Result *result, const int index, const unsigned char min)

Set the two digit minute value (0-59).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **clock_getSecond**(unsigned int *id, struct Result *result, const int index)

Get the two digit second value (0-59).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The two digit second portion of the real-time clock value.
- **index** – The index of the entity in question.

void **clock_setSecond**(unsigned int *id, struct Result *result, const int index, const unsigned char sec)

Set the two digit second value (0-59).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **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.

3.7.4 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.

Returns common entity return values

aErrConfiguration Entity does not support this configuration.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **configuration** – The configuration to be applied
  - Digital Input: digitalConfigurationInput = 0
  - Digital Output: digitalConfigurationOutput = 1
  - RCServo Input: digitalConfigurationRCServoInput = 2
  - RCServo Output: digitalConfigurationRCServoOutput = 3
  - High Z State: digitalConfigurationHiZ = 4
  - Digital Input: digitalConfigurationInputPullUp = 0
  - Digital Input: digitalConfigurationInputNoPull = 4
  - Digital Input: digitalConfigurationInputPullDown = 5

void **digital_getConfiguration** (unsigned int *id, struct Result *result, const int index)

Get the digital configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
  - Current configuration of the digital entity.

- **index** – The index of the entity in question.

void **digital_setState**(unsigned int *id, struct Result *result, const int index, const unsigned char state)

Set the logical state.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **digital_getState**(unsigned int *id, struct Result *result, const int index)

Get the state.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current state of the digital entity. 0 is logic low, 1 is logic high. Note: If in high Z state an error will be returned.
- **index** – The index of the entity in question.

void **digital_setStateAll**(unsigned int *id, struct Result *result, const int index, const unsigned int state)

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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.
- **index** – The index of the entity in question.

3.7.5 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.

Returns common entity return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone 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.

void **equalizer_getReceiverConfig**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Gets the receiver configuration for a given channel.

Returns common entity return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Configuration of the receiver.
- **index** – The index of the entity in question.
- **channel** – The equalizer receiver channel.

void **equalizer_setTransmitterConfig**(unsigned int *id, struct Result *result, const int index, const unsigned char config)

Sets the transmitter configuration

Returns common entity return values.

Parameters

- **id** – ID assigned through "module_createStem"

- **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.

void **equalizer_getTransmitterConfig**(unsigned int *id, struct Result *result, const int index)

Gets the transmitter configuration

Returns common entity return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Configuration of the Transmitter.
- **index** – The index of the entity in question.

3.7.6 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 readLength, unsigned char *buffer)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure. Read from a device on this I2C bus.
- **index** – The index of the entity in question.
- **address** – - The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **readLength** – - The length of the data to read in bytes.
- **buffer** – - The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAINSTEM_MAXPACKETBYTES - 5

void **i2c_write**(unsigned int *id, struct Result *result, const int index, const int address, const int bufferLength, unsigned char *buffer)

Write to a device on this I2C bus.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.
- **bufferLength** – - The length of the data to write in bytes.
- **buffer** – - The data to send to the device, This array should be no larger than aBRAINSTEM_MAXPACKETBYTES - 5

void **i2c_setPullup**(unsigned int *id, struct Result *result, const int index, const 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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
  - The speed setting value.
- **index** – The index of the entity in question.

3.7.7 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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. true: mux is enabled, false: the mux is disabled.
- **index** – The index of the entity in question.

void **mux_setEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char bEnable)

Enable the mux.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **mux_getChannel**(unsigned int *id, struct Result *result, const int index)

Get the current selected mux channel.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Indicates which chanel is selected.
- **index** – The index of the entity in question.

void **mux_setChannel**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Set the current mux channel.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **mux_getChannelVoltage**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of the indicated mux channel.

Returns common entity return values

Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.
- **index** – The index of the entity in question.
- **channel** – The channel in which voltage was requested.

void **mux_getConfiguration**(unsigned int *id, struct Result *result, const int index)

Get the configuration of the mux.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. integer representing the mux configuration either default, or split-mode.
- **index** – The index of the entity in question.

void **mux_setConfiguration**(unsigned int *id, struct Result *result, const int index, const int config)

Set the configuration of the mux.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **mux_getSplitMode**(unsigned int *id, struct Result *result, const int index)

Get the current split mode mux configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.
- **index** – The index of the entity in question.

void **mux_setSplitMode**(unsigned int *id, struct Result *result, const int index, const int splitMode)

Sets the mux's split mode configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

3.7.8 Pointer Entity

group **PointerEntity**

PointerClass: Allows access to the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the BrainStem scratchpad from the host, and provides a mechanism for allowing the host application and BrainStem relexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

void **pointer_getOffset** (unsigned int *id, struct Result *result, const int index)

Get the offset of the pointer

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The value of the offset.
- **index** – The index of the entity in question.

void **pointer_setOffset** (unsigned int *id, struct Result *result, const int index, unsigned short offset)

Set the offset of the pointer

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **pointer_getMode** (unsigned int *id, struct Result *result, const int index)

Get the mode of the pointer

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The mode: aPOINTER_MODE_STATIC or aPOINTER_MODE_AUTO_INCREMENT.
- **index** – The index of the entity in question.

void **pointer_setMode** (unsigned int *id, struct Result *result, const int index, unsigned char mode)

Set the mode of the pointer

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **pointer_getTransferStore** (unsigned int *id, struct Result *result, const int index)

Get the handle to the store.

All possible standard UEI return handles.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The handle of the store.
- **index** – The index of the entity in question.

void **pointer_setTransferStore** (unsigned int *id, struct Result *result, const int index, unsigned char handle)

Set the handle to the store.

All possible standard UEI return handles.

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **pointer_initiateTransferToStore** (unsigned int *id, struct Result *result, const int index, unsigned char transferLength)

Transfer data to the store.

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **transferLength** – The length of the data transfer.

void **pointer_initiateTransferFromStore** (unsigned int *id, struct Result *result, const int index, unsigned char transferLength)

Transfer data from the store.

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **`transferLength`** – The length of the data transfer.

void **`pointer_getChar`**(unsigned int *id, struct Result *result, const int index)

Get a char (1 byte) value from the pointer at this object's index, where elements are 1 byte long.

All possible standard UEI return values.

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The value of a single character (1 byte) stored in the pointer.
- **`index`** – The index of the entity in question.

void **`pointer_setChar`**(unsigned int *id, struct Result *result, const int index, const unsigned char value)

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

All possible standard UEI return values.

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`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.

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

All possible standard UEI return values.

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The value of a single short (2 byte) stored in the pointer.
- **`index`** – The index of the entity in question.

void **`pointer_setShort`**(unsigned int *id, struct Result *result, const int index, const unsigned short value)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

All possible standard UEI return values.

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`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.

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

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The value of a single int (4 byte) stored in the pointer.
- **index** – The index of the entity in question.

void **pointer_setInt** (unsigned int *id, struct Result *result, const int index, const unsigned int value)

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

All possible standard UEI return values.

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

3.7.9 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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The voltage in microvolts (1 == 1e-6V) currently present on Vbus.
- **index** – The index of the entity in question.

void **port_getVbusCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the Vbus Current

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current in microamps (1 == 1e-6A) currently present on Vbus.
- **index** – The index of the entity in question.

void **port_getVconnVoltage** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Voltage

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The voltage in microvolts (1 == 1e-6V) currently present on Vconn.
- **index** – The index of the entity in question.

void **port_getVconnCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the Vconn Current

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current in microamps (1 == 1e-6A) currently present on Vconn.
- **index** – The index of the entity in question.

void **port_getPowerMode** (unsigned int *id, struct Result *result, const int index)

Gets the Port Power Mode: Convenience Function of get/setPortMode

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current power mode.
- **index** – The index of the entity in question.

void **port_setPowerMode** (unsigned int *id, struct Result *result, const int index, const unsigned char powerMode)

Sets the Port Power Mode: Convenience Function of get/setPortMode

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **port_getEnabled**(unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Fully enabled port; 0 = One or more disabled components.
- **index** – The index of the entity in question.

void **port_setEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the entire port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Data enabled; 0 = Data disabled.
- **index** – The index of the entity in question.

void **port_setDataEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the data lines. Sub-component (Data) of setEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Data enabled; 0 = Data disabled.
- **index** – The index of the entity in question.

void **port_setDataHSEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Data enabled; 0 = Data disabled.
- **index** – The index of the entity in question.

void **port_setDataHS1Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Hight Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Data enabled; 0 = Data disabled.
- **index** – The index of the entity in question.

void **port_setDataHS2Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Hight Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Data enabled; 0 = Data disabled.
- **index** – The index of the entity in question.

void **port_setDataSSEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **port_getDataSS1Enabled**(unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Super Speed A side (SSA) data lines.: Sub-component of getDataSSEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Data enabled; 0 = Data disabled.
- **index** – The index of the entity in question.

void **port_setDataSS1Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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 getDataSSEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Data enabled; 0 = Data disabled.
- **index** – The index of the entity in question.

void **port_setDataSS2Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Super Speed B side (SSB) data lines. Sub-component of setDataSSEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Power enabled; 0 = Power disabled.
- **index** – The index of the entity in question.

void **port_setPowerEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or Disables the power lines. Sub-component (Power) of setEnable.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **port_getDataRole**(unsigned int *id, struct Result *result, const int index)

Gets the Port Data Role.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The data role to be set. See datasheet for details.
- **index** – The index of the entity in question.

void **port_getVconnEnabled**(unsigned int *id, struct Result *result, const int index)

Gets the current enable value of the Vconn lines.: Sub-component (Vconn) of getEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Vconn enabled; 0 = Vconn disabled.
- **index** – The index of the entity in question.

void **port_setVconnEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Vconn1 enabled; 0 = Vconn1 disabled.
- **index** – The index of the entity in question.

void **port_setVconn1Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = Vconn2 enabled; 0 = Vconn2 disabled.
- **index** – The index of the entity in question.

void **port_setVconn2Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = CC enabled; 0 = CC disabled.
- **index** – The index of the entity in question.

void **port_setCCEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = CC1 enabled; 0 = CC1 disabled.
- **index** – The index of the entity in question.

void **port_setCC1Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the CC1 lines. Sub-component of setCCEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 1 = CC2 enabled; 0 = CC2 disabled.
- **index** – The index of the entity in question.

void **port_setCC2Enabled**(unsigned int *id, struct Result *result, const int index, const unsigned char enable)

Enables or disables the CC2 lines. Sub-component of setCCEnabled.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **port_getVoltageSetpoint**(unsigned int *id, struct Result *result, const int index)

Gets the current voltage setpoint value for the port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. the voltage setpoint of the port in uV.
- **index** – The index of the entity in question.

void **port_setVoltageSetpoint**(unsigned int *id, struct Result *result, const int index, const unsigned int value)

Sets the current voltage setpoint value for the port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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 he port IS which may differ from what was requested.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the current state.
- **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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Bit mapped value representing the devices speed. See "Devices" reference for details.
- **index** – The index of the entity in question.

void **port_getMode**(unsigned int *id, struct Result *result, const int index)

Gets current mode of the port

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Bit mapped value representing the ports mode. See "Devices" reference for details.
- **index** – The index of the entity in question.

void **port_setMode**(unsigned int *id, struct Result *result, const int index, const unsigned int mode)

Sets the mode of the port

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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 "Devices" documentation for details.

void **port_getErrors**(unsigned int *id, struct Result *result, const int index)

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Bit mapped field representing the current errors of the ports
- **index** – The index of the entity in question.

void **port_getCurrentLimit**(unsigned int *id, struct Result *result, const int index)

Gets the current limit of the port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the limit in microAmps (uA).
- **index** – The index of the entity in question.

void **port_setCurrentLimit**(unsigned int *id, struct Result *result, const int index, const unsigned int limit)

Sets the current limit of the port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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).

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.
- **index** – The index of the entity in question.

void **port_setCurrentLimitMode**(unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the available power in milli-watts (mW).
- **`index`** – The index of the entity in question.

void **`port_getAllocatedPower`**(unsigned int *id, struct Result *result, const int index)

Gets the currently allocated power This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the allocated power in milli-watts (mW).
- **`index`** – The index of the entity in question.

void **`port_getPowerLimit`**(unsigned int *id, struct Result *result, const int index)

Gets the user defined power limit for the port.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the power limit in milli-watts (mW).
- **`index`** – The index of the entity in question.

void **`port_setPowerLimit`**(unsigned int *id, struct Result *result, const int index, const unsigned int limit)

Sets a user defined power limit for the port.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`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).

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.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.
- **index** – The index of the entity in question.

void **port_setPowerLimitMode**(unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **port_getName**(unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **port_setName**(unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.
- **bufferLength** – Length of the buffer to be transferred.

void **port_getCCCurrentLimit**(unsigned int *id, struct Result *result, const int index)

Gets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A
- **index** – The index of the entity in question.

void **port_setCCCurrentLimit**(unsigned int *id, struct Result *result, const int index, const unsigned char value)

Sets the CC Current Limit Resistance The CC Current limit is the value that's set for the pull up resistance on the CC lines for basic USB-C negotations.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – Variable to be filled with an enumerated representation of the CC Current limit. 0 = None, 1 = Default (500/900mA), 2 = 1.5A, and 3 = 3.0A

void **port_getDataHSRoutingBehavior**(unsigned int *id, struct Result *result, const int index)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.
- **index** – The index of the entity in question.

void **port_setDataHSRoutingBehavior**(unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.
- **index** – The index of the entity in question.

void **port_setDataSSRoutingBehavior**(unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **port_getVbusAccumulatedPower**(unsigned int *id, struct Result *result, const int index)

Gets the Vbus Accumulated Power

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The accumuled power on Vbus in milliwatt-hours.
- **index** – The index of the entity in question.

void **port_resetVbusAccumulatedPower**(unsigned int *id, struct Result *result, const int index)

Resets the Vbus Accumulated Power to zero.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **port_getVconnAccumulatedPower**(unsigned int *id, struct Result *result, const int index)

Gets the Vconn Accumulated Power

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The accumuled power on Vconn in milliwatt-hours.
- **index** – The index of the entity in question.

void **port_resetVconnAccumulatedPower**(unsigned int *id, struct Result *result, const int index)

Resets the Vconn Accumulated Power to zero.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **port_setHSBoost**(unsigned int *id, struct Result *result, const int index, const unsigned char boost)

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.
- **index** – The index of the entity in question.

void **port_resetEntityToFactoryDefaults**(unsigned int *id, struct Result *result, const int index)

Resets the PortClass Entity to it factory default configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **port_getCC1State**(unsigned int *id, struct Result *result, const int index)

Gets the current CC1 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:
  - None = 0 = portCC1State_None
  - Invalid = 1 = portCC1State_Invalid
  - Rp (default) = 2 = portCC1State_RpDefault
  - Rp (1.5A) = 3 = portCC1State_Rp1p5
  - Rp (3A) = 4 = portCC1State_Rp3p0
  - Rd = 5 = portCC1State_Rd
  - Ra = 6 = portCC1State_Ra
  - Managed by controller = 7 = portCC1State_Managed
  - Unknown = 8 = portCC1State_Unknown

- **index** – The index of the entity in question.

void **port_getCC2State**(unsigned int *id, struct Result *result, const int index)

Gets the current CC2 Strapping on local and remote The state is a bit packed value where the upper byte is used to represent the remote or partner device attached to the ports resistance and the lower byte is used to represent the local or hubs resistance.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an packed enumerated representation of the CC state. Enumeration values for each byte are as follows:
  - None = 0 = portCC2State_None
  - Invalid = 1 = portCC2State_Invalid
  - Rp (default) = 2 = portCC2State_RpDefault
  - Rp (1.5A) = 3 = portCC2State_Rp1p5
  - Rp (3A) = 4 = portCC2State_Rp3p0
  - Rd = 5 = portCC2State_Rd
  - Ra = 6 = portCC2State_Ra
  - Managed by controller = 7 = portCC2State_Managed
  - Unknown = 8 = portCC2State_Unknown
- **index** – The index of the entity in question.

3.7.10 PowerDelivery Entity

group **PowerDeliveryEntity**

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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Pointer to be filled with the current connection state.
- **index** – The index of the entity in question.

void **powerdelivery_getNumberOfPowerDataObjects**(unsigned int *id, struct Result *result, const int index, const unsigned char partner, const unsigned char powerRole)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the number of PDOs.
- **index** – The index of the entity in question.
- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink

void **powerdelivery_getPowerDataObject**(unsigned int *id, struct Result *result, const int index, const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the requested power rule.
- **index** – The index of the entity in question.
- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

void **powerdelivery_setPowerDataObject**(unsigned int *id, struct Result *result, const int index, const unsigned char powerRole, const unsigned char ruleIndex, const unsigned int pdo)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone 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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone 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.

void **powerdelivery_getPowerDataObjectList**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets all Power Data Objects (PDOs). Equivalent to calling PowerDeliveryClass::getPowerDataObject() on all partners, power roles, and index's.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled. On success this value should be 28 (7 rules * 2 partners * 2 power roles)

- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled The order of which is:
  - Rules 1-7 Local Source
  - Rules 1-7 Local Sink
  - Rules 1-7 Partner Source
  - Rules 1-7 Partner Sink.
- **bufferLength** – Length of the buffer to be filed

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).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with enabled state.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

void **powerdelivery_setPowerDataObjectEnabled**(unsigned int *id, struct Result *result, const int index, const unsigned char powerRole, const unsigned char ruleIndex, const unsigned char enabled)

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone 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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 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)
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink

void **powerdelivery_getRequestDataObject**(unsigned int *id, struct Result *result, const int index, const unsigned char partner)

Gets the current Request Data Object (RDO) for a given partner. RDOs: Are provided by the sinking device. 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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the current RDO. Zero indicates the RDO is not active.
- **index** – The index of the entity in question.
- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote

void **powerdelivery_setRequestDataObject**(unsigned int *id, struct Result *result, const int index, const unsigned int rdo)

Sets the current Request Data Object (RDO) for a given partner. (Only the local partner can be changed.) RDOs: Are provided by the sinking device. 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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **rdo** – Request Data Object to be set.

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).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filed with the power role
  - Disabled = 0 = powerdeliveryPowerRoleDisabled
  - Source = 1= powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
  - Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)
- **index** – The index of the entity in question.

void **powerdelivery_setPowerRole**(unsigned int *id, struct Result *result, const int index, const unsigned char powerRole)

Set the current power role to be advertised by the Local partner. (CC Strapping).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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)

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).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Value to be applied.
  - Disabled = 0 = powerdeliveryPowerRoleDisabled
  - Source = 1= powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **index** – The index of the entity in question.

void **powerdelivery_setPowerRolePreferred**(unsigned int *id, struct Result *result, const int index, const unsigned char powerRole)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 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)
- **index** – The index of the entity in question.

void **powerdelivery_getCableCurrentMax**(unsigned int *id, struct Result *result, const int index)

Gets the maximum current capability report by the e-mark of the attached cable.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of current.
  - Unknown/Unattached (0)
  - 3 Amps (1)
  - 5 Amps (2)
- **index** – The index of the entity in question.

void **powerdelivery_getCableSpeedMax**(unsigned int *id, struct Result *result, const int index)

Gets the maximum data rate capability reported by the e-mark of the attached cable.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 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)
- **index** – The index of the entity in question.

void **powerdelivery_getCableType**(unsigned int *id, struct Result *result, const int index)

Gets the cable type reported by the e-mark of the attached cable.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 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)

- **index** – The index of the entity in question.

void **powerdelivery_getCableOrientation**(unsigned int *id, struct Result *result, const int index)

Gets the current orientation being used for PD communication

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable filled with an enumeration of the orientation.
  - Unconnected (0)
  - CC1 (1)
  - CC2 (2)
- **index** – The index of the entity in question.

void **powerdelivery_request**(unsigned int *id, struct Result *result, const int index, const unsigned char request)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

The 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

Parameters

- **id** – ID assigned through "module_createStem"
- **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)

void **powerdelivery_requestStatus** (unsigned int *id, struct Result *result, const int index)

Gets the status of the last request command sent.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the status
- **index** – The index of the entity in question.

void **powerdelivery_getOverride** (unsigned int *id, struct Result *result, const int index)

Gets the current enabled overrides

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Bit mapped representation of the current override configuration.
- **index** – The index of the entity in question.

void **powerdelivery_setOverride** (unsigned int *id, struct Result *result, const int index, const unsigned int overrides)

Sets the current enabled overrides

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **powerdelivery_resetEntityToFactoryDefaults** (unsigned int *id, struct Result *result, const int index)

Resets the PowerDeliveryClass Entity to it factory default configuration.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the current mode.
  - Disabled (0)
  - Enabled (1)
  - Auto (2) default
- **index** – The index of the entity in question.
- **flag** – Flag/Advertisement to be modified

void **powerdelivery_setFlagMode**(unsigned int *id, struct Result *result, const int index, const unsigned char flag, const unsigned char mode)

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone 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

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. An enumerated value referring to the current configuration.
  - Allowable values are 0 - 4
- **index** – The index of the entity in question.

void **powerdelivery_setPeakCurrentConfiguration** (unsigned int *id, struct Result *result, const int index, const unsigned char configuration)

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. An enumerated value referring to current swap value.
  - 0A (0)
  - 900mA (1)
  - 1.5A (2)
  - 3A (3)
- **index** – The index of the entity in question.

void **powerdelivery_setFastRoleSwapCurrent** (unsigned int *id, struct Result *result, const int index, const unsigned char swapCurrent)

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`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)

void **`powerdelivery_packDataObjectAttributes`** (unsigned int *id, struct Result *result, const int index, unsigned char *attributes, const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex)

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

aErrNone on success; aErrParam with bad input.

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`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.

void **`powerdelivery_unpackDataObjectAttributes`** (unsigned int *id, struct Result *result, const int index, const unsigned char attributes, unsigned char *partner, unsigned char *powerRole)

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

aErrNone on success; aErrParam with bad input.

Parameters

- **`id`** – ID assigned through "module_createStem"

- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Data object index.
- **`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

3.7.11 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. It has the ability to take current and voltage measurements, and depending on hardware, may have additional modes and capabilities.

void **`rail_getCurrent`**(unsigned int *id, struct Result *result, const int index)

Get the rail current.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current in micro-amps (1 == 1e-6A).
- **`index`** – The index of the entity in question.

void **`rail_setCurrentSetpoint`**(unsigned int *id, struct Result *result, const int index, const int microamps)

Set the rail supply current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current 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.
- **index** – The index of the entity in question.

void **rail_setCurrentLimit**(unsigned int *id, struct Result *result, const int index, const int microamps)

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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).

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)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current in micro-amps (1 == 1e-6A).
- **index** – The index of the entity in question.

void **rail_getTemperature**(unsigned int *id, struct Result *result, const int index)

Get the rail temperature.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The 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.
- **index** – The index of the entity in question.

void **rail_getEnable**(unsigned int *id, struct Result *result, const int index)

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage
- **index** – The index of the entity in question.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The 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 to determine if this is a measured or stored value.
- **index** – The index of the entity in question.

void **rail_setVoltageSetpoint**(unsigned int *id, struct Result *result, const int index, const int microvolts)

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The 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 to determine if this is a measured or stored value.
- **index** – The index of the entity in question.

void **rail_setVoltageMinLimit**(unsigned int *id, struct Result *result, const int index, const int microvolts)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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).

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)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The voltage in micro-volts (1 == 1e-6V).
- **index** – The index of the entity in question.

void **rail_setVoltageMaxLimit**(unsigned int *id, struct Result *result, const int index, const int microvolts)

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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).

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)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The voltage in micro-volts (1 == 1e-6V).
- **index** – The index of the entity in question.

void **rail_getPower**(unsigned int *id, struct Result *result, const int index)

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The 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 to determine if this is a measured or stored value.
- **index** – The index of the entity in question.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The 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.
- **index** – The index of the entity in question.

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)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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).

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)

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The power in milli-watts (mW).
- **index** – The index of the entity in question.

void **rail_getResistance** (unsigned int *id, struct Result *result, const int index)

Get the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The 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.
- **index** – The index of the entity in question.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The 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 to determine if this is a measured or stored value.
- **index** – The index of the entity in question.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Kelvin sensing is enabled or disabled.
- **index** – The index of the entity in question.

void **rail_getKelvinSensingState**(unsigned int *id, struct Result *result, const int index)

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Kelvin sensing is enabled or disabled.
- **index** – The index of the entity in question.

void **rail_setOperationalMode**(unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current operational mode setting.
- **index** – The index of the entity in question.

void **rail_getOperationalState**(unsigned int *id, struct Result *result, const int index)

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current operational state, hardware configuration, faults, and operating mode.
- **index** – The index of the entity in question.

void **rail_clearFaults**(unsigned int *id, struct Result *result, const int index)

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

3.7.12 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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **rcservo_getEnable**(unsigned int *id, struct Result *result, const int index)

Get the enable status of the servo channel.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current enable status of the servo entity. 0 is disabled, 1 is enabled.
- **index** – The index of the entity in question.

void **rcservo_setPosition**(unsigned int *id, struct Result *result, const int index, const unsigned char position)

Set the position of the servo channel

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **rcservo_getPosition**(unsigned int *id, struct Result *result, const int index)

Get the position of the servo channel

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current position of the servo channel. Default 64 = a 1ms pulse and 192 = a 2ms pulse.
- **index** – The index of the entity in question.

void **rcservo_setReverse** (unsigned int *id, struct Result *result, const int index, const unsigned char reverse)

Set the output to be reversed on the servo channel

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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, "getPostion" will return the set value of 64. 0 = not reversed, 1 = reversed.

void **rcservo_getReverse** (unsigned int *id, struct Result *result, const int index)

Get the reverse status of the servo channel

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current reverse status of the servo entity. 0 = not reversed, 1 = reversed.
- **index** – The index of the entity in question.

3.7.13 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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

void **relay_getEnable**(unsigned int *id, struct Result *result, const int index)

Get the state.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. False or 0 = Disabled, True or 1 = Enabled
- **index** – The index of the entity in question.

void **relay_getVoltage**(unsigned int *id, struct Result *result, const int index)

Get the scaled micro volt value with reference to ground.

Returns common entity return values

Note: Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. 32 bit signed integer (in micro Volts) based on the boards ground and reference voltages.
- **index** – The index of the entity in question.

3.7.14 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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

void **signal_getEnable**(unsigned int *id, struct Result *result, const int index)

Get the Enable/Disable of the signal.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. True to enable, false to disable
- **index** – The index of the entity in question.

void **signal_setInvert**(unsigned int *id, struct Result *result, const int index, const unsigned char invert)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. to invert, false for normal mode.
- **index** – The index of the entity in question.

void **signal_setT3Time**(unsigned int *id, struct Result *result, const int index, const unsigned int t3_nsec)

Set the signal period or T3 in nanoseconds.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **signal_getT3Time**(unsigned int *id, struct Result *result, const int index)

Get the signal period or T3 in nanoseconds.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.
- **index** – The index of the entity in question.

void **signal_setT2Time**(unsigned int *id, struct Result *result, const int index, const unsigned int t2_nsec)

Set the signal active period or T2 in nanoseconds.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **signal_getT2Time**(unsigned int *id, struct Result *result, const int index)

Get the signal active period or T2 in nanoseconds.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.
- **index** – The index of the entity in question.

3.7.15 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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. true: enabled, false: disabled.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_loadSlot**(unsigned int *id, struct Result *result, const int index, const unsigned char slot, unsigned char *buffer, const unsigned short bufferLength)

Load the slot.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **buffer** – The data.
- **bufferLength** – The data length.

void **store_unloadSlot**(unsigned int *id, struct Result *result, const int index, const unsigned char slot, unsigned char *buffer, const unsigned int bufferLength)

Unload the slot data.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone on success. Non-zero error code on failure. Length of data that was unloaded. Unloaded length will never be larger than dataLength.
- **index** – The index of the entity in question.
- **buffer** – Byte array that the unloaded data will be placed into.
- **bufferLength** – - The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **slot** – The slot number.

void **store_slotEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Enable slot.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_slotDisable**(unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Disable slot.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_getSlotCapacity**(unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The slot capacity.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_getSlotSize**(unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The slot size.
- **index** – The index of the entity in question.
- **slot** – The slot number.

void **store_getSlotLocked**(unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Gets the current lock state of the slot Allows for write protection on a slot.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filed with the locked state.
- **index** – The index of the entity in question.
- **slot** – The slot number

void **store_setSlotLocked**(unsigned int *id, struct Result *result, const int index, const unsigned char slot, const unsigned char lock)

Sets the locked state of the slot Allows for write protection on a slot.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone 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.

3.7.16 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.

void **system_getModule**(unsigned int *id, struct Result *result, const int index)

Get the current address the module uses on the BrainStem network.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The address the module is using on the BrainStem network.
- **index** – The index of the entity in question.

void **system_getModuleBaseAddress**(unsigned int *id, struct Result *result, const int index)

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The address the module is using on the BrainStem network.
- **index** – The index of the entity in question.

void **system_setRouter**(unsigned int *id, struct Result *result, const int index, const unsigned char address)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to "drop off" the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **address** – The router address to be used.

void **system_getRouter**(unsigned int *id, struct Result *result, const int index)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The address.
- **index** – The index of the entity in question.

void **system_setHBInterval**(unsigned int *id, struct Result *result, const int index, const unsigned char interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **interval** – The desired heartbeat delay.

void **system_getHBInterval**(unsigned int *id, struct Result *result, const int index)

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current heartbeat delay.
- **index** – The index of the entity in question.

void **system_setLED**(unsigned int *id, struct Result *result, const int index, 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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bOn** – true: turn the LED on, false: turn LED off.

void **system_getLED**(unsigned int *id, struct Result *result, const int index)

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. true: LED on, false: LED off.
- **index** – The index of the entity in question.

void **system_setLEDMaxBrightness**(unsigned int *id, struct Result *result, const int index, const unsigned char brightness)

Sets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum). The colors of each LED may be inconsistent at low brightness levels. Note that if the brightness is set to zero and the settings are saved, then the LEDs will no longer indicate whether the system is powered on. When troubleshooting, the user configuration may need to be manually reset in order to view the LEDs again.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **brightness** – Brightness value relative to 255

void **system_getLEDMaxBrightness**(unsigned int *id, struct Result *result, const int index)

Gets the scaling factor for the brightness of all LEDs on the system. The brightness is set to the ratio of this value compared to 255 (maximum).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Brightness value relative to 255
- **index** – The index of the entity in question.

void **system_setBootSlot**(unsigned int *id, struct Result *result, const int index, const unsigned char slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number in aSTORE_INTERNAL to be marked as a boot slot.

void **system_getBootSlot**(unsigned int *id, struct Result *result, const int index)

Get the store slot which is mapped when the module boots.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The slot number in aSTORE_INTERNAL that is mapped after the module boots.
- **index** – The index of the entity in question.

void **system_getVersion**(unsigned int *id, struct Result *result, const int index)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The build version date code.
- **index** – The index of the entity in question.

void **system_getBuild**(unsigned int *id, struct Result *result, const int index)

Get the modules firmware build number The build number is a unique hash assigned to a specific firmware.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with build.
- **index** – The index of the entity in question.

void **system_getModel**(unsigned int *id, struct Result *result, const int index)

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's model enumeration.
- **index** – The index of the entity in question.

void **system_getHardwareVersion**(unsigned int *id, struct Result *result, const int index)

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's hardware version information.
- **index** – The index of the entity in question.

void **system_getSerialNumber**(unsigned int *id, struct Result *result, const int index)

Get the module's serial number. The serial number is a unique 32bit integer which is usually communicated in hexadecimal format.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's serial number.
- **index** – The index of the entity in question.

void **system_save**(unsigned int *id, struct Result *result, const int index)

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **system_reset**(unsigned int *id, struct Result *result, const int index)

Reset the system. aErrTimeout indicates a successful reset, as the system resets immediately, which tears down the USB-link immediately, thus preventing an affirmative response.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **system_logEvents**(unsigned int *id, struct Result *result, const int index)

Saves system log events to a slot defined by the module (usually ram slot 0).

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **system_getUptime**(unsigned int *id, struct Result *result, const int index)

Get the module's accumulated uptime in minutes

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's accumulated uptime in minutes.
- **index** – The index of the entity in question.

void **system_getTemperature**(unsigned int *id, struct Result *result, const int index)

Get the module's current temperature in micro-C

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's system temperature in micro-C
- **index** – The index of the entity in question.

void **system_getMinimumTemperature**(unsigned int *id, struct Result *result, const int index)

Get the module's minimum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's minimum system temperature in micro-C
- **index** – The index of the entity in question.

void **system_getMaximumTemperature**(unsigned int *id, struct Result *result, const int index)

Get the module's maximum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's maximum system temperature in micro-C
- **index** – The index of the entity in question.

void **system_getInputVoltage**(unsigned int *id, struct Result *result, const int index)

Get the module's input voltage.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's input voltage reported in microvolts.
- **index** – The index of the entity in question.

void **system_getInputCurrent**(unsigned int *id, struct Result *result, const int index)

Get the module's input current.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's input current reported in microamps.
- **index** – The index of the entity in question.

void **system_getModuleHardwareOffset**(unsigned int *id, struct Result *result, const int index)

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module address offset.
- **index** – The index of the entity in question.

void **system_setModuleSoftwareOffset**(unsigned int *id, struct Result *result, const int index, const unsigned char address)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **address** – The address for the module. Value must be even from 0-254.

void **system_getModuleSoftwareOffset**(unsigned int *id, struct Result *result, const int index)

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The address for the module. Value must be even from 0-254.
- **index** – The index of the entity in question.

void **system_getRouterAddressSetting**(unsigned int *id, struct Result *result, const int index)

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The address for the module. Value must be even from 0-254.
- **index** – The index of the entity in question.

void **system_routeToMe**(unsigned int *id, struct Result *result, const int index, 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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bOn** – Enable or disable of the route to me function 1 = enable.

void **system_getPowerLimit**(unsigned int *id, struct Result *result, const int index)

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The available power in milli-Watts (mW, 1 t)
- **index** – The index of the entity in question.

void **system_getPowerLimitMax**(unsigned int *id, struct Result *result, const int index)

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the power limit in milli-Watts (mW)
- **index** – The index of the entity in question.

void **system_setPowerLimitMax**(unsigned int *id, struct Result *result, const int index, const unsigned int power)

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **power** – Limit in milli-Watts (mW) to be set.

void **system_getPowerLimitState** (unsigned int *id, struct Result *result, const int index)

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit().

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the state.
- **index** – The index of the entity in question.

void **system_getUnregulatedVoltage** (unsigned int *id, struct Result *result, const int index)

Gets the voltage present at the unregulated port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the voltage in micro-Volts (uV).
- **index** – The index of the entity in question.

void **system_getUnregulatedCurrent** (unsigned int *id, struct Result *result, const int index)

Gets the current passing through the unregulated port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the current in micro-Amps (uA).
- **index** – The index of the entity in question.

void **system_getInputPowerSource** (unsigned int *id, struct Result *result, const int index)

Provides the source of the current power source in use.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with enumerated representation of the source.
- **index** – The index of the entity in question.

void **system_getInputPowerBehavior**(unsigned int *id, struct Result *result, const int index)

Gets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated value representing behavior.
- **index** – The index of the entity in question.

void **system_setInputPowerBehavior**(unsigned int *id, struct Result *result, const int index, const unsigned char behavior)

Sets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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 to be set.

void **system_getInputPowerBehaviorConfig**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **system_setInputPowerBehaviorConfig**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.
- **bufferLength** – Length of the buffer to be transferred.

void **system_getName**(unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **system_setName**(unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const unsigned int bufferLength)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.
- **bufferLength** – Length of the buffer to be transferred.

void **system_resetEntityToFactoryDefaults**(unsigned int *id, struct Result *result, const int index)

Resets the SystemClass Entity to it factory default configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **system_resetDeviceToFactoryDefaults**(unsigned int *id, struct Result *result, const int index)

Resets the device to it factory default configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **system_getLinkInterface**(unsigned int *id, struct Result *result, const int index)

Gets the link interface configuration. This refers to which interface is being used for control by the device.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated value representing interface.
  - 0 = Auto= systemLinkAuto
  - 1 = Control Port = systemLinkUSBControl
  - 2 = Hub Upstream Port = systemLinkUSBHub
- **index** – The index of the entity in question.

void **system_setLinkInterface**(unsigned int *id, struct Result *result, const int index, const unsigned char linkInterface)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **linkInterface** – An enumerated representation of interface to be set.
  - 0 = Auto= systemLinkAuto
  - 1 = Control Port = systemLinkUSBControl
  - 2 = Hub Upstream Port = systemLinkUSBHub

void **system_getErrors**(unsigned int *id, struct Result *result, const int index)

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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Bit mapped field representing the devices errors
- **index** – The index of the entity in question.

3.7.17 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

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The temperature in micro-Celsius (1 == 1e-6C).
- **index** – The index of the entity in question.

void **temperature_getValueMin**(unsigned int *id, struct Result *result, const int index)

Get the module's minimum temperature in micro-C since the last power cycle.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's minimum temperature in micro-C
- **index** – The index of the entity in question.

void **temperature_getValueMax** (unsigned int *id, struct Result *result, const int index)

Get the module's maximum temperature in micro-C since the last power cycle.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The module's maximum temperature in micro-C
- **index** – The index of the entity in question.

void **temperature_resetEntityToFactoryDefaults** (unsigned int *id, struct Result *result, const int index)

Resets the TemperatureClass Entity to it factory default configuration.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

3.7.18 Timer Entity

group **TimerEntity**

TimerClass: The Timer Class provides access to a simple scheduler. The timer can set to fire only once, or to repeat at a certain interval. Additionally, a timer entity can execute custom Reflex routines upon firing.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The timer expiration duration in microseconds.
- **index** – The index of the entity in question.

void **timer_setExpiration** (unsigned int *id, struct Result *result, const int index, const unsigned int usecDuration)

Set the expiration time for the timer entity. When the timer expires, it will fire the associated timer[index]() reflex.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The mode of the time. aTIMER_MODE_REPEAT or aTIMER_MODE_SINGLE.
- **index** – The index of the entity in question.

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.

Returns common entity return values

aErrNone Action completed successfully.

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

3.7.19 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 bEnabled)

Enable the UART channel.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnabled** – true: enabled, false: disabled.

void **uart_getEnable**(unsigned int *id, struct Result *result, const int index)

Get the enabled state of the uart.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. true: enabled, false: disabled.
- **index** – The index of the entity in question.

void **uart_setBaudRate**(unsigned int *id, struct Result *result, const int index, const unsigned int rate)

Set the UART baud rate.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **rate** – baud rate.

void **uart_getBaudRate**(unsigned int *id, struct Result *result, const int index)

Get the UART baud rate.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Pointer variable to be filled with baud rate.
- **index** – The index of the entity in question.

void **uart_setProtocol**(unsigned int *id, struct Result *result, const int index, const unsigned char protocol)

Set the UART protocol.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **protocol** – An enumeration of serial protocols.

void **uart_getProtocol**(unsigned int *id, struct Result *result, const int index)

Get the UART protocol.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Pointer to where result is placed.
- **index** – The index of the entity in question.

3.7.20 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 int index, const unsigned char channel)

Enable both power and data lines for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setPortDisable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable both power and data lines for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setDataEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setDataDisable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setHiSpeedDataEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setHiSpeedDataDisable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setSuperSpeedDataEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setSuperSpeedDataDisable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setPowerEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Enable only the power line for a port without changing the state of the data lines.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setPowerDisable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Disable only the power line for a port without changing the state of the data lines.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getPortCurrent**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current through the power line for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel current in micro-amps (1 == 1e-6A).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getPortVoltage**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage on the power line for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel voltage in microvolts (1 == 1e-6V).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getHubMode**(unsigned int *id, struct Result *result, const int index)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB hub mode.
- **index** – The index of the entity in question.

void **`usb_setHubMode`**(unsigned int *id, struct Result *result, const int index, const unsigned int mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone on success. Non-zero error code on failure.
- **`index`** – The index of the entity in question.
- **`mode`** – The USB hub mode.

void **`usb_clearPortErrorStatus`**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Clear the error status for the given port.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone on success. Non-zero error code on failure.
- **`index`** – The index of the entity in question.
- **`channel`** – The port to clear error status for.

void **`usb_getUpstreamMode`**(unsigned int *id, struct Result *result, const int index)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The Upstream port mode.
- **`index`** – The index of the entity in question.

void **`usb_setUpstreamMode`**(unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0, usbUpstreamModePort1, and usbUpstreamModeNone.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone on success. Non-zero error code on failure.
- **`index`** – The index of the entity in question.

- **mode** – The Upstream port mode.

void **usb_getUpstreamState** (unsigned int *id, struct Result *result, const int index)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The Upstream port state.
- **index** – The index of the entity in question.

void **usb_setEnumerationDelay** (unsigned int *id, struct Result *result, const int index, const unsigned int ms_delay)

Set the inter-port enumeration delay in milliseconds.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **ms_delay** – Millisecond delay in 100mS increments (100, 200, 300 etc.)

void **usb_getEnumerationDelay** (unsigned int *id, struct Result *result, const int index)

Get the inter-port enumeration delay in milliseconds.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Millisecond delay in 100mS increments (100, 200, 300 etc.)
- **index** – The index of the entity in question.

void **usb_setPortCurrentLimit** (unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned int microamps)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a stem.system.save() call.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

void **usb_getPortCurrentLimit**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current limit for the port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current limit setting.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel to limit.

void **usb_setPortMode**(unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned int mode)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices use a common bit mapping for port mode at usbPortMode

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – USB downstream channel to set the mode on.
- **mode** – The port mode setting as packed bit field.

void **usb_getPortMode**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device reference page for a complete list of capabilities. Some devices implement a common bit mapping for port mode at usbPortMode

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care
- **index** – The index of the entity in question.
- **channel** – USB downstream channel.

void **usb_getPortState**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current State for the Port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care
- **index** – The index of the entity in question.
- **channel** – USB downstream channel.

void **usb_getPortError**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current error for the Port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care
- **index** – The index of the entity in question.
- **channel** – USB downstream channel.

void **usb_setUpstreamBoostMode**(unsigned int *id, struct Result *result, const int index, const unsigned char setting)

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **setting** – Upstream boost setting 0, 1, 2, or 3.

void **usb_setDownstreamBoostMode**(unsigned int *id, struct Result *result, const int index, const unsigned char setting)

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **setting** – Downstream boost setting 0, 1, 2, or 3.

void **usb_getUpstreamBoostMode**(unsigned int *id, struct Result *result, const int index)

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current Upstream boost setting 0, 1, 2, or 3.
- **index** – The index of the entity in question.

void **usb_getDownstreamBoostMode**(unsigned int *id, struct Result *result, const int index)

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current Downstream boost setting 0, 1, 2, or 3.
- **index** – The index of the entity in question.

void **`usb_getDownstreamDataSpeed`**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Filled with the current port data speed
  - N/A: usbDownstreamDataSpeed_na = 0
  - Hi Speed: usbDownstreamDataSpeed_hs = 1
  - SuperSpeed: usbDownstreamDataSpeed_ss = 2
- **`index`** – The index of the entity in question.
- **`channel`** – USB downstream channel to check.

void **`usb_setConnectMode`**(unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char mode)

Sets the connect mode of the switch.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone on success. Non-zero error code on failure.
- **`index`** – The index of the entity in question.
- **`channel`** – The USB sub channel.
- **`mode`** – The connect mode
  - usbManualConnect = 0
  - usbAutoConnect = 1

void **`usb_getConnectMode`**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Gets the connect mode of the switch.

Returns common entity return values

Parameters

- **`id`** – ID assigned through "module_createStem"
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current connect mode
- **`index`** – The index of the entity in question.
- **`channel`** – The USB sub channel.

void **usb_setCC1Enable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char bEnable)

Set Enable/Disable on the CC1 line.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – USB channel.
- **bEnable** – State to be set
  - Disabled: 0
  - Enabled: 1

void **usb_getCC1Enable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get Enable/Disable on the CC1 line.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. State to be filled
  - Disabled: 0
  - Enabled: 1
- **index** – The index of the entity in question.
- **channel** – USB channel.

void **usb_setCC2Enable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char bEnable)

Set Enable/Disable on the CC2 line.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – USB channel.
- **bEnable** – State to be filled
  - Disabled: 0

– Enabled: 1

void **usb_getCC2Enable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get Enable/Disable on the CC1 line.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
  - State to be filled
  - Disabled: 0
  - Enabled: 1
- **index** – The index of the entity in question.
- **channel** – - USB channel.

void **usb_getCC1Current**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current through the CC1 for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel current in micro-amps (1 == 1e-6A).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getCC2Current**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the current through the CC2 for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel current in micro-amps (1 == 1e-6A).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getCC1Voltage**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of CC1 for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel voltage in micro-volts (1 == 1e-6V).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getCC2Voltage**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of CC2 for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel voltage in micro-volts (1 == 1e-6V).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setSBUEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char bEnable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.
- **bEnable** – The state to be set
  - Disabled: 0
  - Enabled: 1

void **usb_getSBUEnable**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the Enable/Disable status of the SBU

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The enable/disable status of the SBU
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setCableFlip**(unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned char bEnable)

Set Cable flip. This will flip SBU, CC and SS data lines.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.
- **bEnable** – The state to be set The state to be set
  - Disabled: 0
  - Enabled: 1

void **usb_getCableFlip**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get Cable flip setting.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The enable/disable status of cable flip.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_setAltModeConfig**(unsigned int *id, struct Result *result, const int index, const unsigned char channel, const unsigned int configuration)

Set USB Alt Mode Configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

void **usb_getAltModeConfig**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get USB Alt Mode Configuration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB configuration for the given channel.
- **index** – The index of the entity in question.
- **channel** – The USB sub channel

void **usb_getSBU1Voltage**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of SBU1 for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel voltage in micro-volts (1 == 1e-6V).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

void **usb_getSBU2Voltage**(unsigned int *id, struct Result *result, const int index, const unsigned char channel)

Get the voltage of SBU2 for a port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The USB channel voltage in micro-volts (1 == 1e-6V).
- **index** – The index of the entity in question.
- **channel** – The USB sub channel.

3.7.21 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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.

void **usbsystem_setUpstream**(unsigned int *id, struct Result *result, const int index, const unsigned char port)

Sets the upstream port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. the current inter-port delay in milliseconds.
- **index** – The index of the entity in question.

void **usbsystem_setEnumerationDelay**(unsigned int *id, struct Result *result, const int index, const unsigned int msDelay)

Sets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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

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.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. A bit packed representation of the data role for all ports.
- **index** – The index of the entity in question.

void **usbsystem_getEnabledList**(unsigned int *id, struct Result *result, const int index)

Gets the current enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Bit packed representation of the enabled status for all ports.
- **index** – The index of the entity in question.

void **usbsystem_setEnabledList**(unsigned int *id, struct Result *result, const int index, const unsigned int enabledList)

Sets the enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **usbsystem_getModeList**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the current mode of all ports with a single call. Equivalent to calling PortClass:getMode() on each port.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **usbsystem_setModeList**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **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.
- **bufferLength** – Length of the buffer to be transferred.

void **usbsystem_getStateList**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **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.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.
- **index** – The index of the entity in question.

void **usbsystem_setPowerBehavior**(unsigned int *id, struct Result *result, const int index, const unsigned char behavior)

Sets the behavior of how available power is managed. i.e. What happens when requested power is greater than available power.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **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.

void **usbsystem_getPowerBehaviorConfig**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

Returns common entity return values

Parameters

- **id** – ID assigned through “module_createStem”
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **usbsystem_setPowerBehaviorConfig**(unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Sets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.
- **bufferLength** – Length of the buffer to be transferred.

void **usbsystem_getDataRoleBehavior** (unsigned int *id, struct Result *result, const int index)

Gets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.
- **index** – The index of the entity in question.

void **usbsystem_setDataRoleBehavior** (unsigned int *id, struct Result *result, const int index, const unsigned char behavior)

Sets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **usbsystem_getDataRoleBehaviorConfig** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Gets the current data role behavior configuration Certain data role behaviors use a list of ports to determine priority host priority.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Length that was actually received and filled.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufferLength** – Length of the buffer to be filed

void **usbsystem_setDataRoleBehaviorConfig** (unsigned int *id, struct Result *result, const int index, unsigned int *buffer, const unsigned int bufferLength)

Sets the current data role behavior configuration Certain data role behaviors use a list of ports to determine host priority.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.
- **bufferLength** – Length of the buffer to be transferred.

void **usbsystem_getSelectorMode** (unsigned int *id, struct Result *result, const int index)

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Variable to be filled with the selector mode
- **index** – The index of the entity in question.

void **usbsystem_setSelectorMode** (unsigned int *id, struct Result *result, const int index, const unsigned char mode)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **mode** – Mode to be set.

void **usbsystem_resetEntityToFactoryDefaults**(unsigned int *id, struct Result *result, const int index)

Resets the USBSystemClass Entity to it factory default configuration.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

void **usbsystem_getUpstreamHS**(unsigned int *id, struct Result *result, const int index)

Gets the USB HighSpeed upstream port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.

void **usbsystem_setUpstreamHS**(unsigned int *id, struct Result *result, const int index, const unsigned char port)

Sets the USB HighSpeed upstream port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **usbsystem_getUpstreamSS**(unsigned int *id, struct Result *result, const int index)

Gets the USB SuperSpeed upstream port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.

void **usbsystem_setUpstreamSS** (unsigned int *id, struct Result *result, const int index, const unsigned char port)

Sets the USB SuperSpeed upstream port.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **usbsystem_getOverride** (unsigned int *id, struct Result *result, const int index)

Gets the current enabled overrides

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Bit mapped representation of the current override configuration.
- **index** – The index of the entity in question.

void **usbsystem_setOverride** (unsigned int *id, struct Result *result, const int index, const unsigned int overrides)

Sets the current enabled overrides

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **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.

void **usbsystem_setDataHSMaxDatarate** (unsigned int *id, struct Result *result, const int index, const unsigned int datarate)

Sets the USB HighSpeed Max datarate

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **datarate** – Maximum datarate for the USB HighSpeed signals.

void **usbsystem_getDataHSMaxDatarate** (unsigned int *id, struct Result *result, const int index)

Gets the USB HighSpeed Max datarate

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Current maximum datarate for the USB HighSpeed signals.
- **index** – The index of the entity in question.

void **usbsystem_setDataSSMaxDatarate** (unsigned int *id, struct Result *result, const int index, const unsigned int datarate)

Sets the USB SuperSpeed Max datarate

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **datarate** – Maximum datarate for the USB SuperSpeed signals.

void **usbsystem_getDataSSMaxDatarate** (unsigned int *id, struct Result *result, const int index)

Gets the USB SuperSpeed Max datarate

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. Current maximum datarate for the USB SuperSpeed signals.
- **index** – The index of the entity in question.

3.7.22 Module Entity

group **ModuleEntity**

The Module Entity provides a generic interface to a BrainStem hardware module. The Module Class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

void **module_createStem**(unsigned int *id, struct Result *result, unsigned char moduleAddress, bool autoNetworking, unsigned char model)

Creates a brainstem object that the library will manage internally and creates a unique identifier that will be used for other function calls in this library.

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_disconnectAndDestroyStem**(unsigned int *id, struct Result *result)

Disconnects from device defined by the ID and will destroy any internal memory associated with the Device.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_discoverAndConnect**(unsigned int *id, struct Result *result, int transport, unsigned int serialNumber)

Finds and connects to the first device found on the given transport. If a serial number was provided when module_createStem was called then it will only connect to that specific id.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR or a non zero Error code.
- **transport** – Defines what connection method should be searched for BrainStem devices. (i.e. USB, TCPIP, etc.)

void **module_sDiscover**(unsigned int *id, struct Result *result, struct *linkSpec_CCA* *stemList, int listLength, int transport)

Discovers all of the BrainStem devices on a given transport. The return list is filled with device specifiers which contains information about the device.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR or a non zero Error code.
- **stemList** – List of device specifiers for each of the discovered devices
- **listLength** – Indicates how long the list is.
- **transport** – Defines what connection method should be searched for BrainStem devices. (i.e. USB, TCPIP, etc.)

void **module_disconnect**(unsigned int *id, struct Result *result)

Disconnects device, but does not destroy underlying object. i.e. "module_reconnect" could be called without calling "module_createStem" again.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_reconnect**(unsigned int *id, struct Result *result)

Reestablishes a connection with a preexisting stem object. The original stem would of been created with "module_createStem".

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_connectThroughLinkModule**(unsigned int *id, unsigned int *id_linkStem, struct Result *result)

Establishes connection through another stems link/connection (i.e. transport: USB, TCPIP). Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through "module_createStem"
- **id_linkStem** – The link stem's id assigned through "module_createStem" (The stem providing the connection.)
- **result** – object, containing NO_ERROR or a non zero Error code.

void **module_setModuleAddress**(unsigned int *id, struct Result *result, int address)

Changes the module address of the stem object that was created via "module_createStem". Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR or a non zero Error code.
- **address** – New address to be set.

void **module_getModuleAddress**(unsigned int *id, struct Result *result)

Retrieves the module address of the stem object that was created via "module_createStem". Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR and the module/stems current module address or a non zero Error code.

void **module_isConnected**(unsigned int *id, struct Result *result)

Returns the current state of the module/stem's connection. Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR and the status of the connection or a non zero Error code. (0 = disconnected; 1 = connected.)

void **module_setNetworkingMode**(unsigned int *id, struct Result *result, int mode)

Changes the networking mode of the stem object. Auto mode is enabled by default which allows automatic adjustment of the module/stems networking configuration. Refer to BrainStem Networking at www.acroname.com/support

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – object, containing NO_ERROR or a non zero Error code.
- **mode** – New mode to be set.

void **module_clearAllStems**()

Disconnects and destroys all modules/stems. During development the dll doesn't always "detach" between runs. This can create problematic scenarios if there are not subsequent disconnect and destroy calls for every create call made. This function can be a helpful post-execution/tear-down process when bringing up new BrainStem Networks. However, not required if connections are handles correctly. Refer to BrainStem Networking at www.acroname.com/support

3.7.23 Link Entity

group **LinkEntity**

The Link class provides link level access to streaming information.

struct **linkSpec_CCA**

CCA linkSpec structure - It contains the necessary information for connecting to a BrainStem module.

Public Members

unsigned int **type**

The transport type of this spec.

unsigned int **serial_num**

The serial number of the module

unsigned int **module**

The module address

unsigned int **router**

The BrainStem network router address

unsigned int **router_serial_num**

The BrainStem network router serial number

unsigned int **model**

The model type

unsigned int **usb_id**

The usb_id of the BrainStem module.

unsigned int **ip_address**

The IP4 address of the module.

unsigned int **ip_port**

The TCP port for socket connection on the module.

unsigned int **baudrate**

The serial port baudrate

char **port**[100]

The serial port path or name

struct **StreamStatusEntry_CCA**

StreamStatusEntry structure - It contains members of streaming entries in the form of key value pairs. Keys are comprised of the devices module address, command, option, index, and subindex API values.

Public Members

unsigned long long **key**

The stream key (64bit).

unsigned int **value**

The value associated with the key (32bit).

void **link_enableStream**(unsigned int *id, struct Result *result, const unsigned char moduleAddress, const unsigned char cmd, const unsigned char option, const unsigned char index, const bool enable)

Enables streaming for the supplied criteria.

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **moduleAddress** – Address to filter on.
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - Enables streaming; False - disables streaming

void **link_getLinkSpecifier**(unsigned int *id, struct Result *result, struct *linkSpec_CCA* *spec)

Get linkSpecifier

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **spec** – - allocated linkspec struct will be filled with spec.

void **link_registerStreamCallback**(unsigned int *id, struct Result *result, const unsigned char moduleAddress, const unsigned char cmd, const unsigned char option, const unsigned char index, const bool enable, cStreamCallback_t cb, void *pRef)

Registers a callback function based on a specific module, cmd, option, and index.

::aErrNotFound - Item not found (uninstalling only)

::aErrNone - success

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - installs/updates callback and ref; False - uninstalls callback
- **cb** – Callback to be executed when a new packet matching the criteria is received.
- **pRef** – Pointer to user reference for use inside the callback function.

void **link_getStreamStatus**(unsigned int *id, struct Result *result, const unsigned char moduleAddress, const unsigned char cmd, const unsigned char option, const unsigned char index, const unsigned char subindex, struct *StreamStatusEntry_CCA* *buffer, const unsigned int bufferLength)

Gets all available stream values based on the search criteria.

::aErrParam if status or unloadedSize is null

::aErrNone - success

Parameters

- **id** – Unique identifier for the internally created stem.
- **result** – object, containing NO_ERROR or a non zero Error code.
- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **subindex** – subindex to filter by (supports Wildcards)
- **buffer** – Buffer of user allocated memory to be filled with stream data Note: Link::getStreamKeyElement should be used to decode the keys
- **bufferLength** – Number of elements the buffer can hold.

void **link_getStreamKeyElement**(struct Result *result, const unsigned long long key, const unsigned char element)

Convenience function to unpack a stream key.

Parameters

- **key** – The key to be unpacked
- **element** – The element to unpack from the key.

3.7.24 PDChannelLogger

group **PDChannelLogger**

PDChannelLogger: Provides an interface for managing BrainStem Power Delivery Packets. Packets are accepted and decoded asynchronously. Pay careful attention to packet cleanup. Portions of the structure do not belong to the caller.

struct **BS_PD_Packet_CCA**

BrainStem Power Delivery Packet Structure - Contains information representing a Power Delivery packet along with contextual device information

Public Members

unsigned char **channel**

Channel/Index

unsigned int **seconds**

Seconds in device time since power on.

unsigned int **uSeconds**

Micro Seconds in device time since power on .

unsigned char **direction**

Direction of packet transmission relative to the device.

unsigned char **sop**

See bs_pd_packet.h for more details

unsigned int **event**

Packet type - See powerdeliveryLogEvent in aProtocolDefs.h

unsigned int **payloadSize**

Length of the payload.

unsigned char ***payload**

Raw PD Packet data

void **PDChannelLogger_create**(unsigned int *id, struct Result *result, const int index, const unsigned int bufferLength)

Creates internal object for managing BrainStem Power Delivery logging packets.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.
- **bufferLength** – Number of packets the class should queue before dropping.

void **PDChannelLogger_destroy**(unsigned int *id, struct Result *result, const int index)

Destroys internal object for managing BrainStem Power Delivery logging packets.

Returns common entity return values

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.

void **PDChannelLogger_setEnabled**(unsigned int *id, struct Result *result, const int index, const bool enable)

Enables Power Delivery logging.

True on success.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.
- **enable** – True enables logging; False disables logging

void **PDChannelLogger_getPacket**(unsigned int *id, struct Result *result, const int index, struct *BS_PD_Packet_CCA* *packet)

Attempts to takes a packet from the internal buffer.

True if the function successfully acquired any number of packets. False if no packets were available.

Parameters

- **id** – ID assigned through "module_createStem"
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **index** – The index of the entity in question.
- **packet** – Reference to a packet to be filled by the function.

void **`PDChannelLogger_getPackets`**(unsigned int *id, struct Result *result, const int index, struct *BS_PD_Packet_CCA* *packetBuffer, const unsigned int bufferLength)

Attempts to take a multiple packets (up to a maximum) from the internal buffer.

True if the function successfully acquired any number of packets. False if no packets were available.

Parameters

- **`id`** – ID assigned through “module_createStem”
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **`index`** – The index of the entity in question.
- **`packetBuffer`** – pointer to a buffer to be filled
- **`bufferLength`** – The length of the buffer provided.

void **`PDChannelLogger_freePayloadBuffer`**(unsigned int *id, struct Result *result, struct *BS_PD_Packet_CCA* *packet)

Releases the internal memory contained within the payload parameter.

Returns common entity return values

Parameters

- **`id`** – ID assigned through “module_createStem”
- **`result`** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The current upstream port.
- **`packet`** – Reference to a packet to have its memory freed.

3.7.25 PortMapping

group **`PortMapping`**

PortMapping Class: Provides an interface for usb descriptor information of devices downstream of Acroname hub products.

struct **`DeviceNode_CCA`**

Device Node Structure - Contains information linking the downstream device to the Acroname Hub.

Public Members

unsigned int **hubSerialNumber**

Serial number of the Acroname hub where the device was found.

unsigned char **hubPort**

Port of the Acroname hub where the device was found.

unsigned short **idVendor**

Manufactures Vendor ID of the downstream device.

unsigned short **idProduct**

Manufactures Product ID of the downstream device.

unsigned char **speed**

The devices downstream device speed. (PORT_SPEED_CCA_t)

char **productName**[255]

USB string descriptor

char **serialNumber**[255]

USB string descriptor

char **manufacturer**[255]

USB string descriptor

void **portMapping_getDownstreamDevices**(struct Result *result, struct *DeviceNode_CCA* *buffer, const unsigned int bufferLength)

Gets downstream device USB information for all Acroname hubs.

::aErrNone on success

::aErrParam: Passed in values are not valid. (NULL, size etc).

::aErrMemory: No more room in the list

::aErrNotFound: No Acroname devices were found.

Parameters

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure. The number of devices found.
- **buffer** – Pointer to the start of a list/array to be used by the function.
- **bufferLength** – Size of the list/array in *DeviceNode_CCA*'s, not bytes.

3.7.26 Version

group **Version**

Functions for getting and comparing software and firmware version.

void **version_ParseMajor** (struct Result *result, unsigned int build)

Parses the major revision level from the given build number.

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_ParseMinor** (struct Result *result, unsigned int build)

Parses the minor revision level from the given build number.

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_ParsePatch** (struct Result *result, unsigned int build)

Parses the revision patch level from the given build number.

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_IsLegacyFormat** (struct Result *result, unsigned int build)

Check if the given build version is of the legacy packing format

Parameters

- **result** – object, containing NO_ERROR or a non zero Error code.
- **build** – The packed version number returned from the system.getVersion call.

void **version_GetMajor** (struct Result *result)

Return the major revision number for the software package.

Parameters

result – Object containing aErrNone and the requested value on success.

void **version_GetMinor** (struct Result *result)

Return the minor revision number for the software package.

Parameters

result – Object containing aErrNone and the requested value on success.

void **version_GetPatch** (struct Result *result)

Return the patch revision number for the software package.

Parameters

result – Object containing aErrNone and the requested value on success.

void **version_IsAtLeast** (struct Result *result, unsigned int major, unsigned int minor, unsigned int patch)

Check that the current software version is at least major.minor.patch

Parameters

- **result** – Object containing aErrNone and the requested value on success.
- **major** – The major revision level.
- **minor** – The minor revision.
- **patch** – The patch level.

void **version_IsAtLeastCompare** (struct Result *result, unsigned int major_lhs, unsigned int minor_lhs, unsigned int patch_lhs, unsigned int major_rhs, unsigned int minor_rhs, unsigned int patch_rhs)

Check that the supplied left hand side (lhs) version is at least (>=) the right hand side (rhs).

Parameters

- **result** – Object containing aErrNone and the requested value on success.
- **major_lhs** – The lhs major revision level.
- **minor_lhs** – The lhs minor revision.
- **patch_lhs** – The lhs patch level.
- **major_rhs** – The rhs major revision level.
- **minor_rhs** – The rhs minor revision.
- **patch_rhs** – The rhs patch level.

void **version_Pack** (struct Result *result, unsigned int major, unsigned int minor, unsigned int patch)

Packs the given version into a single integer

Parameters

- **result** – Object containing aErrNone and the requested value on success.
- **major** – The major revision level.
- **minor** – The minor revision.
- **patch** – The patch level.

3.8 LabVIEW API Reference

The LabVIEW API has been renamed to the CCA API. Generally speaking the functionality is the same with some exceptions which are outlined in the 2.11.x release notes. Please refer to the CCA API reference for information regarding the LabVIEW API.

CCA API Reference

3.9 Reflex Language Reference

3.9.1 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++)
  - **aInclude** - 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.

3.9.2 A Basic "Hello World" Example.

```
#include <a40PinModule.reflex>
#define ON 1
#define OFF 0

a40PinModule stem;

reflex mapEnable()
{
    stem.system.setLED(ON);
}

reflex mapDisable()
{
    stem.system.setLED(OFF);
}
```

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 that 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 acroname 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.

3.9.3 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.

```
// file: flashmyled.reflex

#include <a40PinModule.reflex>
a40PinModule stem;

// 1/2 second delay
#define DELAY 500000
#define ON 1
#define OFF 0

pad[0:0] unsigned char state;

```

(continues on next page)

(continued from previous page)

```
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.

3.9.4 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 is 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() {
    // ...
}
```

3.9.5 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		
reflex	pad	asm
if	else	for
while	do	switch
case	return	char
int	short	unsigned
signed	continue	break

Keyword EBNF

```
keyword  ::=   'reflex' | 'pad' | 'asm' | 'if' | 'else' | 'for' | 'while' | 'do'
               | 'switch' | 'case' | 'return' | 'char' | 'int' | 'short'
               | 'unsigned' | 'signed' | 'continue' | 'break' ;
```

3.9.6 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			Left-to-right
	`++`	Pre Increment	
	`--`	Pre Decrement	
	`()`	Function call	
	`[]`	Index Subscript	
	`.`	command selection	

continues on next page

Table 8 – 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 8 – continued from previous page

Level	Operator	Description	Associativity
13	?:	Ternary Operator	Right-to-left
14	= += -= *= /= %= <<= >>=	Assignment Assignment by sum Assignment by difference Assignment by product Assignment by quotient Assignment by modulo Assignment by left shift Assignment by right shift	Right-to-left

3.9.7 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 hexidecimal values to wite 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 | nontermial } ;
```

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 ;
```

3.9.8 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>
```

3.9.9 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. Redeclaring 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 interace 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 intitialized when they are declared but initialization at decleration is not required.

```
char value;
// or
char value = 12;
```

Routine Variable Declaration EBNF

```
variable_declaration ::= type_specifier , ID , [ "=" , INT ] , ";" ;
```

3.9.10 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 preceed 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);
```

3.9.11 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. The 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 '}' ;
statment_list        ::=  { variable_declaration } , { statement } ;
statment             ::=  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"
```

3.9.12 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 aInclude 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 preproccessor 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 and 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
                                   0x01
0x0002   pushls      0x0000        0x04
                                   0x00
                                   0x00
0x0005   poppc                     0x3C
0x0006   pushls      0x0000        0x04
                                   0x00
                                   0x00
0x0009   pushpc                    0x39
0x000A   popec       0x038460      0x74
                                   0x03
                                   0x84
                                   0x60
0x000E   popn        0x01          0x02
```

(continues on next page)

(continued from previous page)

```
                                0x01
0x0010  pushlc      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
```

3.10 Q-Sys API Reference

Welcome to the BrainStem Q-Sys API reference documentation. This interface provides access to the BrainStem software interfaces.

Changes that do not break a Q-Sys API will not constitute a major version change. Removing features or making significant changes will result in a minor version change. Deploying breaking changes may be necessary for legal, performance, or security reasons. When a new version is released, the current version will remain supported for one year and may become unavailable anytime after this period.

3.10.1 Product Plugins

Table 18: Plugin Product Availability

Product	USBHub3c
usbhub3c	RS232 interface for control and telemetry

USBHub3c Interfaces

Release Notes

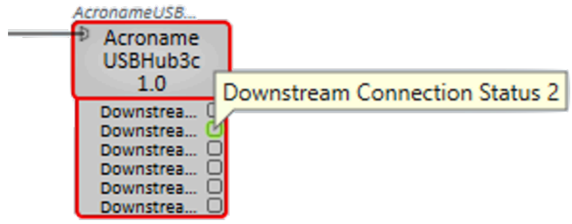
```
# 1.0.0.0
- Initial Release
  - Upstream port switching control
  - Downstream port enable and disable
  - Port enable status
  - USBHub3c firmware version and model information
```

Interface Requirements

A valid Acroname USBHub3c with the Serial Control add-on software feature must be installed. For additional information is available *USBHub3c*.

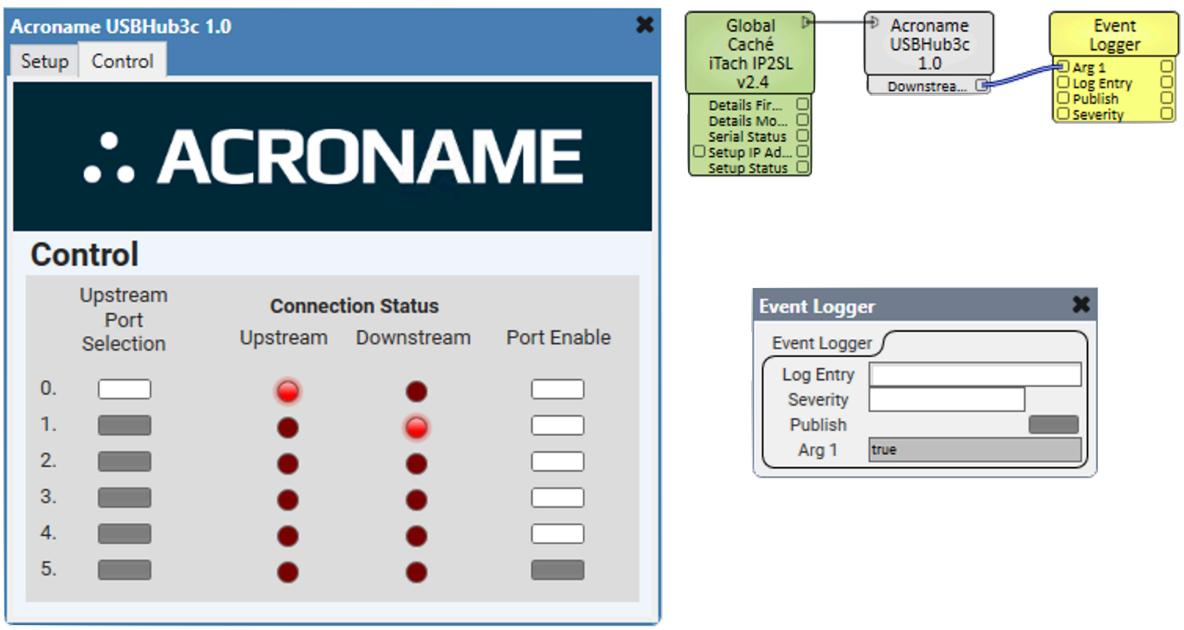
Downstream Connection Status

The downstream connection status interface indicates whether a port is enabled or disabled. The return value is "true" when a USB device is connected, and "false" when disconnected. An active host is required for USB devices to be connected.



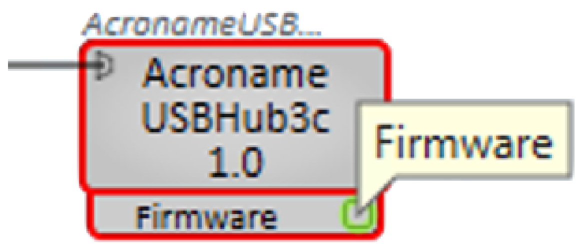
Example Read of Downstream Connection Status Information

Reading the downstream connection status information passed into debugger logger demonstrates reading the model value. Arg1 shows the model read information into the Event Logger.



Firmware Information

Current firmware information can be retrieved as a string with a format of `MAJOR.MINOR.PATCH.REVISION`. The value is equivalent of calling the Module Entities BrainStem System Entity call for get Version.

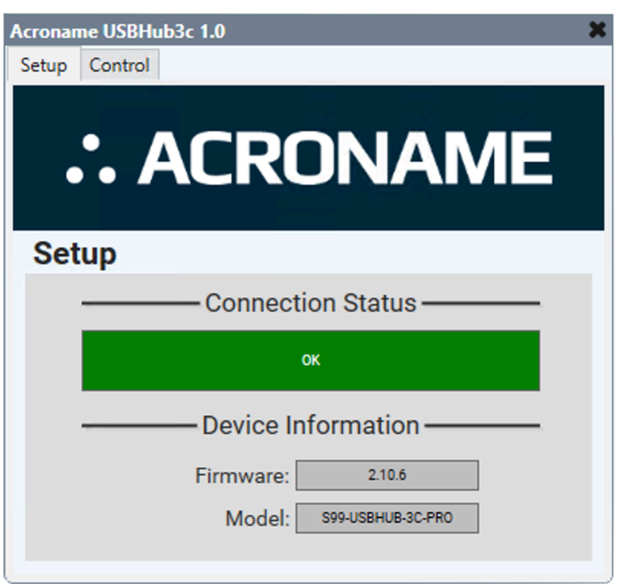


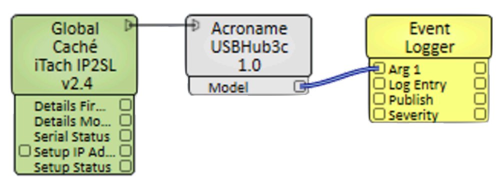
Model Information

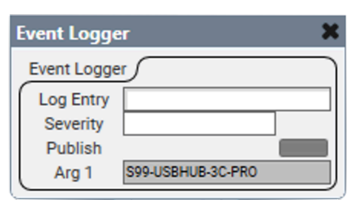
Model information can be retrieved as a string with a format of `S99-USBHUB-3C-PRO` or `S99-USBHUB-3C-LAB`.

Example Read of Model Information

Reading the model information passed into debugger logger demonstrates reading the model value. Arg1 shows the model read information into the Event Logger.

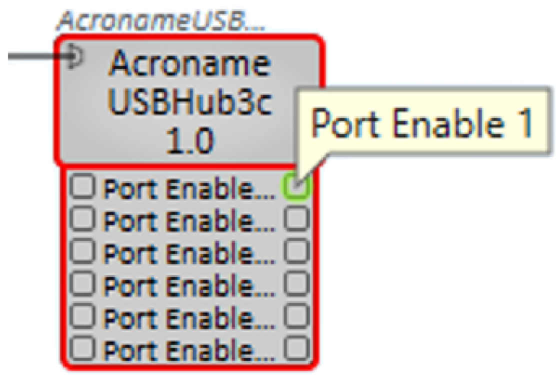






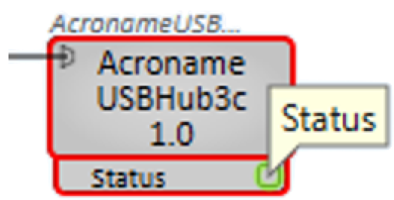
Port Enable

The “port-enable” interfaces enable turning a specific port on or off. When a port is enabled, both power and data lines become active. Conversely, power and data lines get disconnected when the port is disabled, simulating the removal of a USB cable. Note that for USB-C ports, the CC line will be activated to negotiate the appropriate power level for Vbus power. Vbus will not be active until proper CC negotiation has occurred.



Status

Indication of the plugin connection status to the USBHub3c.



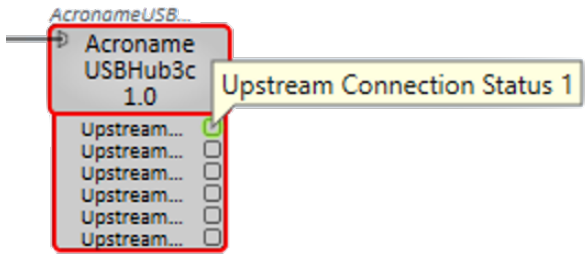
Status Codes

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

Status	Description
0	Connection established or data is received successfully and everything is OK.
1	Compromised connection.
2	Connection or data received with an error indicating a fault.
3	Missing serial port to connect to device.
4	Plugin initializing.

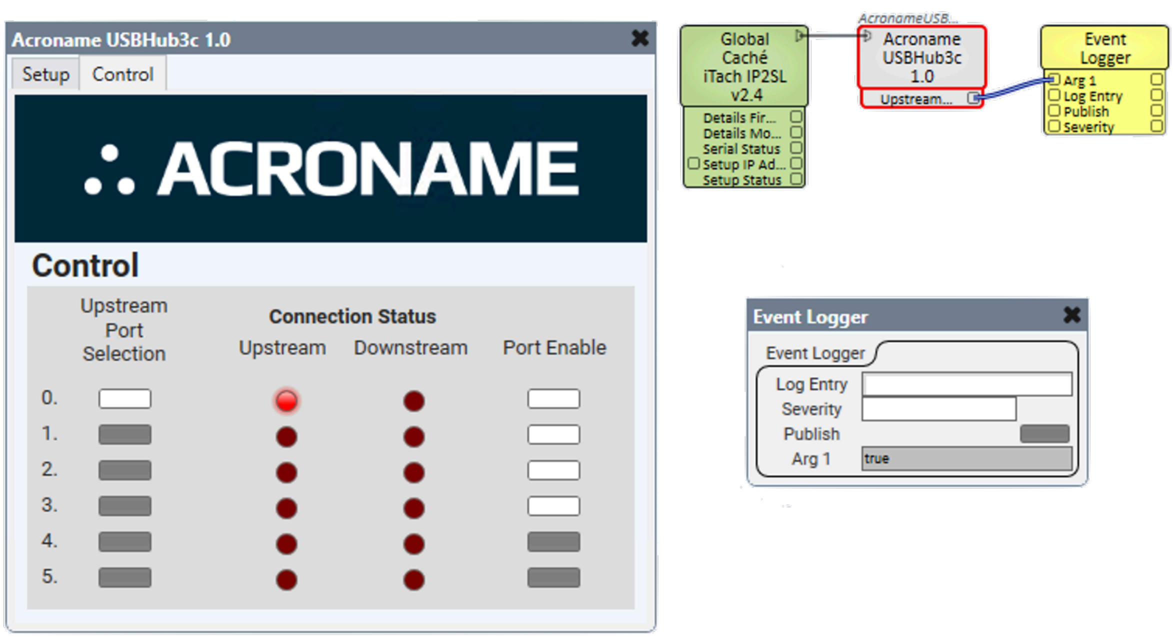
Upstream Connection Status

The "upstream connection status" indicates that a valid USB host is connected to the device. If the connection is established, the output value of the plugin will be "true", which means the device is ready to communicate with the host. However, if the upstream host connection is removed, the output value will be "false", indicating that the communication between the device and the host is no longer available.



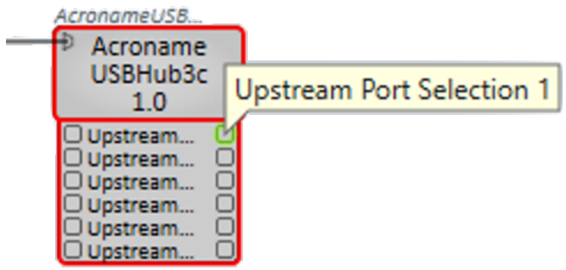
Example Read of Upstream Connection Status Information

Reading the upstream connection status information passed into debugger logger demonstrates reading the model value. Arg1 shows the model read information into the Event Logger.



Upstream Port Selection

The interface for selecting the upstream port determines which port to use as an upstream interface. When calling the function, any input value below 0 will be set to 0.



Version

The `version` endpoint shows version information for the Q-Sys plugin version.

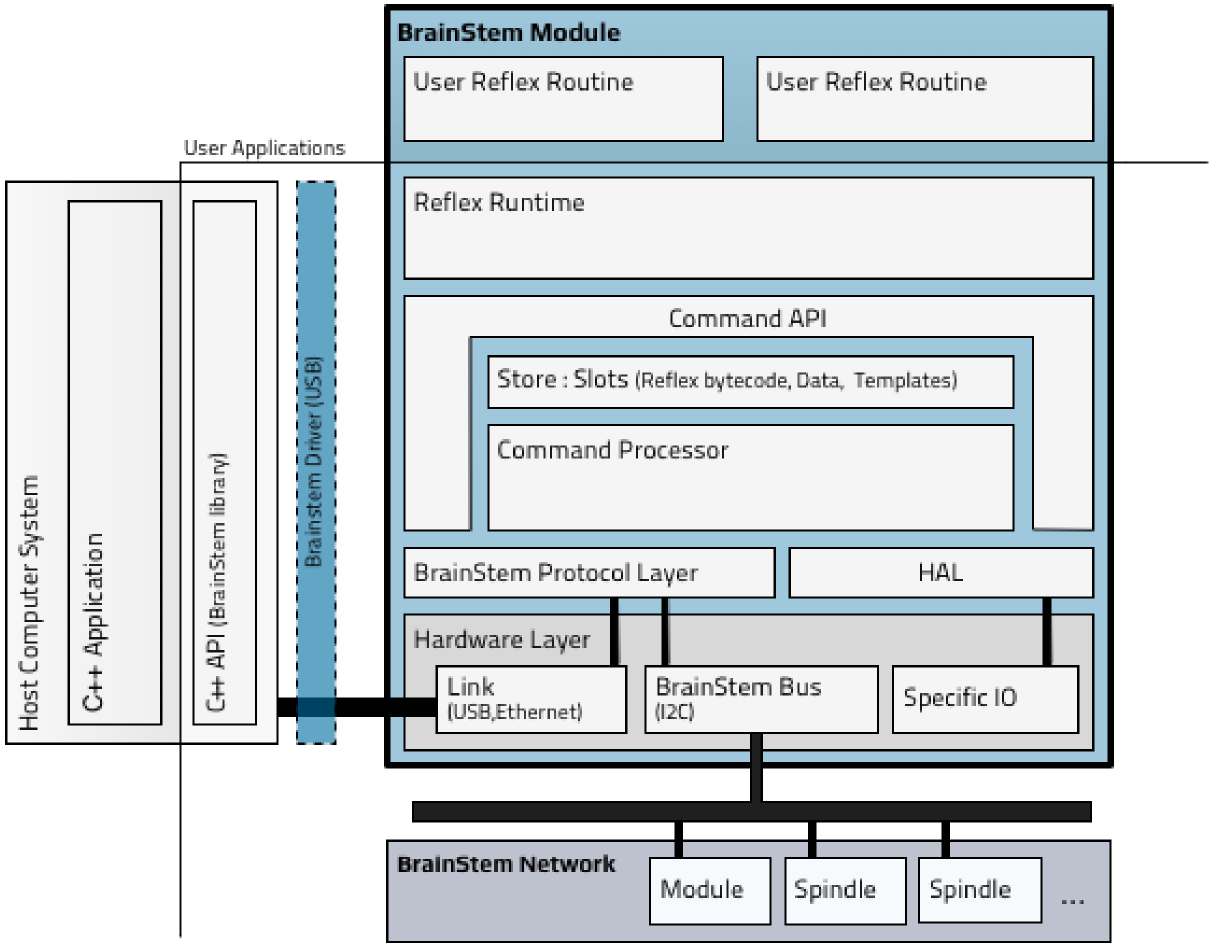
4 BrainStem

4.1 What is BrainStem

Acroname's BrainStem technology is a foundational tool for bridging the gap between hardware and software systems. BrainStem controllers gather and relay peripheral information to other systems, which may be a host computer, additional microcontrollers or an array of sensors and testing equipment. The core BrainStem concept is metaphorically based on the human nervous system's data gathering, interpreting and transmitting processes.

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 Next Steps

The best place to get started with your new BrainStem hardware is by checking out the *Getting Started* section of our documentation.

4.2 aEther

4.2.1 Overview

What is it?

aEther is a software feature that encapsulates the BrainStem software object and provides synchronization across multiple threads, processes and computers by way of a client-server model.

How does aEther help me utilize Acroname hardware?

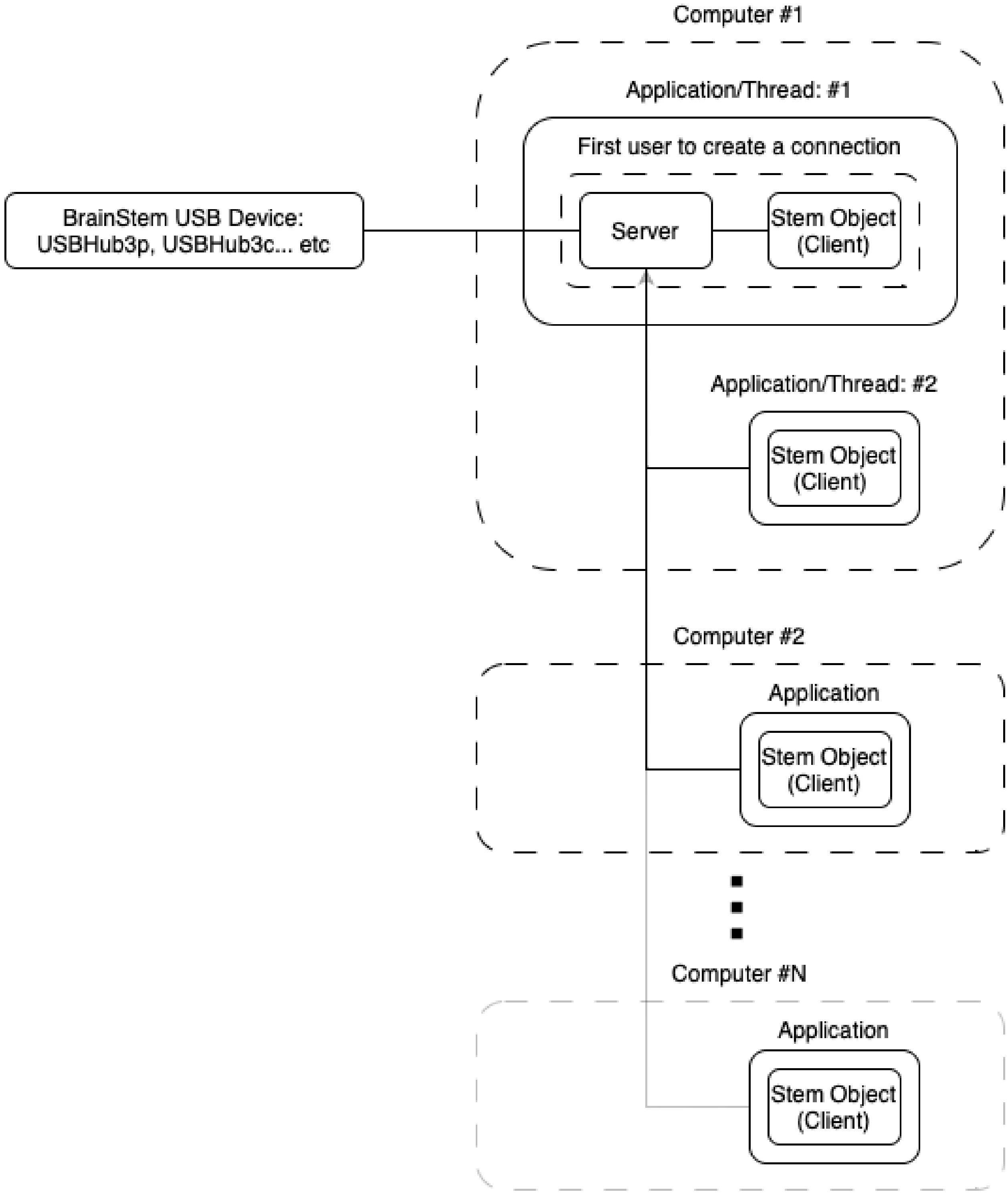
1. **Simplified Synchronization:** Tired of working with Mutexes? Simply create and connect to an object for each thread and we will handle the rest.
2. **Improved Debugging:** Want to know what's going on with Acroname hardware while your application is running or paused on a break point? No problem, just open up HubTool and monitor the devices state.
3. **Remote Access:** Is remote desktop too sluggish or overkill? With the proper network configuration you can access your devices from the other side of the world.

4.2.2 Communication Models

Models define a specific pattern or behavior. These behaviors have different attributes that might be more applicable for specific situations. The BrainStem protocol supports multiple communications models. By default the Client-Server model (aEther) is used because it is the most user friendly and adaptable.

Client-Server Model (aEther)

The Client-Server model works by routing all traffic through the operating systems socket layer. When the first connection is made a "server" and "client" pair will be created. All subsequent connections will be "clients" only. Note that the server is owned by the first user to connect to the device. Deletion of that object will result in ALL clients being disconnected. The server encapsulates the device by way of the direct connection model (discussed below). By doing this the server can manage simultaneous access of many clients.


Computer #1
Application/Thread: #1
First user to create a connection
BrainStem USB Device:
USBHub3p, USBHub3c... etc
Server
Stem Object
(Client)
Application/Thread: #2
Stem Object
(Client)
Computer #2
Application
Stem Object
(Client)
Computer #N
Application
Stem Object
(Client)

Configuring the device for external exposure

By default all sockets are restricted to the localhost. In other words, by default these sockets do NOT have access to the outside world and therefore should not trigger any firewall warnings. In the event you do want the device to be accessible remotely it simply needs to be enabled. Generally speaking this only exposes it to your local network; further exposure to the internet will depend on configurations outside of BrainStem.

C++

```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.localOnly = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config = stem.getConfig()
config.localOnly = False

#NOTE: This must be done before connection
stem.setConfig(config)

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

Configuring the device for a specific network adapter

Sometimes it is necessary to define the network interface you would like to use. If your system has many network adapters (including virtual ones) you might need to programmatically configure this setting. Typically, this is only common for the following cases.

1. Hosting a device for external access (localOnly = False) - By default the first interface will be used. Typically, this is the default interface. If the default is not the desired interface you will need to configure it as shown below.
2. Discovery - By default discovery will search the localhost only; however, if you provide a valid network interface it will search that instead. This is particularly important if using stem.discoverAndConnect() for a remote device that is not directly connected to the computer. The network interface of the module must be set before calling a connect function as shown below.

Note that you can programmatically request the available interfaces on your machine through "aDiscovery_GetIPv4Interfaces"

C++

```
#include "BrainStem2/BrainStem-all.h"

```

(continues on next page)

(continued from previous page)

```
USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.networkInterface = 0xA00A8C0; //Example: 192.168.0.10 (network byte order)
config.localOnly = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config = stem.getConfig()
config.networkInterface = 0xA00A8C0 #Example: 192.168.0.10 (network byte order)
config.localOnly = False

#NOTE: This must be done before connection
stem.setConfig(config)

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

Disabling auto fallback

In order to make a more pleasant user experience, the "discoverAndConnect" process will automatically check other transport types for devices if a device is unavailable with the given transport type. For instance, we want to create 2x BrainStem software objects that connect to the same USB module (ie USBHub3p etc). The first caller should use the transport type "USB" while the second should use "AETHER". When fallback is enabled, this can be ignored and the underlying software will automatically determine the appropriate transport for you. If you use the "USB" transport and it is determined that someone is already connected to this device it will automatically try the "AETHER" transport next. Depending on your situation you might not want this behavior. If so, it can be disabled through the NetworkConfig.

C++

```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.fallback = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem
```

(continues on next page)

(continued from previous page)

```
stem = brainstem.stem.USBHub3c()
config = stem.getConfig()
config.fallback = False

#NOTE: This must be done before connection
stem.setConfig(config)

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

Disabling the Client-Server model

There might be specific situations in which you want to disable the client-server model and revert to the previous direct connection model paradigm. This is very simply done through the aEtherConfig structure.

C++

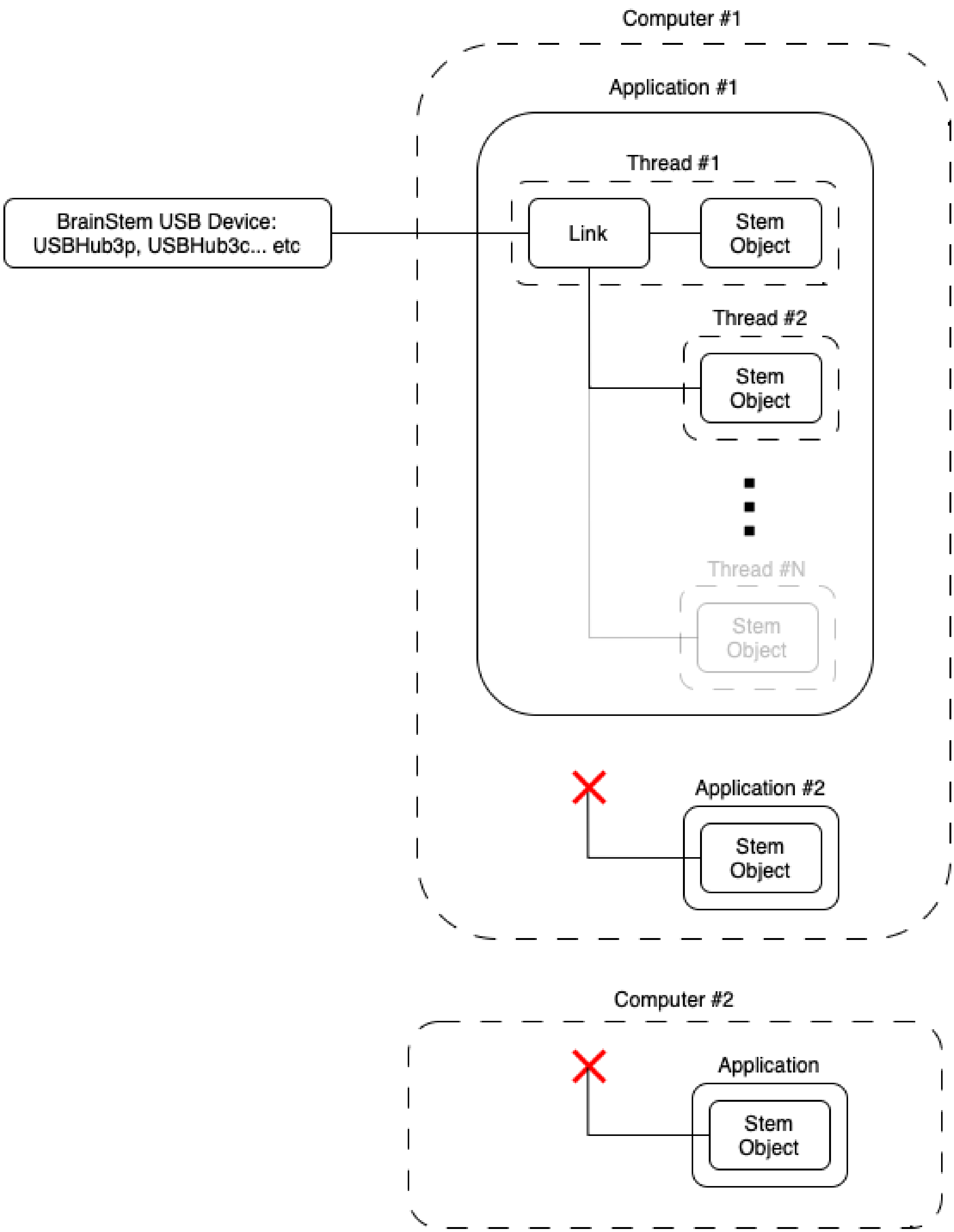
```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.enabled = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config = stem.getConfig()
config.enabled = False

#NOTE: This must be done before connection
stem.setConfig(config)

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

Direct Connection Model

The direct connect model is Acroname's original model and is the core of the server in the client-server model. In this model the BrainStem software object owns the device communications path and is generally thread safe. When using this model the device is NOT accessible in any other processes nor is it accessible by another computer. This model is slightly faster, but more restrictive.



To select this configuration it must be done through the aEtherConfig structure.

C++

```
#include "BrainStem2/BrainStem-all.h"

USBHub3c stem;
Acroname::BrainStem::aEtherConfig config;
stem.getConfig(&config);
config.enabled = false;

//NOTE: This must be done before connection
stem.setConfig(config);

stem.discoverAndConnect(USB)
```

Python

```
import brainstem

stem = brainstem.stem.USBHub3c()
config = stem.getConfig()
config.enabled = False

#NOTE: This must be done before connection
stem.setConfig(config)

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

4.3 Getting Started

Before you begin you'll need to collect the following items.

- A BrainStem Module with development board or an Acroname Hub/Switch.
- A Link Transport Cable (Ethernet or USB).
- The BrainStem Support software package.

BrainStem Devices and Development boards can be purchased from the Acroname Products.

The latest version of the BrainStem Development Kit (BDK)[100] or HubTool[101] can be downloaded from the Acroname Download Page. Extract the package to the location of your choice when the download has finished.

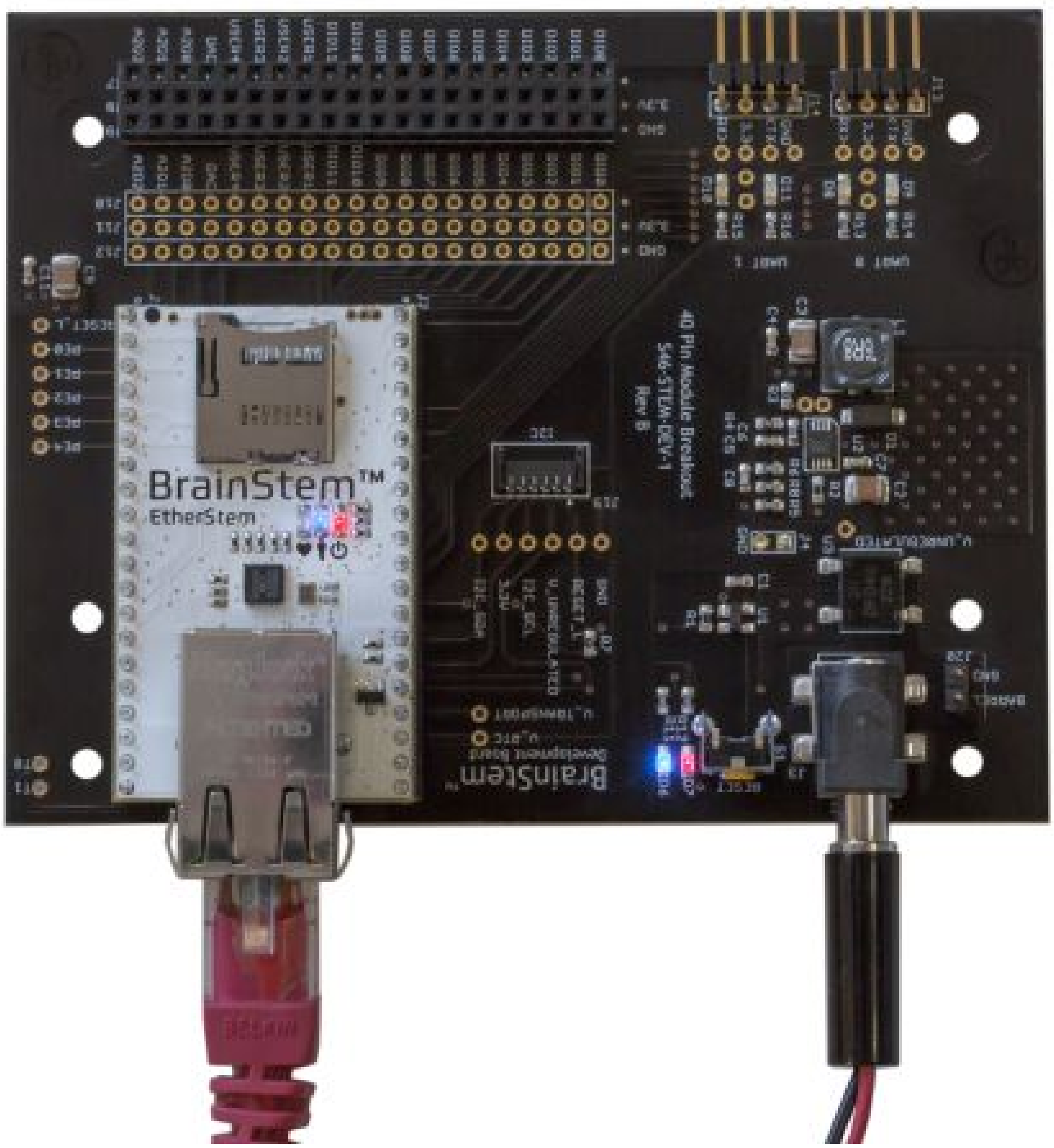
4.3.1 Do I need Drivers?

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.

[100] https://acroname.com/api
[101] https://acroname.com/hubtool

4.3.2 Connecting to a BrainStem Device

With the exception of our Hubs/Switches most BrainStem devices must be plugged into the BrainStem Development Board to receive power. Once the device has been plugged into a development board it is safe to insert the link transport cable. The red power LED should be on and the green heartbeat LED should flicker rapidly when power is first applied. If you are using a Hub/Switch connect the supplied power adapter to the device.



4.3.3 Launch HubTool

For BrainStem devices and Hubs navigate to the bin folder of the BrainStem Development Kit to find HubTool.

4.3.4 Toggling the LED

HubTool will periodically search for connected devices and display them in the lower right corner. Selecting your device will cause a UI to 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 Updating Firmware via HubTool

BrainStem firmware can be loaded using the HubTool application, located within the `bin` folder within the Brainstem Software Development Kit. Downloads for all of Acroname's products can be found at Download Center under the Support tab.

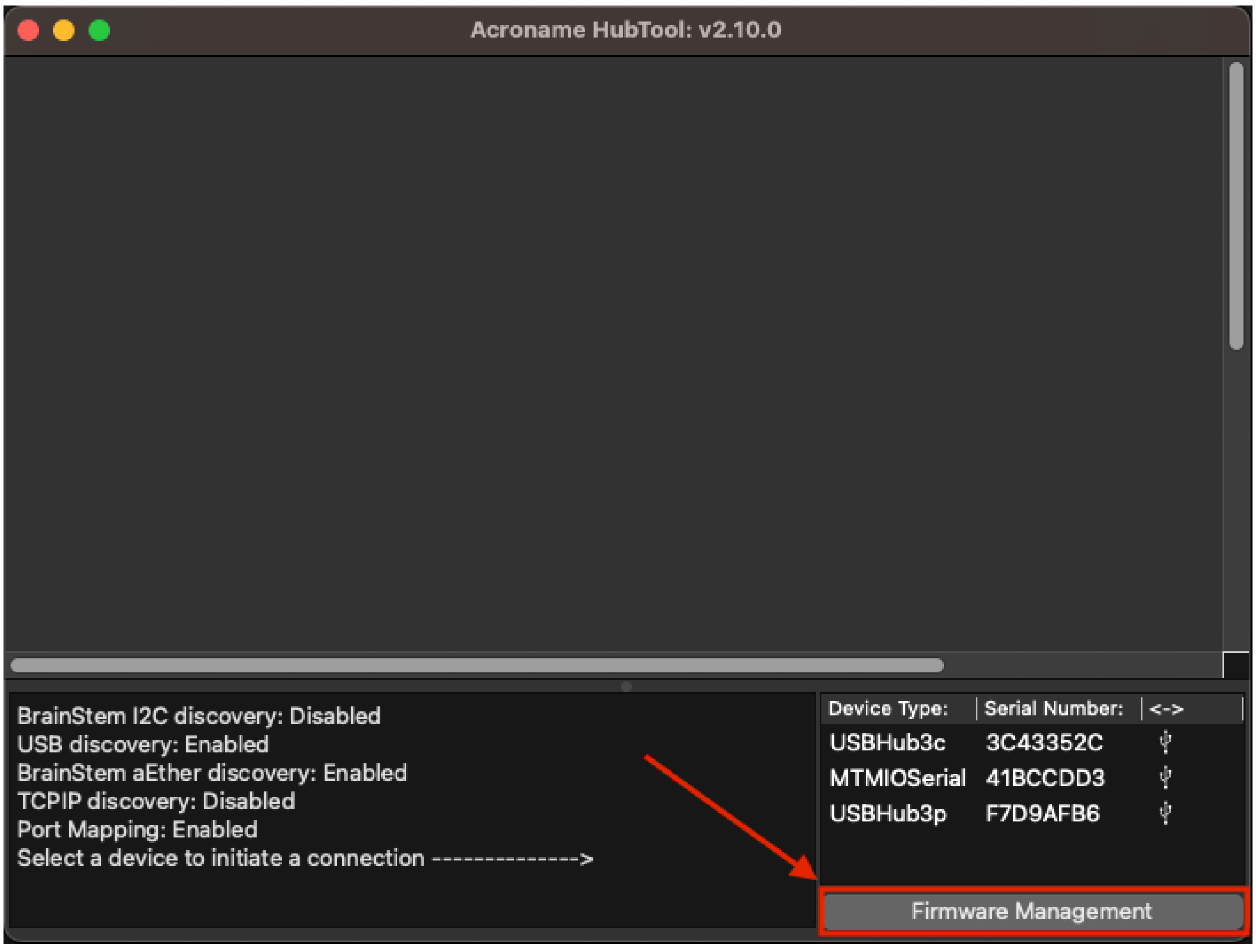
When HubTool is opened, click the "Firmware Management" button to switch to the Updater window:

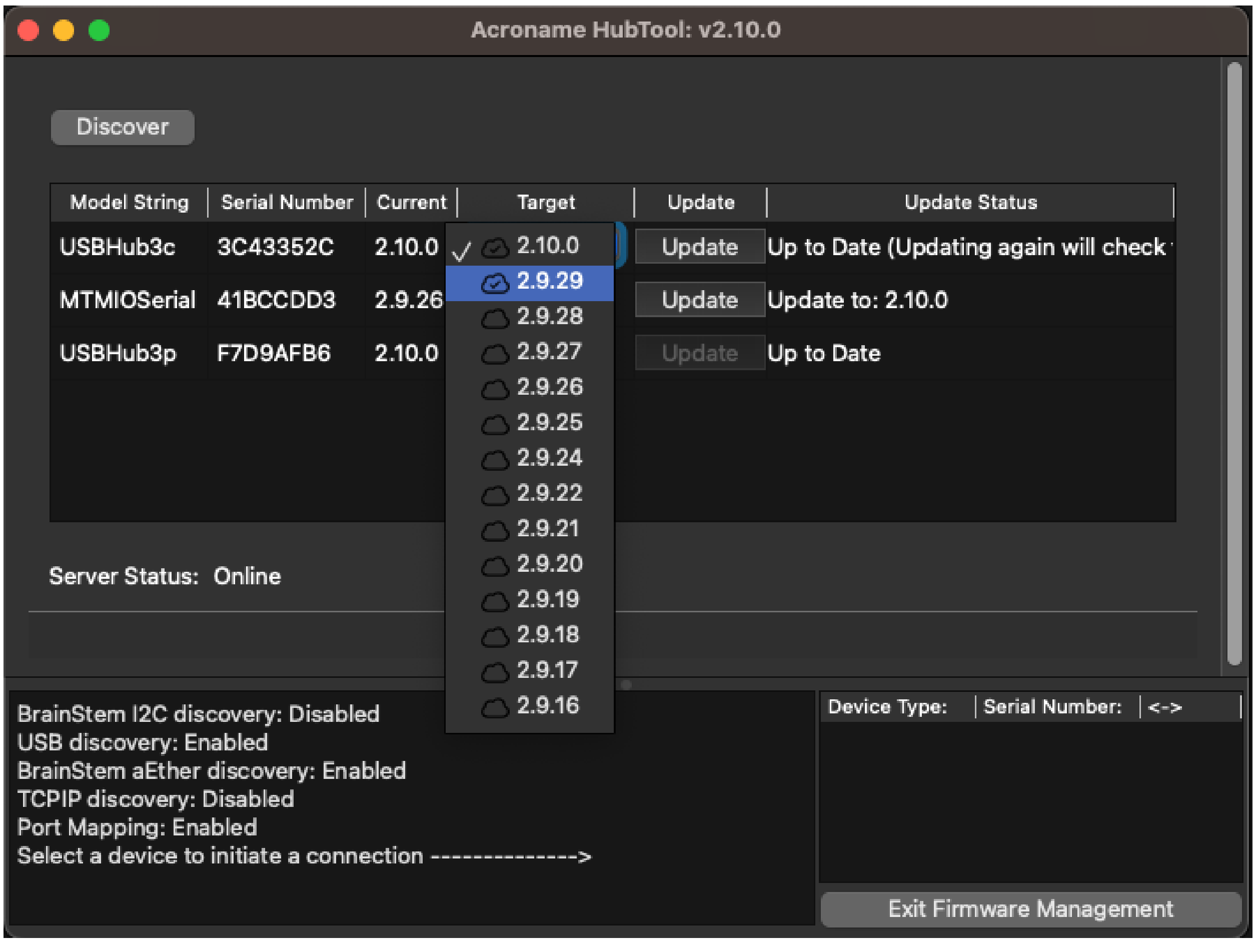
When the Updater window is opened, it will automatically populate with a list of all attached Acroname devices. If a new device is attached after this window is open, the "Discover" button will refresh the device list.

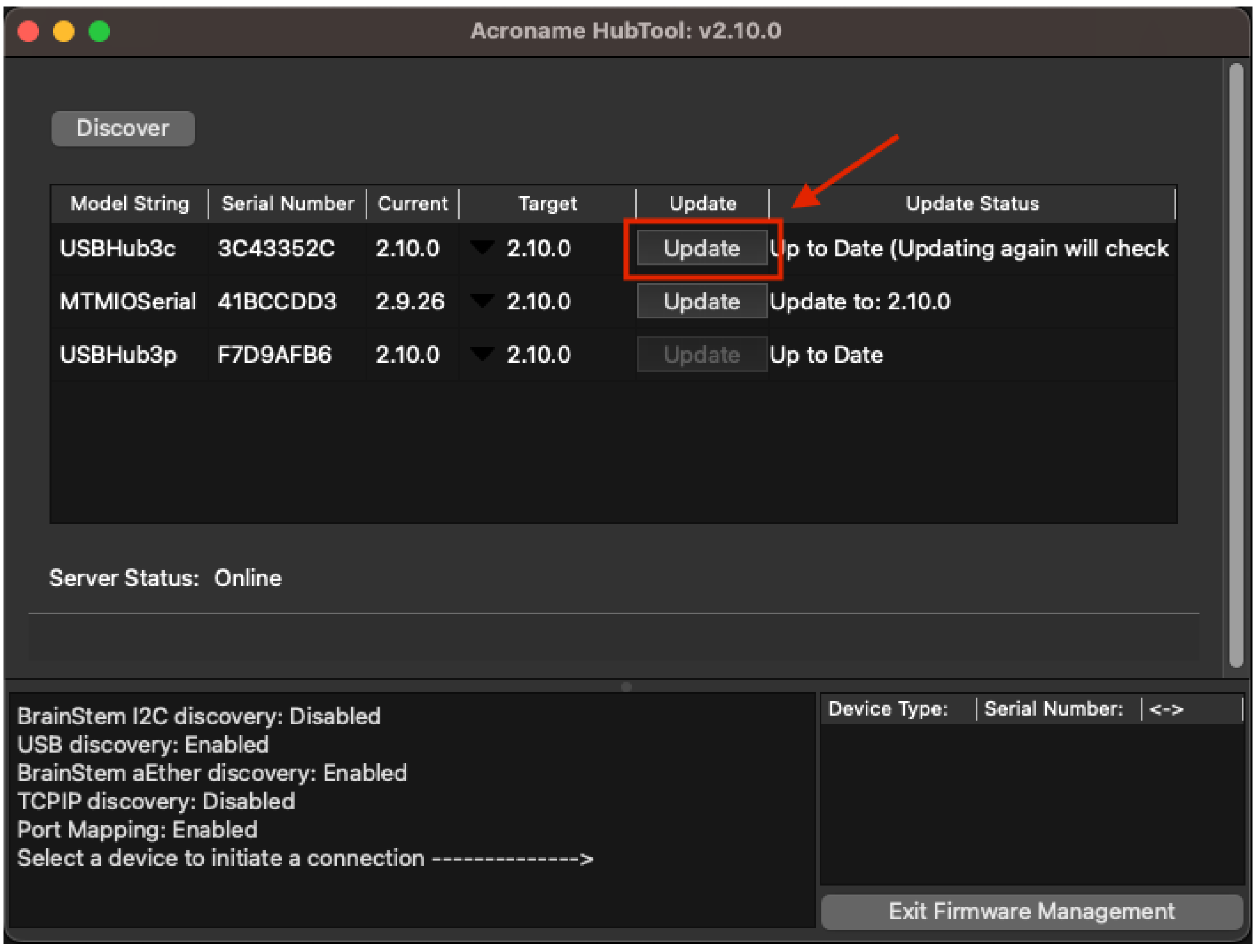
Each device entry in the Updater shows the model name, the serial number, the current firmware version, the firmware version to update to, and the status of the update. The update version will automatically select the most recent release. To select a different version, click the Target version field and click the desired version:

When the correct target version for a device has been selected, press the "Update" button for that device:

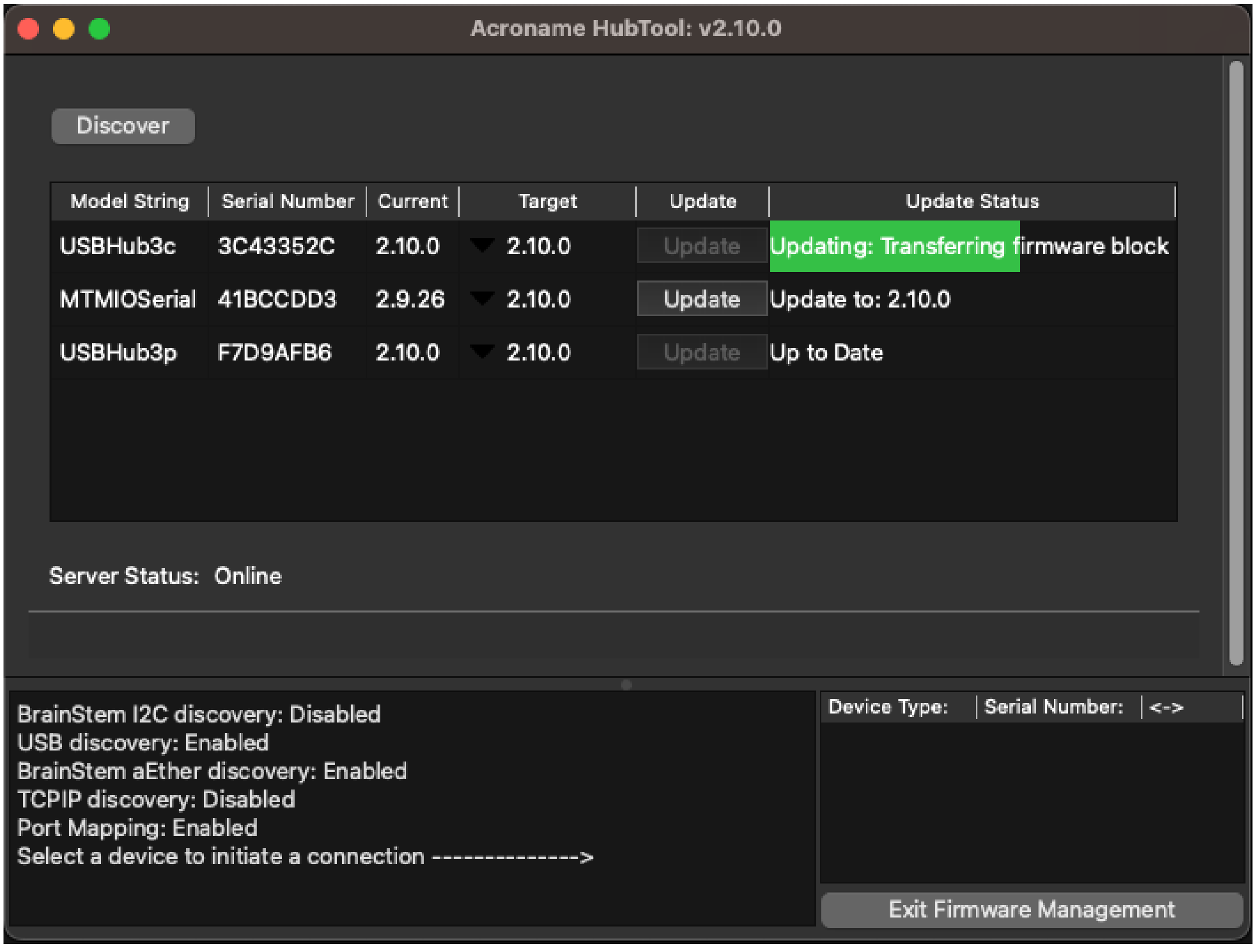
While the device is updating, the process will be shown in the right-most column.


Acroname HubTool: v2.10.0
BrainStem I2C discovery: Disabled
USB discovery: Enabled
BrainStem aEther discovery: Enabled
TCPIP discovery: Disabled
Port Mapping: Enabled
Select a device to initiate a connection -------------->
Device Type:
Serial Number:
<->
USBHub3c
3C43352C
MTMIOSerial
41BCCDD3
USBHub3p
F7D9AFB6
Firmware Management


Acroname HubTool: v2.10.0
Discover
Model String
Serial Number
Current
Target
Update
Update Status
USBHub3c
3C43352C
2.10.0
2.10.0
Update
Up to Date (Updating again will check
2.9.29
MTMIOSerial
41BCCDD3
2.9.26
2.9.28
Update
Update to: 2.10.0
USBHub3p
F7D9AFB6
2.10.0
2.9.27
Update
Up to Date
2.9.26
2.9.25
2.9.24
2.9.22
2.9.21
2.9.20
2.9.19
2.9.18
2.9.17
2.9.16
Server Status: Online
BrainStem I2C discovery: Disabled
USB discovery: Enabled
BrainStem aEther discovery: Enabled
TCPIP discovery: Disabled
Port Mapping: Enabled
Select a device to initiate a connection -------------->
Device Type:
Serial Number:
<->
Exit Firmware Management


Acroname HubTool: v2.10.0
Discover
Model String
Serial Number
Current
Target
Update
Update Status
USBHub3c
3C43352C
2.10.0
2.10.0
Update
Up to Date (Updating again will check
MTMIOSerial
41BCCDD3
2.9.26
2.10.0
Update
Update to: 2.10.0
USBHub3p
F7D9AFB6
2.10.0
2.10.0
Update
Up to Date
Server Status: Online
BrainStem I2C discovery: Disabled
USB discovery: Enabled
BrainStem aEther discovery: Enabled
TCPIP discovery: Disabled
Port Mapping: Enabled
Select a device to initiate a connection -------------->
Device Type:
Serial Number:
<->
Exit Firmware Management

Note: Do not quit HubTool during the update. This may cause the device firmware to become corrupted, requiring a separate firmware recovery.



If the update was successful, the device that was updated will show the current version and "Update Success" in the right-most column.

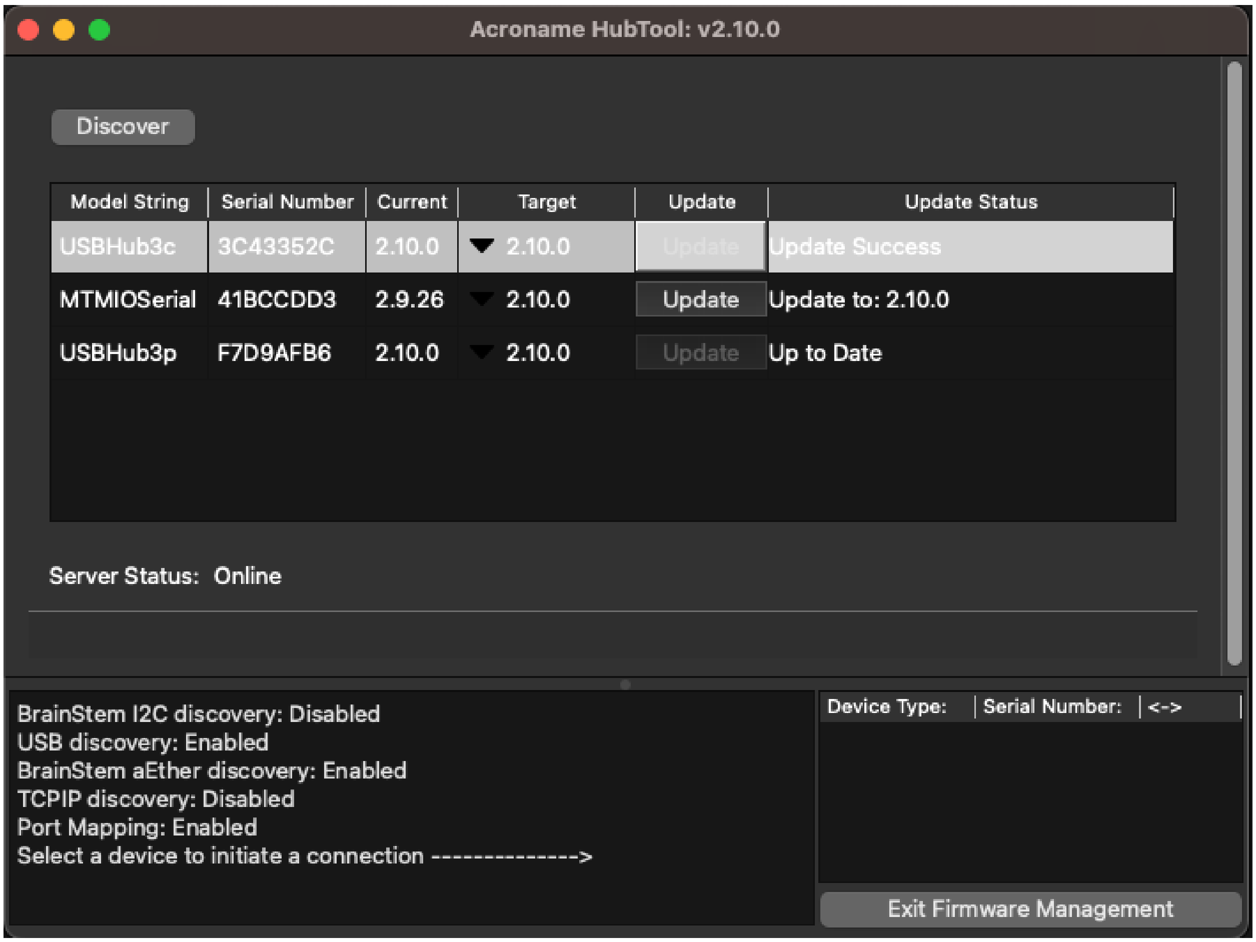
Click "Exit Firmware Management" in the lower-right corner to return to the main HubTool application.

4.4.2 Updating Firmware via CLI

BrainStem firmware can be modified using the Updater CLI utility. This method is often very useful in remote installations and large scale automation systems as it is easily invoked with a simple script. Using your computer's operating systems terminal interface, navigate to the "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.


Acroname HubTool: v2.10.0
Discover
Model String
Serial Number
Current
Target
Update
Update Status
USBHub3c
3C43352C
2.10.0
2.10.0
Update
Update Success
MTMIOSerial
41BCCDD3
2.9.26
2.10.0
Update
Update to: 2.10.0
USBHub3p
F7D9AFB6
2.10.0
2.10.0
Update
Up to Date
Server Status: Online
BrainStem I2C discovery: Disabled
USB discovery: Enabled
BrainStem aEther discovery: Enabled
TCPIP discovery: Disabled
Port Mapping: Enabled
Select a device to initiate a connection -------------->
Device Type:
Serial Number:
<->
Exit Firmware Management

```
$> ./Updater
```

If you would like additional help information, including examples you can use the -H command.

```
$> ./Updater -H
```

```
Updater (Version: 2.10.1 Sep 28 2023)
(Copyright (C) 1994-2023, Acroname Inc.)

Application Usage:
Updater <-B | -D | -G | -U > [-Hh | --help] [-b build_num]
    [-d serial_num]
    [-t transport] [-r router_serial]
    [-s serial_port] [-o baud_rate]

  BrainStem Update and Maintenance Application
    -H -h or --help for full usage information.

  Parameters to commands:
     -b <build number>        - Build version of the firmware to transfer.
                                [Default: latest version available].
     -d <device>              - Device serial number of the device to access.
     -t <transport type>      - communications method [USB,TCP,RECOVER].
     -r <router serial>       - indirect connection through router device
     -s <serial port>         - name of serial port to use for recovery
     -o <baud rate>           - baud rate for serial port (2400, 4800, 9600,
                                19200, 38400, 57600, 115200, 230400)
  Commands:
     -B                       - list the Builds available for given device.

     -D                       - Discover devices that are currently connected.

     -G                       - Get build file from the Acroname server and
                                store on local machine for download to device.
                                Must specify a specific device ["-d" option].
                                Default is to obtain the latest build for the
                                given device, override with "-b" option.

     -U                       - Update the firmware on device specified.
                                Must specify a specific device ["-d" option].
                                Default is to download the latest build for the
                                given device, override with "-b" option or
                                specify specific BIRD file using "-f" option.
     -H or -h                 - Display extended usage information.

 Sample usage:

-D                             Discover all devices connected by either USB or TCPIP.
                               Records the device information into the settings for␣
 ↪that device.

-D -t USB                      Discover all the devices connected by USB. Records the
                               device information into the settings for that device.

-D -t USB -d 0F5849A           Discover settings of the specific device (serial␣
 ↪number 0x40F5849A)
```

(continues on next page)

(continued from previous page)

```
-G -d 40F5849A              Get the latest firmware for the given device.
                            Downloads the firmware file into the updater specific␣
↪device directory.

-U -d 40F5849A              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 40F5849A -b 99528558  Get the specified build's firmware for the given␣
↪device.
                            Downloads the firmware file into the updater's specific
                            device directory with the build number as the filename.

-G -U -d 40F5849A           Get the latest firmware for the given device.
                            Downloads the firmware file into the updater specific
                            device directory. After download, update the firmware
                            using the latest release.

-U -t RECOVER -s /dev/serial  Update the firmware as described above, but uses the␣
↪serial interface
                            in communicating with the device to install the latest␣
↪firmware.

 Sample usage for network discovery and updates:

-D -r D272031D              Discover all devices connected to the I2C BrainStem␣
↪network
                            of the given routing device D272031D.

-G -U -d 2181F0EE -r D272031D
                            Get and Update to the latest firmware for device␣
↪2181F0EE
                            indirectly through routing device D272031D.
```

As you can see there are a lot of options for customizing the Updater utility to meet your needs; however, there are just a few basic command that will fit the needs of most users.

Next we will look at a few examples of how to use the updater.

Example: Updating to the Latest Firmware

In this example we will go through the steps required to update our BrainStem module. We will be using a 40pin USBStem module throughout this excersize; however, the other modules work in a similar way.

Make sure your device is connected and has power. Refer to the *Getting Started* page for additional information.

Checking for connected devices:

This command will check for USB devices connected to your machine.

```
$> ./Updater -D
```

Looking at the output below you can see that two devices were discovered, one USBHub3+ and one USBHub3c. In this example we will be using the USBHub3c information. This information will be handy in the next step as we will need the serial number of our device in order to update it.

```
Updater (Version: 2.10.1 Sep 28 2023)
Searching for BrainStem devices
Discovered Devices: (USB)
    Device    Module   Router   Model          Firmware Version (Build)
    3C43352C  06       06       USBHub3c       2.10.2 (1234567890)
    F7D9AFB6  06       06       USBHub3p       2.10.2 (4101717688)
```

Getting the latest firmware from Acroname's servers:

Using the serial number for the relevant device above, we will construct the following command. The "-G" will pull the most recent firmware from Acroname's server for the given device serial number ("-d").

```
$> ./Updater -G -d 0x3C43352C
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Retrieving firmware for device with serial number 3C43352C
No build specified
Using Latest build
Retrieving build 1256480617
Validating build 1256480617 with server
Build 1256480617 is valid
```

Loading the latest firmware:

Now that we have successfully pulled the most up to date firmware we now need to apply it to the device. The following code will apply the most up to date firmware to the given device.

```
$> ./Updater -U -d 0x3C43352C
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Updating from Birdfile: /Users/cgoss/.acroname/updater/3C43352C/1256480617.bird
Transferring loader to device
Transferring loader block 1 of 3, 65540 bytes
Transferring loader block 2 of 3, 50068 bytes
Transferring loader block 3 of 3, 12288 bytes
Transferred 3 blocks, 127896 total bytes
Starting Load on Device
Transferring firmware block 1 of 20, 65540 bytes
Transferring firmware block 2 of 20, 65540 bytes
Transferring firmware block 3 of 20, 65540 bytes
Transferring firmware block 4 of 20, 65540 bytes
Transferring firmware block 5 of 20, 65540 bytes
```

(continues on next page)

(continued from previous page)

```
Transferring firmware block 6 of 20, 65540 bytes
Transferring firmware block 7 of 20, 65540 bytes
Transferring firmware block 8 of 20, 49268 bytes
Transferring firmware block 9 of 20, 57348 bytes
Transferring firmware block 10 of 20, 65540 bytes
Transferring firmware block 11 of 20, 65540 bytes
Transferring firmware block 12 of 20, 65540 bytes
Transferring firmware block 13 of 20, 65540 bytes
Transferring firmware block 14 of 20, 65540 bytes
Transferring firmware block 15 of 20, 65540 bytes
Transferring firmware block 16 of 20, 49268 bytes
Transferring firmware block 17 of 20, 3924 bytes
Transferring firmware block 18 of 20, 3924 bytes
Transferring firmware block 19 of 20, 4104 bytes
Transferring firmware block 20 of 20, 8 bytes
Transferred 20 blocks, 1019864 total bytes
Update time: 57.5460 sec
```

At this point the firmware has been downloaded to the device and the device has reset and is running with the new firmware.

Example: Reverting to a Previous Version

In our next example we are going to assume that we have just updated to the 2.1.5 firmware; however, for some reason we are not satisfied with how the device is behaving and we want to return back to 2.1.4. With the Updater utility this is easy to do.

Before we begin lets make sure your device is connected and has power. Refer to the *Getting Started* page for additional information.

Locating the previous firmware:

Using the '-B' command we can request the available builds for a given device. Since firmware is specific to each device we must also supply the device's serial number with the '-d' parameter.

```
$> ./Updater -B -d 0x3C43352C
```

Below you will see what was printed on my machine[102]. All we need to do is take note of the specific build number that is associated with the version we would like to revert to.

```
Updater (Version: 2.10.1 Sep 28 2023)
Build List for device [3C43352C]:

Build                   Version
910961995               2.10.0
3592403412              2.10.1
1256480617              2.10.2
287855797               2.9.16
273238664               2.9.17
3802420316              2.9.18
804960244               2.9.19
```

(continues on next page)

[102] 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.

(continued from previous page)

```
3723629684            2.9.20
526817175             2.9.21
3401723282            2.9.22
619738343             2.9.24
2286324745            2.9.25
3108222154            2.9.26
2422939150            2.9.27
227123222             2.9.28
2425394894            2.9.29
```

Applying a specific build to a device:

As previously explained we are hypothetically having issues with our device after updating and we want to revert to a previous version. Now that we have located the previous build number lets apply it by adding the "-b" parameter to the last command in the example before.

```
$> ./Updater -G -U -d 0x3C43352C -b 227123222
```

You should see something similar to the below output.

```
Updater (Version: 2.10.1 Sep 28 2023)
Retrieving firmware for device with serial number 3C43352C
Retrieving firmware build 227123222
Retrieving build 227123222
Validating build 227123222 with server
Build 227123222 is valid
Updating firmware with build: 227123222
Transferring loader to device
Transferring loader block 1 of 3, 65540 bytes
Transferring loader block 2 of 3, 50068 bytes
Transferring loader block 3 of 3, 12288 bytes
Transferred 3 blocks, 127896 total bytes
Starting Load on Device
Transferring firmware block 1 of 20, 65540 bytes
Transferring firmware block 2 of 20, 65540 bytes
Transferring firmware block 3 of 20, 65540 bytes
Transferring firmware block 4 of 20, 65540 bytes
Transferring firmware block 5 of 20, 65540 bytes
Transferring firmware block 6 of 20, 65540 bytes
Transferring firmware block 7 of 20, 65540 bytes
Transferring firmware block 8 of 20, 49268 bytes
Transferring firmware block 9 of 20, 57348 bytes
Transferring firmware block 10 of 20, 65540 bytes
Transferring firmware block 11 of 20, 65540 bytes
Transferring firmware block 12 of 20, 65540 bytes
Transferring firmware block 13 of 20, 65540 bytes
Transferring firmware block 14 of 20, 65540 bytes
Transferring firmware block 15 of 20, 65540 bytes
Transferring firmware block 16 of 20, 49268 bytes
Transferring firmware block 17 of 20, 3924 bytes
Transferring firmware block 18 of 20, 3924 bytes
Transferring firmware block 19 of 20, 4104 bytes
Transferring firmware block 20 of 20, 8 bytes
Transferred 20 blocks, 1019864 total bytes
Update time: 57.5460 sec
```

Confirm that you have successfully restored the old firmware:

Just to be safe 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 2.9.28 firmware.

```
Updater (Version: 2.10.1 Sep 28 2023)
Searching for BrainStem devices
Discovered Devices: (USB)
    Device    Module   Router  Model          Firmware Version (Build)
    3C43352C  06       06      USBHub3c       2.9.28 (227123222)
    F7D9AFB6  06       06      USBHub3p       2.10.2 (4101717688)
```

4.4.3 Updating Firmware without an Internet Connection

In order for the Updater to retrieve a firmware image from the Acroname servers, a connection to the public internet is required. However, in certain circumstances, a BrainStem device may be used on a system which is unable to access the external internet. In this case, some additional steps are necessary to retrieve the required firmware image for a product.

Using your computer's operating systems terminal interface, navigate to the "cli" folder located in the Brainstem Dev Kit download. Downloads for all of Acroname's products can be found at Download Center under the Support tab.

Make sure your device is connected and has power. Refer to the *Getting Started* page for additional information.

Retrieve the Serial Number for the device to be updated

This command will check for USB devices connected to your machine:

```
$> ./Updater -D
```

Looking at the output below you can see that two devices were discovered, one USBHub3+ and one USBHub3c. In this example we will be using the USBHub3c information. This information will be handy in the next step as we will need the serial number of our device in order to update it.

```
Updater (Version: 2.10.1 Sep 28 2023)
Searching for BrainStem devices
Discovered Devices: (USB)
    Device    Module   Router  Model          Firmware Version (Build)
    3C43352C  06       06      USBHub3c       2.10.2 (1234567890)
    F7D9AFB6  06       06      USBHub3p       2.10.2 (4101717688)
```

Getting the latest firmware from Acroname's servers

Using the serial number for the relevant device above, we will construct the following command. The "-G" will pull the most recent firmware from Acroname's server for the given device serial number ("-d").

```
$> ./Updater -G -d 0x3C43352C
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Retrieving firmware for device with serial number 3C43352C
No build specified
Using Latest build
Retrieving build 1256480617
Validating build 1256480617 with server
Build 1256480617 is valid
```

The firmware files will be stored in a hidden directory in the user's home directory:

1. Windows: `\Users\<username>\.acroname\updater\<serial number>\`
2. Mac/Linux: `~/.acroname/updater/<serial number>/`

The directory for each serial number will have at least one `<build number>.bird` file; these files are the firmware image to be flashed onto the device. The build number in the filename corresponds to the build number in the output of the Updater command.

These files can be copied to the offline computer that is attached to the device that is to be updated.

Note: The downloaded firmware files are specific to one serial number; a `.bird` file that was downloaded for one serial number cannot be installed onto a different device.

Loading the firmware files

After the firmware files for the hub to be updated are copied to the offline host computer, the following command will apply the firmware file to the given device.

```
$> ./Updater -U -d 0x3C43352C -f <path to .bird file>
```

```
Updater (Version: 2.10.1 Sep 28 2023)
Updating from Birdfile: /Users/cgoss/.acroname/updater/3C43352C/1256480617.bird
Transferring loader to device
Transferring loader block 1 of 3, 65540 bytes
Transferring loader block 2 of 3, 50068 bytes
Transferring loader block 3 of 3, 12288 bytes
Transferred 3 blocks, 127896 total bytes
Starting Load on Device
Transferring firmware block 1 of 20, 65540 bytes
Transferring firmware block 2 of 20, 65540 bytes
Transferring firmware block 3 of 20, 65540 bytes
Transferring firmware block 4 of 20, 65540 bytes
Transferring firmware block 5 of 20, 65540 bytes
Transferring firmware block 6 of 20, 65540 bytes
Transferring firmware block 7 of 20, 65540 bytes
Transferring firmware block 8 of 20, 49268 bytes
Transferring firmware block 9 of 20, 57348 bytes
```

(continues on next page)

(continued from previous page)

```
Transferring firmware block 10 of 20, 65540 bytes
Transferring firmware block 11 of 20, 65540 bytes
Transferring firmware block 12 of 20, 65540 bytes
Transferring firmware block 13 of 20, 65540 bytes
Transferring firmware block 14 of 20, 65540 bytes
Transferring firmware block 15 of 20, 65540 bytes
Transferring firmware block 16 of 20, 49268 bytes
Transferring firmware block 17 of 20, 3924 bytes
Transferring firmware block 18 of 20, 3924 bytes
Transferring firmware block 19 of 20, 4104 bytes
Transferring firmware block 20 of 20, 8 bytes
Transferred 20 blocks, 1019864 total bytes
Update time: 57.5460 sec
```

At this point the firmware has been downloaded to the device and the device has reset and is running with the new firmware.

4.4.4 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 a 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[103] where it will then be converted to the I2C transport and sent to a MTM-PM1[104] module.

Before we begin lets make sure your device is connected and has power. Refer to the *Getting Started* page for additional information.

[103] https://acroname.com/store/s67-mtm-etherstem
[104] https://acroname.com/products/ACRONAME-MTM-1-CHANNEL-POWER-MODULE

Discovery

Since we will be updating a device through another device we will need to know the serial number of the device we will be communicating through. To find the serial number we can simply use the '-D' discover command.

```
$> ./Updater -D
```

```
Updater [Version 0.2 Dec 28 2015 13:54:37] [Copyright (C) 1994-2015, Acroname Inc.]

Discovering Devices [USB]:
    Device     Module    Router   Model                 Firmware Version

Discovering Devices [TCPIP]:
    Device     Module    Router   Model                 Firmware Version    [IP address]
    D272031D   04        04       0F [MTMEtherStem]     2.2.0 (0)           [10.128.38.159]
    856C1C03   02        02       05 [EtherStem    ]    2.1.4 (99528558)    [10.128.38.122]

Completed processing: Updater [Version 0.2 Dec 28 2015 13:54:37] [Copyright (C) 1994-
↪2015,
Acroname Inc.]
```

Brainstem Network Discovery

From the discovery we found a MTM-EtherStem with serial number D272031D. We will need this number in order to preform 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.
Transferring firmware block 1, 33528 bytes

(4 of 33528 bytes (0%) of firmware block transferred.
(1028 of 33528 bytes (3%) of firmware block transferred.
(2052 of 33528 bytes (6%) of firmware block transferred.
(3076 of 33528 bytes (9%) of firmware block transferred.
(4100 of 33528 bytes (12%) of firmware block transferred.
(5124 of 33528 bytes (15%) of firmware block transferred.
(6148 of 33528 bytes (18%) of firmware block transferred.
(7172 of 33528 bytes (21%) of firmware block transferred.
(8196 of 33528 bytes (24%) of firmware block transferred.
(9220 of 33528 bytes (27%) of firmware block transferred.
(10244 of 33528 bytes (30%) of firmware block transferred.
(11268 of 33528 bytes (33%) of firmware block transferred.
(12292 of 33528 bytes (36%) of firmware block transferred.
(13316 of 33528 bytes (39%) of firmware block transferred.
(14340 of 33528 bytes (42%) of firmware block transferred.
(15364 of 33528 bytes (45%) of firmware block transferred.
(16388 of 33528 bytes (48%) of firmware block transferred.
(17412 of 33528 bytes (51%) of firmware block transferred.
(18436 of 33528 bytes (54%) of firmware block transferred.
(19460 of 33528 bytes (58%) of firmware block transferred.
(20484 of 33528 bytes (61%) of firmware block transferred.
(21508 of 33528 bytes (64%) of firmware block transferred.
(22532 of 33528 bytes (67%) of firmware block transferred.
(23556 of 33528 bytes (70%) of firmware block transferred.
(24580 of 33528 bytes (73%) of firmware block transferred.
(25604 of 33528 bytes (76%) of firmware block transferred.
```

(continues on next page)

(continued from previous page)

```
(26628 of 33528 bytes (79%) of firmware block transferred.
(27652 of 33528 bytes (82%) of firmware block transferred.
(28676 of 33528 bytes (85%) of firmware block transferred.
(29700 of 33528 bytes (88%) of firmware block transferred.
(30724 of 33528 bytes (91%) of firmware block transferred.
(31748 of 33528 bytes (94%) of firmware block transferred.
(32772 of 33528 bytes (97%) of firmware block transferred.
(33528 of 33528 bytes (100%) of firmware block transferred.
Transferred 2 blocks, 34300 total bytes
Resetting router number of device:[04:2181f0ee] with module# of router[D272031D]
Device [2181F0EE] may need a physical reset before router number can be reset.
Completed updating firmware on device [2181F0EE]
Sending Reset to device [2181F0EE]

Completed processing: Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem
 ↪Release:2.1.5]
[Copyright (C) 1994-2015, Acroname Inc.]
```

Please note that the Updater utility may appear to hang after it has transferred the firmware. This is because the device is waiting for the device to reset so that it can try and return all the previous BrainStem network settings (ie router and module addresses) for a seamless update process; however, with some updates a hard reset will be required. Simply press the reset button on the development board or power cycle the device.

4.4.5 Recovering BrainStem Firmware via CLI

In rare circumstances, a BrainStem device may become unresponsive after a failed firmware update (aka: "Bricked"). Several Acroname devices have the capability to be recovered in the field, using the `Updater` CLI utility.

Using your computer's terminal interface, navigate to the "bin" folder within the Brainstem Software Development Kit. This directory contains the *Updater CLI utility* that will be used for recovery. Downloads for all of Acroname's products can be found at Download Center under the Support tab.

USBHub3c

In order to recover a USBHub3c, you will need the following equipment on hand:

1. USB-C to USB-C cable or USB-C to USB-A cable
2. A pen or other sharp object to press the recessed Reset button.

Execute the following procedure to recover to a working firmware image:

1. Unplug everything from the USBHub3c, including all the USB-C ports and the DC power input.
2. On the bottom of the hub, press and hold the recessed Reset button.
3. While holding the reset button, insert the USB-C cable into the Control port.
4. After 5 seconds, release the Reset button.
5. Run the following updater command: `./Updater -U -t RECOVER -s USB`
6. After all the lights turn off, unplug everything from the hub.

Note: If Updater cannot find the device, try flipping the USB-C connector on the Control port and repeat the process over again.

If you are still experiencing issues, contact Acroname[105] for assistance.

USBHub3p

In order to recover a USBHub3p, you will need the following equipment on hand:

1. DC Power Supply
2. USB-B cable
3. A pen or other sharp object to press the recessed Reset button.

Execute the following procedure to recover to a working firmware image:

1. Unplug everything from the USBHub3p, including all the USB-C ports and the DC power input.
2. Connect the DC Power Supply to the hub
3. Connect the USB-B cable to the "Up0" port.
4. On the side of the hub, press and hold the recessed Reset button for at least 5 seconds, and then release.
5. *Identify the recovery Serial Port for the hub*
6. Run the following updater command:

1. Windows: `./Updater -U -t RECOVER -s COM#` where `#` is the number of the COM port
2. Mac/Linux: `./Updater -U -t RECOVER -s /dev/tty<NAME>` where `<NAME>` is the full TTY file path.

If you are still experiencing issues, contact Acroname[106] for assistance.

USBCSwitch

Note: The USBCSwitch is unable to be recovered in the field. Contact Acroname[107] for more information.

USBHub2x4

Note: The USBHub2x4 is unable to be recovered in the field. Contact Acroname[108] for more information.

105 https://acroname.com/contact-us
106 https://acroname.com/contact-us
107 https://acroname.com/contact-us
108 https://acroname.com/contact-us

4.4.6 Recovering MTM Module Firmware via CLI

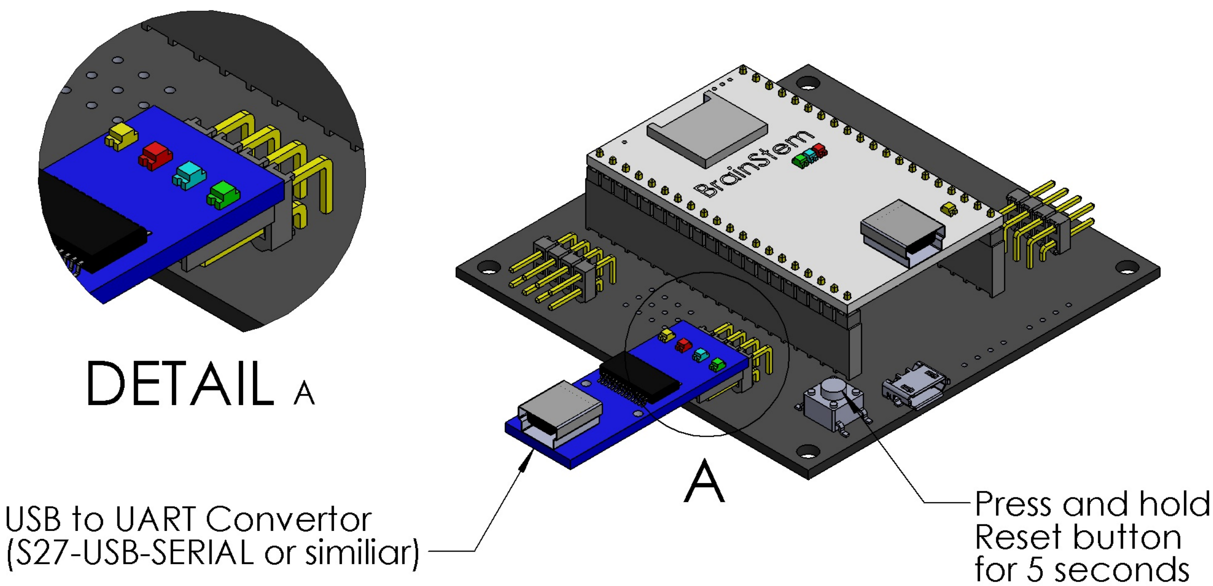
Lets take a look at how we can recover a BrainStem module after it has become unresponsive (aka: "Bricked"). This is also very helpful when dealing with old devices with a firmware version that might not be compatible with the new Updater utility.

In order to preform this recovery process you will need a USB to Serial Module[109]. 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[110].

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.

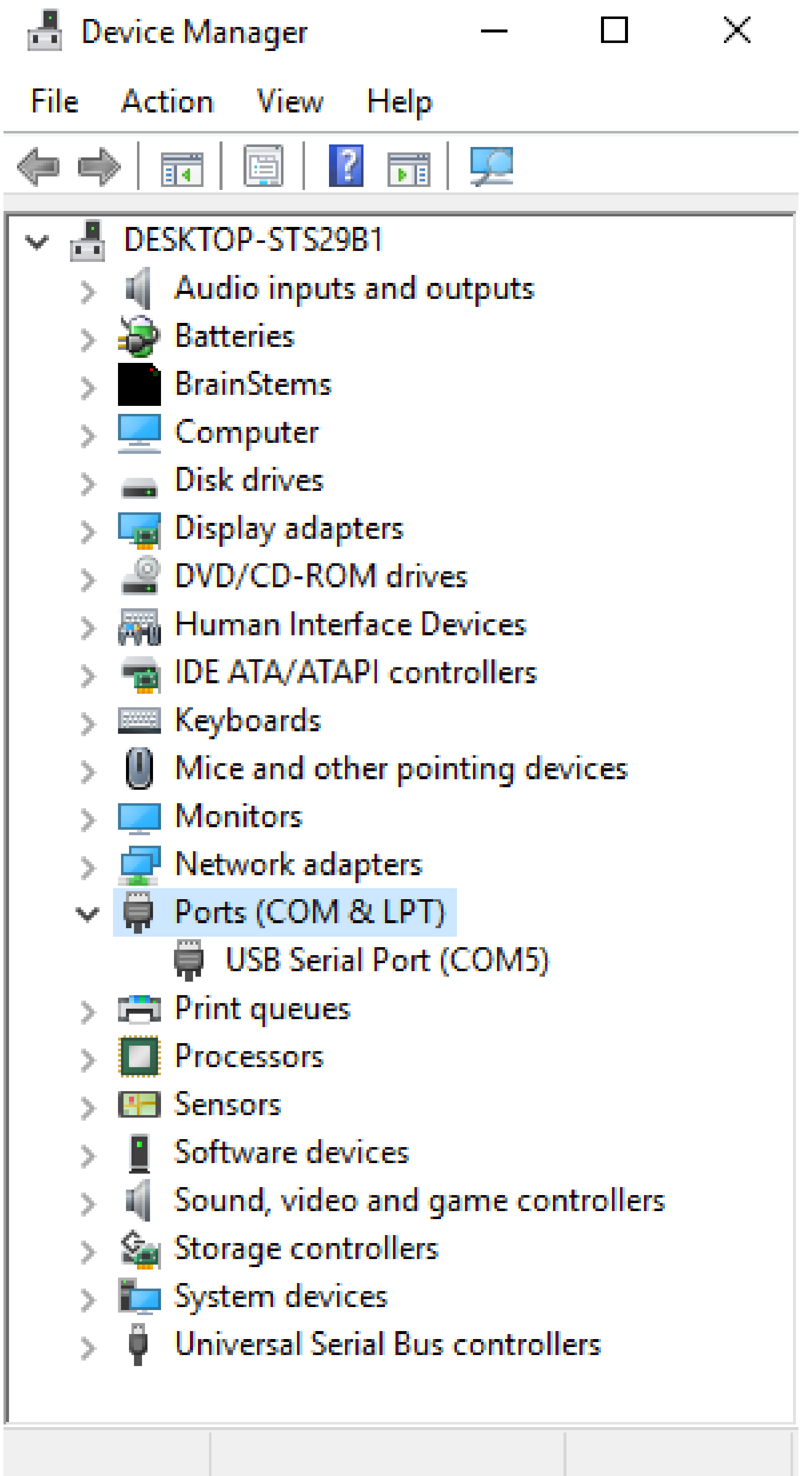


Finding your Communications Port

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).

[109] https://acroname.com/store/serial-converter-s27-usb-serial
[110] https://acroname.com/site-search/dev



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 the 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]
        [REVOCER]
        [-s]
        [/dev/tty.usbserial-A601R8QJ]
Updater [Version 0.2 Dec  9 2015 14:14:42] [Copyright (C) 1994-2015, Acroname Inc.]

Firmware_Recovery via Serial Interface:

        Sync to Device:
```

(continues on next page)

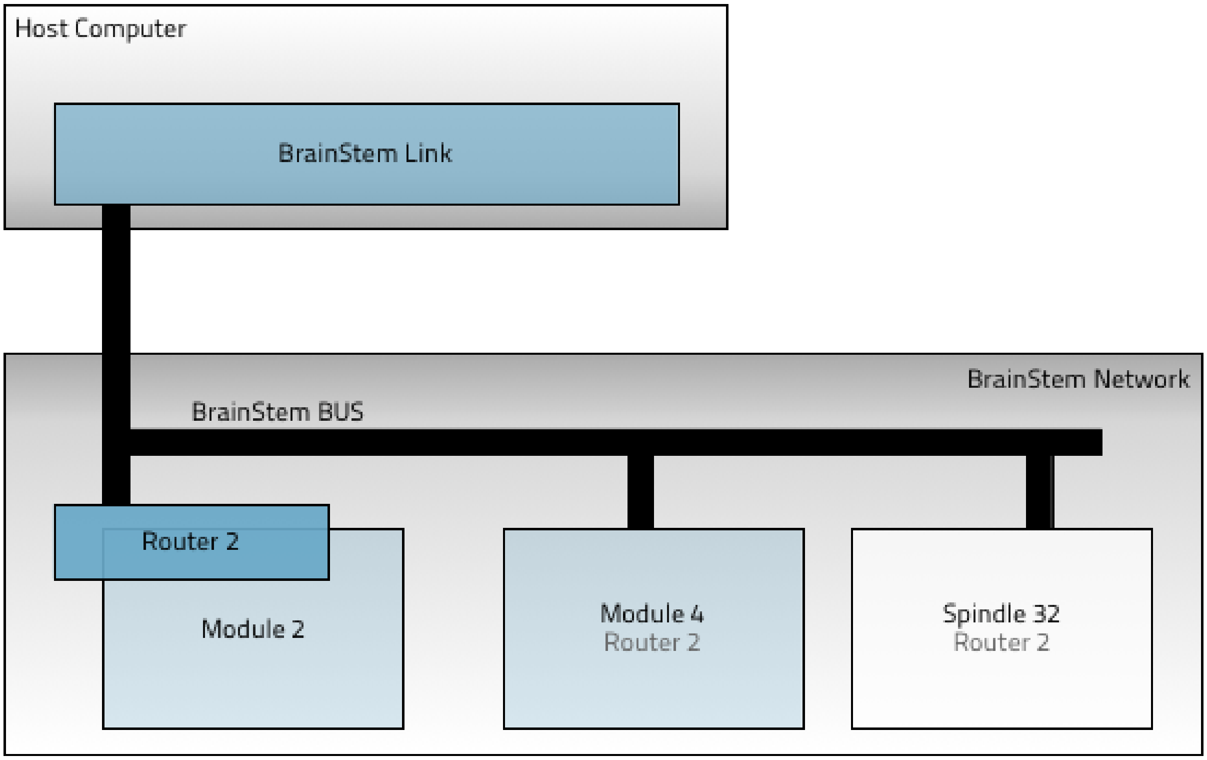
(continued from previous page)

```
        Get Info from Device:
Device responded with version: 2.4
Device responded with part number: 637615927
Device ID: [ 05005006 AE061446 508B34C6 F5001E43 ]
Device [05005006 AE061446 508B34C6 F5001E43] ==> Serial#:71E3928C, Build=130702287
GetBuild BIRD [/Users/Mitch/.acroname/updater/71E3928C/130702287.bird] VERIFIED.

        Unlock Memory:
Device ID: [ 05005006 AE061446 508B34C6 F5001E43 ]
Transferring firmware to device from [/Users/Mitch/.acroname/updater/71E3928C/
↪130702287.bird]
Transferring firmware block 0, 772 bytes
Transferred 768 of 768 bytes (100%)
Transferring firmware block 1, 49668 bytes
Transferred 4096 of 49664 bytes (8%)
Transferred 8192 of 49664 bytes (16%)
Transferred 12288 of 49664 bytes (25%)
Transferred 16384 of 49664 bytes (33%)
Transferred 20480 of 49664 bytes (41%)
Transferred 24576 of 49664 bytes (49%)
Transferred 28672 of 49664 bytes (58%)
Transferred 32768 of 49664 bytes (66%)
Transferred 36864 of 49664 bytes (74%)
Transferred 40960 of 49664 bytes (82%)
Transferred 45056 of 49664 bytes (91%)
Transferred 49152 of 49664 bytes (99%)
Transferred 49664 of 49664 bytes (100%)
Transferred 2 blocks, 50432 total bytes
Rebooting Device


END of Firmware_Recovery via Serial Interface

Completed processing: Updater [Version 0.2 Dec  9 2015 14:14:42] [Copyright (C)
1994-2015, Acroname Inc.]
```

4.5 Terminology

4.5.1 BrainStem® Network

The BrainStem is a network of devices that offer rich I/O capabilities and comprehensive interactions. The hardware comes primarily in the form of modules and that can be combined to accomplished I/O specific tasks. These hardware devices all share a common backbone network called the BrainStem Network that uses the I2C bus to exchange and route information around the system. A host or hosts can be employed using a link to then inject information or receive information from this BrainStem network of hardware modules.



4.5.2 BrainStem® Bus

The BrainStem Bus is the network backbone of the BrainStem network. For existing BrainStem modules, the bus uses I2C[111] as the hardware transport. The brainstem network is a multiple master I2C fast mode plus network. Traffic on this bus generally follows the specification of the BrainStem protocol. Third party devices can be connected to this network, but it is most common to connect I2C peripherals to the BrainStem Module's peripheral I2C ports.

4.5.3 Routing

Each module in the BrainStem Network has a unique I2C address. When a link is employed, it connects through a specific transport to one of the modules in the BrainStem Network. If the host wants to send information to a module or 3rd-party device in the network, it uses the address to send the information over the link. If the linked module is not the recipient, it acts as the router to relay the information from the link's transport to the destination module over the BrainStem Network's I2C bus.

Modules can interact with one another on the BrainStem Network as peers where each can manipulate one-another's I/O. From the I2C parlance, this means the BrainStem Network has multiple masters.

When a module needs to communicate back to the host (if present), the module can send the information to the router and the router will take care of relaying that information back across the link transport to the host.

[111] http://i2c.info/i2c-bus-specification

4.5.4 Module

Modules are the heart of the BrainStem architecture. Each is a self-contained hardware solution that employs the BrainStem OS and can communicate with other modules over the BrainStem Network. Additionally, all modules have a link transport that enables them to talk with a host computer outside the BrainStem Network. Examples of link transports a module may use are TCP/IP, Bluetooth, USB, or other industry standards. Available BrainStem modules are listed in the BrainStem Products[112] 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, I^2C 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.

[112] https://acroname.com/store-grid/field_manufacturer/acroname

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 ar 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

On Linux systems, users must run a script to properly set ownership and permissions for BrainStem devices. The script is located within the brainstem_linux_driverless folder, and is called udev.sh.

```
$> cd /path/to/brainstem_linux_driverless
$> ./udev.sh
```

Note: Executing the commands within udev.sh requires sudo privileges. You will be prompted for your login password when you execute the script. Once you execute the script, you may have to log out and back in.

In addition to setting the udev rules, Linux operating systems require system packages to be installed:

- **x86_64 Ubuntu LTS 16.04, 18.04, 20.04, 22.04, 24.04**
  - Required dependencies: *apt install xcb**
- **x86_64 Red Hat 8**
  - Required dependencies: *dnf install qt5-qtbase-gui*
- **x86_64 Red Hat 9**
  - Required dependencies: *dnf install qt6-qtbase-gui*
- **arm64v8 Ubuntu LTS 16.04, 18.04, 20.04, 22.04, 24.04**
  - Required dependencies: *apt install xcb**
- **i686 Ubuntu LTS 16.04**
  - Required dependencies: *apt install xcb**

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 that 7 this process is automatic and BrainStem devices need no install.

There is a windows_driver_installation.pdf, within the Drivers folder of the BrainStem Development Kit (BDK)[113], that describes the process for installing the WinUSB Driver on Windows 7 OS.

[113] https://acroname.com/api

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.

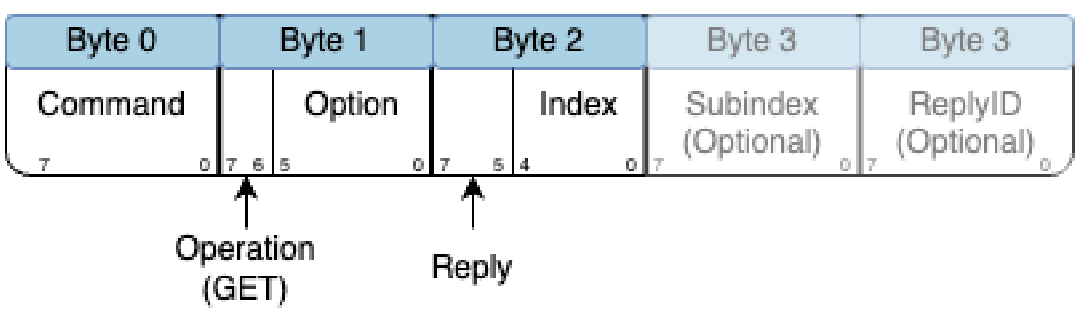


Fig. 1: Packed Byte GET UEI Structure

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 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.
- 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.

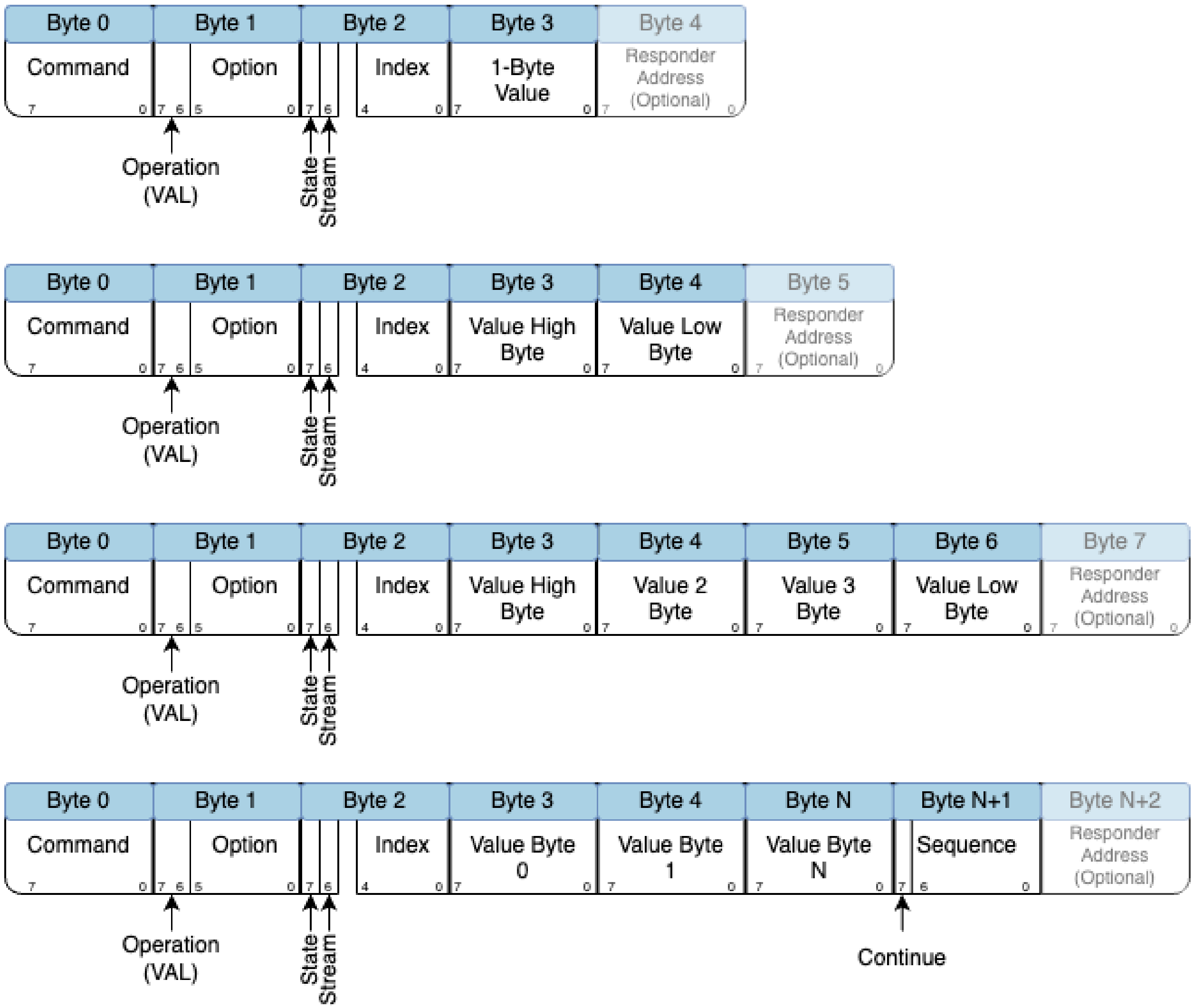


Fig. 2: Packed Byte VAL UEI Structure 1, 2, 4, and N Byte Values

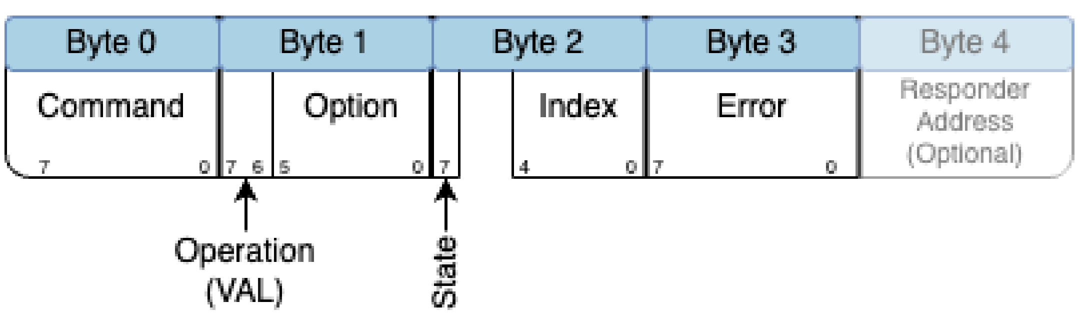


Fig. 3: Packed Byte VAL UEI Error Structure

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.

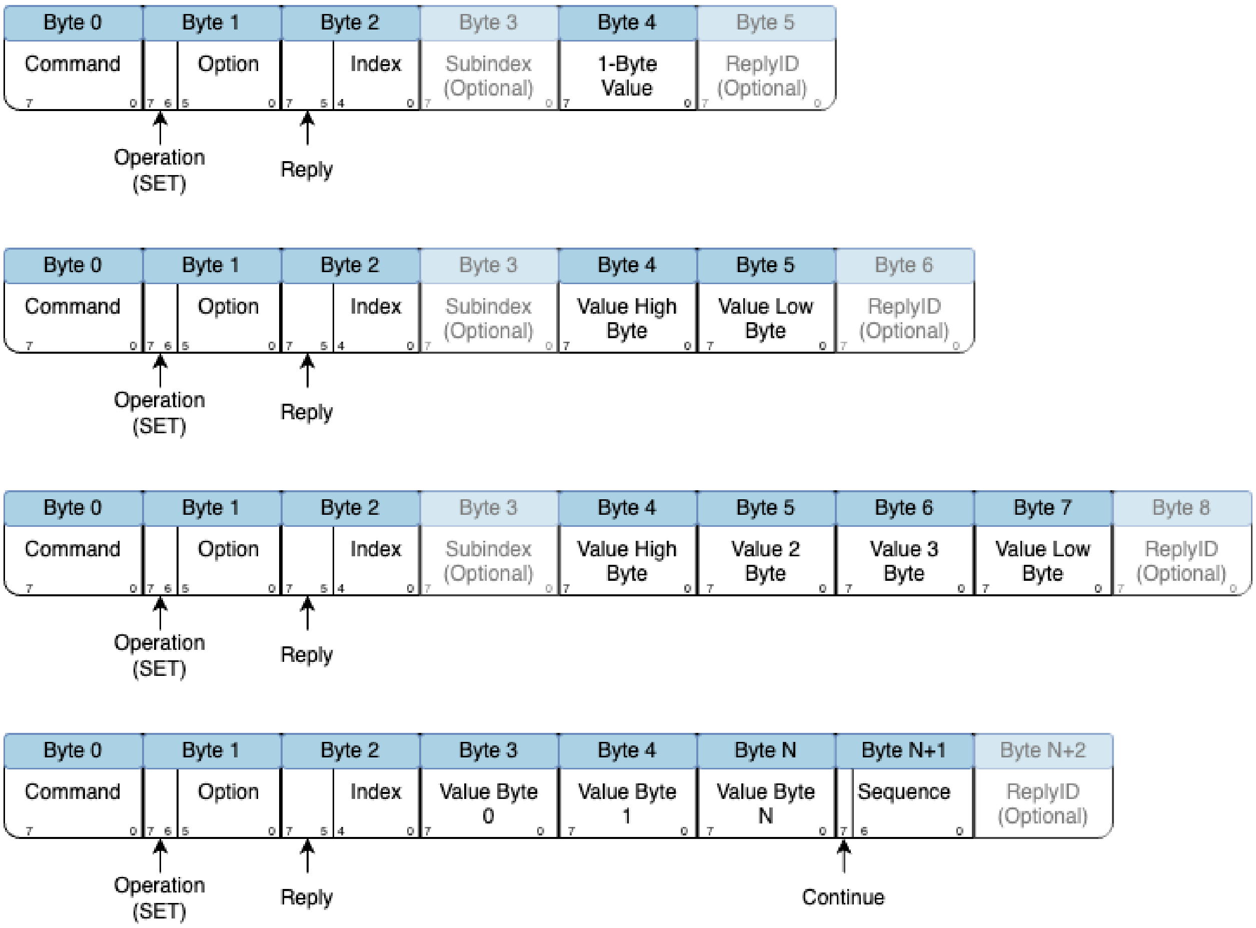


Fig. 4: Packed Byte SET UEI Structure 1, 2, 4, and N Byte Values

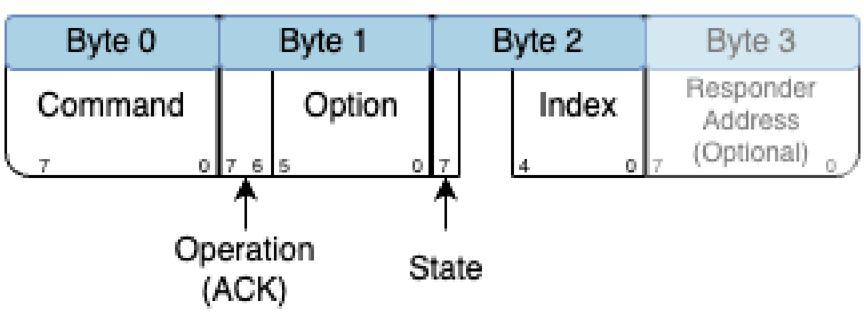


Fig. 5: Packed Byte ACK UEI Structure

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.

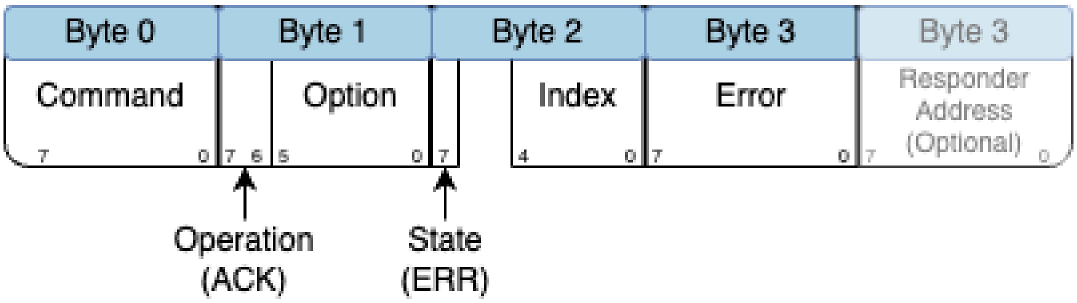


Fig. 6: Packed Byte ACK UEI Error Structure

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.

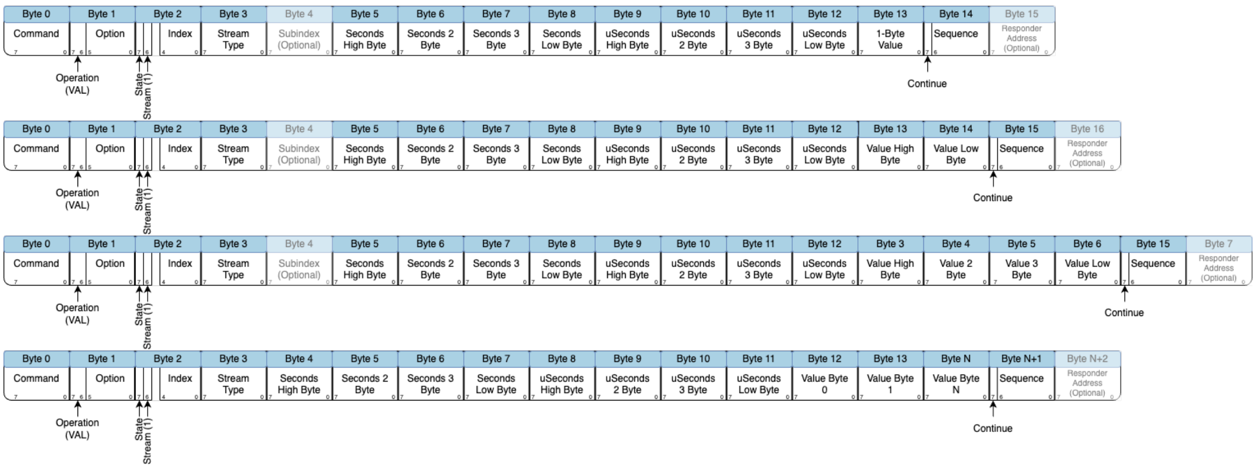


Fig. 7: Packed Byte VAL UEI Stream Structure 1, 2, 4, and N Byte Values

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.

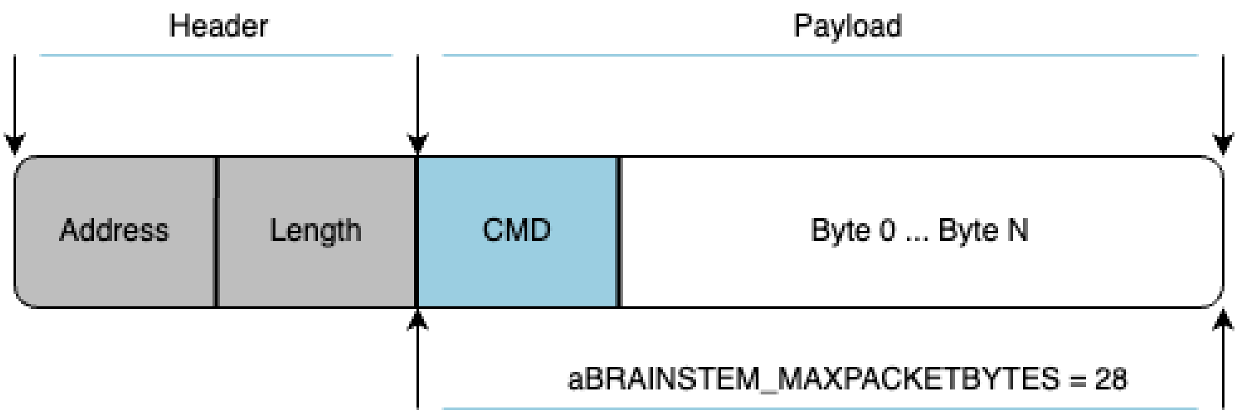


Fig. 8: Typical BrainStem Command Structure

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

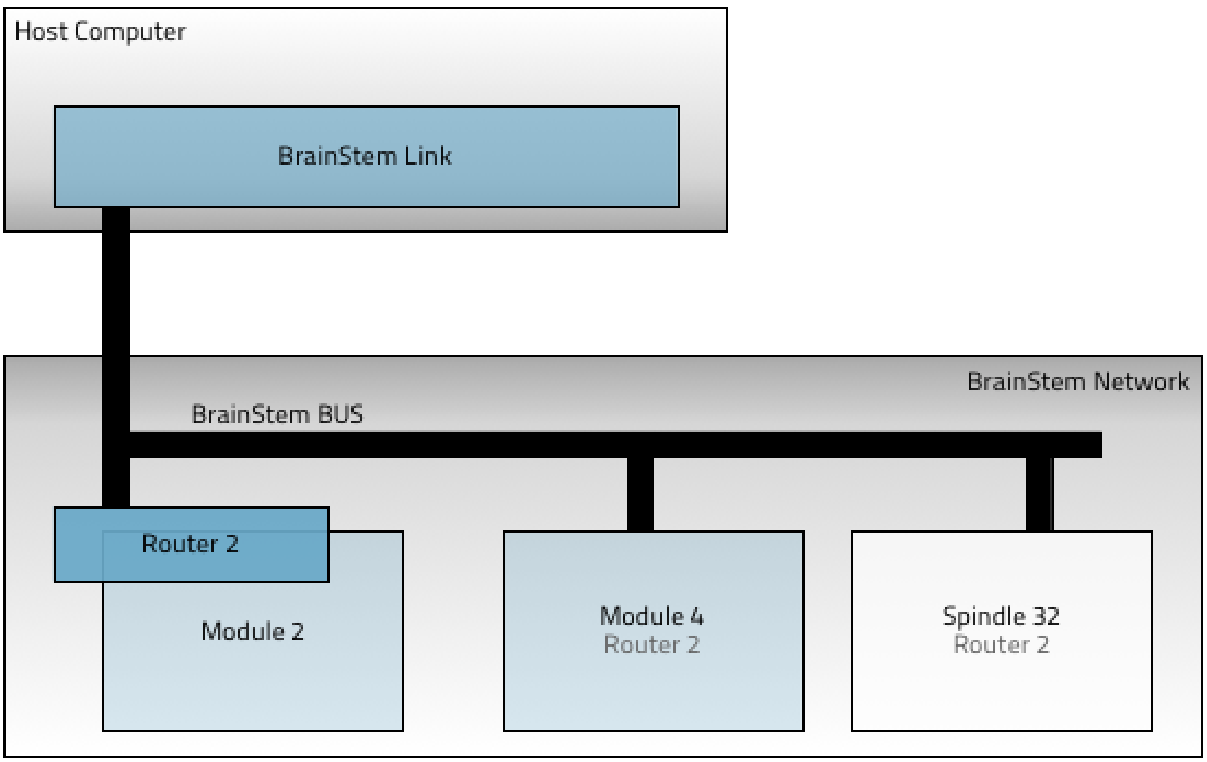
UEI's impose a request response behavior on top of the BrainStem protocol, but the protocol does not itself define a need for a response. Delivery of data is best effort. Some Link transports (TCP/IP) make guarantees about data delivery, this is not part of the BrainStem protocol. Please see specific 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 occures 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 occures 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 I2C[114] 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.



[114] http://i2c.info/i2c-bus-specification

Module Addresses

BrainStem devices rely on having a unique module address on the bus that following I2C conventions. The Brainstem module address is a single unsigned-byte, and can take even (non-odd) values from 2 to 254. Each class of BrainStem module has a specific default base address, listed in the table below. A software offset to this address can be set with the BrainStem API, and MTM modules include a set of hardware offset pins which can be used to modify module addresses with external pin connections.

BrainStem Model	Default Base Address
40Pin BrainStems EtherStem USBStem	2
MTM-EtherStem MTM-USBStem	4
MTM-PM-1 USBHub2x4 USBHub3+ USBHub3c USBCSwitch	6
MTM-IO-Serial	8
MTM-DAQ-2	10
MTM-Relay	12
MTM-Load-1	14

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 1kΩ 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

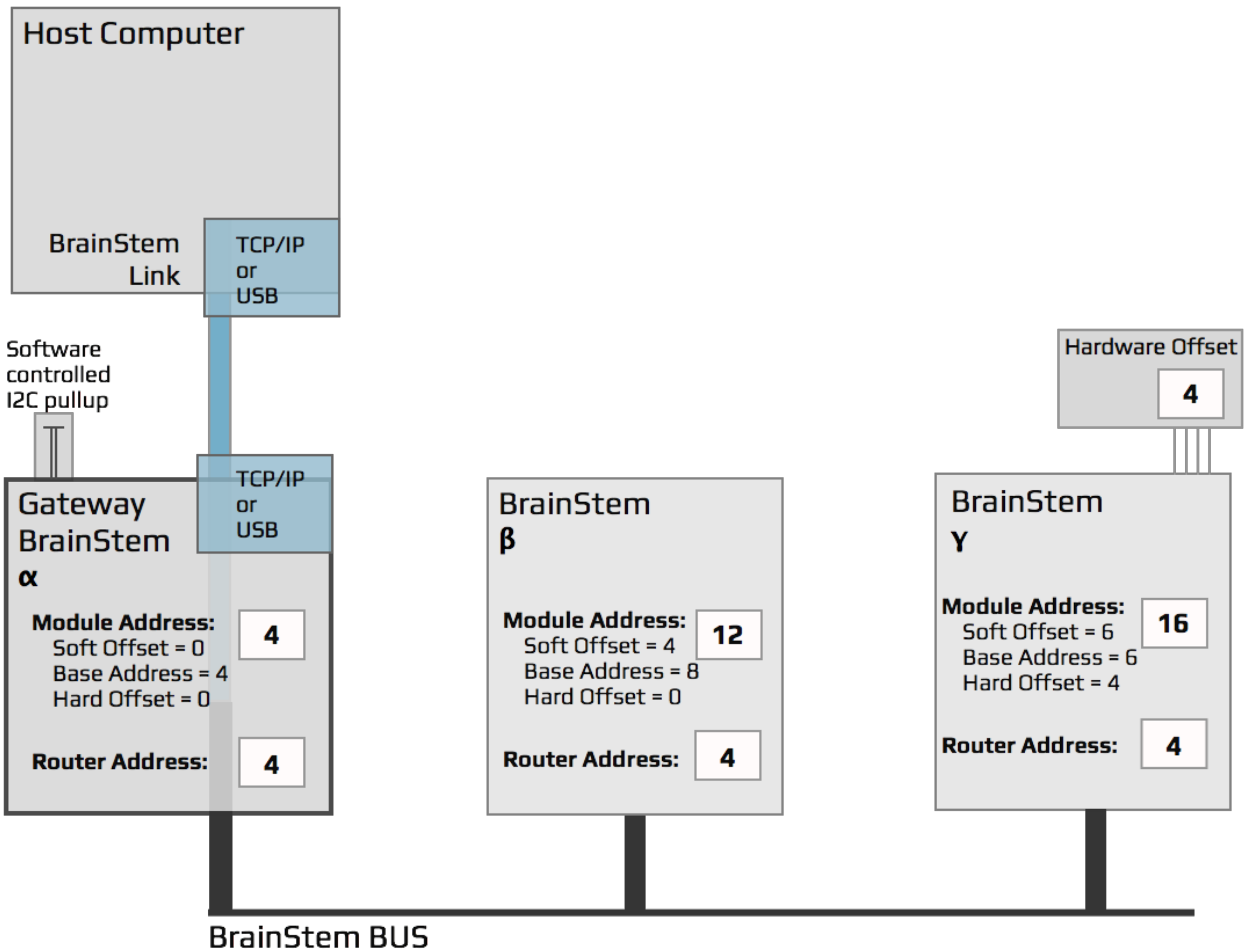
Router Addresses

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.


Host Computer
BrainStem
Link
TCP/IP
or
USB
Software
controlled
I2C pullup
Gateway
BrainStem
α
TCP/IP
or
USB
Module Address:
4
Soft Offset = 0
Base Address = 4
Hard Offset = 0
Router Address:
4
BrainStem
β
Module Address:
12
Soft Offset = 4
Base Address = 8
Hard Offset = 0
Router Address:
4
Hardware Offset
4
BrainStem
Y
Module Address:
16
Soft Offset = 6
Base Address = 6
Hard Offset = 4
Router Address:
4
BrainStem BUS

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 330Ω 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 recieving 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 and 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 the routed to address 4 on the BrainStem network, instead of going through the module's transport link connector.

The following code sets the module software offset and the router settings on γ to match the diagram. Connecting to γ is done in same way as shown above for β, simply changing the module type to the appropriate module being used.

```
>>> gamma.system.setModuleSoftwareOffset(6)
>>> gamma.system.setRouter(4)
>>> gamma.system.save()
>>> gamma.system.reset() # similarly gamma stops communicating.
>>> gamma.disconnect() # good practice to always call disconnect
```

Now the BrainStem network is configured as described in the diagram above. Moving the link cable to be connected to α will allow the entire network to show up via a single link cable. We can continue to use the same objects created earlier in this process by simply setting the module address to what was configured. This tells the host software what address the module can be reached at:

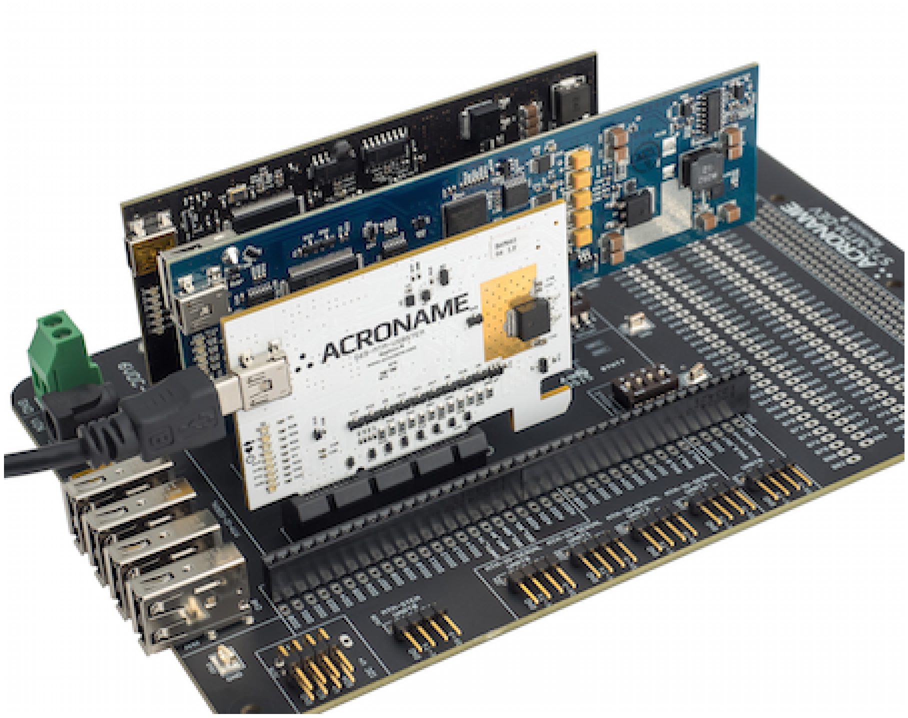
```
>>> beta.setModuleAddress(12) # Set the Module object's address so that we␣
↪communicate correctly when we reconnect.
>>> gamma.setModuleAddress(16)
```

Finally we connect directly to the Gateway Brainstem α, and then connect β and γ through α's link. We should now be able to successfully send commands and receive responses on all three brainstems through the single link connection to α.

```
# Connect to the gateway BrainStem. All three stem heartbeats should be active.
>>> alpha.connect(discover.findFirstModule(link.Spec.USB))
>>> beta.connectThroughLinkModule(alpha) # Now reconnect the beta and gamma through␣
↪the gateway module.
>>> gamma.connectThroughLinkModule(alpha)
```

In Practice

The above example is intentionally complex in order to show the interaction of hardware offsets and software offsets within a BrainStem network. In most applications and with the MTM development board, usually the user will prefer to make minimal changes in order to form the brainstem network.



In a real life use case of an MTM-USBStem acting as the Gateway, and an MTM-IO-Serial and MTM-PM-1 boards acting as the other two devices, the only changes that need to be made are to set the router address of the MTM-IO-Serial, and the MTM-PM-1 to match the MTM-USBStem base address of 4. Each of the modules have unique base addresses so there are no software or hardware offsets to apply. Also, the default object instantiation can be used without having to set the module address.

```
>>> beta.setRouter(4)
>>> gamma.setRouter(4)
>>> beta.save()
>>> gamma.save()
>>> beta.reset()
>>> gamma.reset()
>>> beta.disconnect()
>>> gamma.disconnect()
>>> beta.connectThroughLinkModule(alpha)
>>> gamma.connectThroughLinkModule(alpha)
```

4.7.4 Appendix IV: Updater File Structure

As we discussed in the *"BrainStem Firmware Management"* section, Updater is our new tool updating and recovering your Brainstem modules. After playing around with it a bit you might of noticed that it keeps a history of all the devices it interacts with. In the following appendix I will be explaining the file structure Updater creates on your in or to make things easier on you.

Locating Updater's files on your machine

Updater stores all its files in your home directory under a hidden folder named ".acroname". In Mac if you would like to find this directory you can open up your terminal window and type the following.

```
$> cd ~/              #Change to your home directory
$> ls -a              #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.

Index

A

D

E

F

G

M

N

O

P

R

S

T

U

V

W