

---

# ∴ ACRONAME

## BrainStem Reference Manual

*Release 2.9.25*

Acroname, Inc.

Dec 19, 2022



## Contents

<b>1 Overview</b>	<b>1</b>
1.1 Organization . . . . .	1
1.1.1 Products . . . . .	1
1.1.2 API Reference . . . . .	1
1.1.3 BrainStem . . . . .	1
1.2 Support . . . . .	1
<b>2 Products</b>	<b>3</b>
2.1 USBHub3p . . . . .	4
2.1.1 Quick Start Guide . . . . .	4
2.1.2 Basic Example . . . . .	5
2.1.3 Indicators and Connections . . . . .	7
2.1.4 Programming Interface . . . . .	8
2.1.5 USBHub3+ Module Entities . . . . .	9
2.2 USBHub3c . . . . .	19
2.2.1 Quick Start Guide . . . . .	19
2.2.2 Basic Example . . . . .	20
2.2.3 Indicators and Connections . . . . .	22
2.2.4 Programming Interface . . . . .	24
2.2.5 USBHub3c Module Entities . . . . .	29
2.2.6 USBHub3c Software Features . . . . .	49
2.3 USBHub2x4 . . . . .	65
2.3.1 Quick Start Guide . . . . .	65
2.3.2 Basic Example . . . . .	66
2.3.3 Indicators and Connections . . . . .	69
2.3.4 Programming Interface . . . . .	70
2.3.5 USBHub2x4 Module Entities . . . . .	71
2.4 USB-C-Switch . . . . .	80
2.4.1 Passive and Redriver Models . . . . .	80
2.4.2 Overview . . . . .	80
2.4.3 Quick Start Guide . . . . .	81
2.4.4 Basic Example . . . . .	81
2.4.5 Indicators and Connections . . . . .	84
2.4.6 Programming Interface . . . . .	85
2.4.7 USB-C-Switch Module Entities . . . . .	88
<b>3 API Reference</b>	<b>101</b>
3.1 BrainStem Entities . . . . .	102

3.1.1	Entities . . . . .	102
3.1.2	Analog Entity . . . . .	102
3.1.3	App Entity . . . . .	104
3.1.4	Clock Entity . . . . .	105
3.1.5	Digital Entity . . . . .	107
3.1.6	Equalizer Entity . . . . .	109
3.1.7	I2C Entity . . . . .	110
3.1.8	Mux Entity . . . . .	112
3.1.9	Pointer Entity . . . . .	113
3.1.10	Port Entity . . . . .	115
3.1.11	Power Delivery Entity . . . . .	119
3.1.12	Rail Entity . . . . .	123
3.1.13	RCServo Entity . . . . .	127
3.1.14	Relay Entity . . . . .	129
3.1.15	Signal Entity . . . . .	130
3.1.16	Store Entity . . . . .	132
3.1.17	System Entity . . . . .	135
3.1.18	Temperature Entity . . . . .	139
3.1.19	Timer Entity . . . . .	140
3.1.20	UART Entity . . . . .	141
3.1.21	USB Entity . . . . .	143
3.1.22	USB System Entity . . . . .	149
3.2	Python API Reference . . . . .	152
3.2.1	Getting (Quickly) Started . . . . .	152
3.2.2	Acroname Modules . . . . .	155
3.2.3	Package Structure . . . . .	173
3.2.4	Analog . . . . .	174
3.2.5	App . . . . .	178
3.2.6	Clock . . . . .	179
3.2.7	Definitions . . . . .	181
3.2.8	Digital . . . . .	182
3.2.9	Discovery . . . . .	184
3.2.10	Entities . . . . .	186
3.2.11	Equalizer . . . . .	193
3.2.12	I2C . . . . .	194
3.2.13	Link . . . . .	195
3.2.14	Module . . . . .	196
3.2.15	Mux . . . . .	202
3.2.16	Pointer . . . . .	204
3.2.17	Port . . . . .	207
3.2.18	Power Delivery . . . . .	218
3.2.19	Rail . . . . .	225
3.2.20	RCServo . . . . .	232
3.2.21	Relay . . . . .	233
3.2.22	Results . . . . .	234
3.2.23	Signal . . . . .	235
3.2.24	System . . . . .	236
3.2.25	Store . . . . .	245
3.2.26	Temperature . . . . .	246
3.2.27	Timer . . . . .	247
3.2.28	UART . . . . .	248
3.2.29	USB . . . . .	250
3.2.30	USB System . . . . .	257
3.2.31	Version . . . . .	260

3.3	C++ API Reference . . . . .	262
3.3.1	Errors . . . . .	262
3.3.2	Classes . . . . .	262
3.3.3	Acroname Modules . . . . .	266
3.3.4	Analog Class . . . . .	298
3.3.5	App Class . . . . .	301
3.3.6	Clock Class . . . . .	302
3.3.7	Digital Class . . . . .	304
3.3.8	Entity Class . . . . .	306
3.3.9	Equalizer Class . . . . .	310
3.3.10	I2C Class . . . . .	311
3.3.11	Link Class . . . . .	312
3.3.12	Module Class . . . . .	324
3.3.13	Mux Class . . . . .	328
3.3.14	Pointer Class . . . . .	330
3.3.15	Port Class . . . . .	333
3.3.16	Power Delivery Class . . . . .	342
3.3.17	Rail Class . . . . .	350
3.3.18	RCServo Class . . . . .	355
3.3.19	Relay Class . . . . .	357
3.3.20	Signal Class . . . . .	358
3.3.21	Store Class . . . . .	359
3.3.22	System Class . . . . .	361
3.3.23	Temperature Class . . . . .	369
3.3.24	Timer Class . . . . .	370
3.3.25	UART Class . . . . .	371
3.3.26	USB Class . . . . .	372
3.3.27	USBSystem Class . . . . .	380
3.4	C API Reference . . . . .	385
3.4.1	Modules . . . . .	385
3.4.2	aDefs.h . . . . .	387
3.4.3	aDiscovery.h . . . . .	388
3.4.4	Error Codes . . . . .	391
3.4.5	aFile.h . . . . .	394
3.4.6	aLink.h . . . . .	397
3.4.7	aMutex.h . . . . .	401
3.4.8	aPacket.h . . . . .	402
3.4.9	aProtocoldefs.h . . . . .	404
3.4.10	aStream.h . . . . .	431
3.4.11	aTime.h . . . . .	442
3.4.12	aUEI.h . . . . .	443
3.4.13	aVersion.h . . . . .	445
3.5	.NET API Reference . . . . .	448
3.5.1	Classes . . . . .	448
3.5.2	Errors . . . . .	450
3.5.3	BrainStem2 CLI Types . . . . .	452
3.5.4	Analog Class . . . . .	454
3.5.5	App Class . . . . .	458
3.5.6	Clock Class . . . . .	459
3.5.7	Digital Class . . . . .	461
3.5.8	Equalizer Class . . . . .	462
3.5.9	I2C Class . . . . .	463
3.5.10	Module Class . . . . .	465
3.5.11	Mux Class . . . . .	468

3.5.12	Pointer Class . . . . .	470
3.5.13	Port Class . . . . .	473
3.5.14	Power Delivery Class . . . . .	482
3.5.15	Rail Class . . . . .	490
3.5.16	RCServo Class . . . . .	495
3.5.17	Relay Class . . . . .	496
3.5.18	Signal Class . . . . .	497
3.5.19	Store Class . . . . .	499
3.5.20	System Class . . . . .	500
3.5.21	Temperature Class . . . . .	505
3.5.22	Timer Class . . . . .	506
3.5.23	UART Class . . . . .	507
3.5.24	USB Class . . . . .	508
3.5.25	USBSystem Class . . . . .	516
3.6	LabVIEW API Reference . . . . .	521
3.6.1	Entities . . . . .	521
3.6.2	Analog Entity . . . . .	523
3.6.3	App Entity . . . . .	528
3.6.4	Clock Entity . . . . .	529
3.6.5	Digital Entity . . . . .	532
3.6.6	Equalizer Entity . . . . .	534
3.6.7	I2C Entity . . . . .	535
3.6.8	Module Entity . . . . .	537
3.6.9	Mux Entity . . . . .	539
3.6.10	Pointer Entity . . . . .	542
3.6.11	Port Entity . . . . .	545
3.6.12	PowerDelivery Entity . . . . .	562
3.6.13	Rail Entity . . . . .	573
3.6.14	RCServo Entity . . . . .	581
3.6.15	Relay Entity . . . . .	582
3.6.16	Signal Entity . . . . .	583
3.6.17	Store Entity . . . . .	585
3.6.18	System Entity . . . . .	588
3.6.19	Temperature Entity . . . . .	599
3.6.20	Timer Entity . . . . .	600
3.6.21	UART Entity . . . . .	601
3.6.22	USB Entity . . . . .	602
3.6.23	USBSystem Entity . . . . .	614
3.7	Reflex Language Reference . . . . .	620
3.7.1	Introduction . . . . .	620
3.7.2	Working with Reflex files . . . . .	620
3.7.3	A Basic “Hello World” Example. . . . .	622
3.7.4	Blink My LED Example . . . . .	624
3.7.5	Built in reflex origins . . . . .	627
3.7.6	Keywords in the Reflex Language . . . . .	629
3.7.7	Operators and Precedence . . . . .	629
3.7.8	Types, Identifiers and Numbers . . . . .	631
3.7.9	The Reflex Preprocessor . . . . .	633
3.7.10	Variable Declaration . . . . .	633
3.7.11	Statements . . . . .	635
3.7.12	Reflex and Routine Definition . . . . .	637
3.7.13	Appendix . . . . .	638

4.1	BrainStem Platform . . . . .	647
4.1.1	What is BrainStem? . . . . .	647
4.1.2	Explore your module's capabilities With HubTool. . . . .	647
4.1.3	Are you up-to-date? . . . . .	647
4.1.4	Hello World Reflex . . . . .	647
4.1.5	Hello World C++ . . . . .	648
4.1.6	Next Steps . . . . .	648
4.2	What is BrainStem . . . . .	648
4.2.1	Embedded With Reflex . . . . .	648
4.2.2	Scalable . . . . .	649
4.2.3	Usable . . . . .	650
4.2.4	Next Steps . . . . .	650
4.3	Getting Started . . . . .	650
4.3.1	Do I need Drivers? . . . . .	650
4.3.2	Connecting to a BrainStem Device . . . . .	650
4.3.3	Launch HubTool . . . . .	651
4.3.4	Toggling the LED . . . . .	652
4.4	Firmware Management . . . . .	652
4.4.1	Firmware Upgrade Precautions . . . . .	652
4.4.2	Brainstem Firmware . . . . .	652
4.4.3	Firmware Update Tools . . . . .	653
4.4.4	Using Updater via CLI . . . . .	653
4.4.5	Example: Updating to the Latest Firmware . . . . .	656
4.4.6	Example: Reverting to a Previous Version . . . . .	657
4.4.7	Example: Recovering a BrainStem Module . . . . .	660
4.4.8	Example: Updating a Brainstem Module via the Brainstem Network. . . . .	664
4.5	Terminology . . . . .	668
4.5.1	BrainStem® Network . . . . .	668
4.5.2	BrainStem® Bus . . . . .	668
4.5.3	Routing . . . . .	669
4.5.4	Module . . . . .	669
4.5.5	Host . . . . .	669
4.5.6	Reflex . . . . .	669
4.5.7	Entity . . . . .	669
4.5.8	Discovery . . . . .	670
4.6	USB Drivers . . . . .	670
4.6.1	Mac OS X . . . . .	670
4.6.2	Linux Ubuntu . . . . .	670
4.6.3	Windows 7 USB Driverless Installation . . . . .	670
4.7	Appendix . . . . .	671
4.7.1	Appendix I: BrainStem Universal Entity Interface (UEI) . . . . .	672
4.7.2	Appendix II: BrainStem Communication Protocol . . . . .	678
4.7.3	Appendix III: BrainStem Networking . . . . .	680
4.7.4	Appendix IV: Updater File Structure . . . . .	687



# 1

## Overview

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

### 1.1 Organization

#### 1.1.1 Products

If you recently purchased one of our products and are looking to get started quickly the *Products* section will introduce you to your acroname device and get you up and running quickly.

#### 1.1.2 API Reference

The *API Reference* is organized by language and provides full API documentation when working with the Brain-Stem APIs and trying to answer specific questions.

#### 1.1.3 BrainStem

Acroname products are built around *BrainStem* technology. Brainstem is s a protocol, organizing principal, and operating system underlying all of our devices. If you want to learn more, check out this section.

### 1.2 Support

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

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



# 2

## Products

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

## 2.1 USBHub3p



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

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

---

### 2.1.1 Quick Start Guide

#### Power

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

## Data

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

## Download

- Download the [BrainStem Development Kit \(BDK\)](#)<sup>1</sup> for your particular operating system and architecture.

## Play

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

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

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

### 2.1.2 Basic Example

C++

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

static const uint8_t PORT = 5;

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

    //Create an instance of the USBHub3p
    aUSBHub3p hub;

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

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

    ///////////
}
```

(continues on next page)

<sup>1</sup> <https://acroname.com/software/brainstem-development-kit>

(continued from previous page)

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

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

//Disconnect
hub.disconnect();

return 0
}
```

### Python

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

PORT = 5

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

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

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

hub.usb.setPortEnable(PORT)

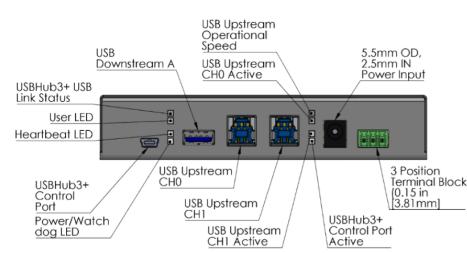
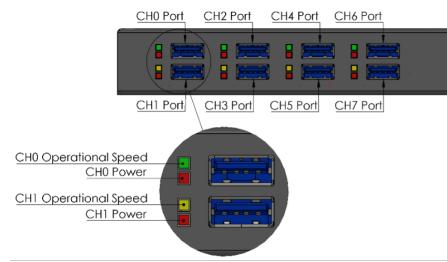
#####
#Do Stuff
#####

hub.usb.setPortDisable(PORT)

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

## 2.1.3 Indicators and Connections

### Connections



### LEDs

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

## 2.1.4 Programming Interface

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

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

---

### Software Control

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

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

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

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

### BrainStem Control Port

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

### Using Multiple Hosts with USBHub3+

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

## Device Drivers

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

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

### 2.1.5 USBHub3+ Module Entities

#### Temperature

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

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

---

#### System Temperature

The temperature of the USBHub3+ can be measured with:

```
stem.temperature[0].getTemperature(µC) [cpp] [python] [.NET] [LabVIEW]
```

where temperature is in micro-degrees Celcius.

#### System

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

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

---

## Serial Number

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

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

## Module Default Base Address

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

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

## Saved Settings

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

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

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

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

## USB

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

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

---

## USB Downstream Channel Control

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## USB Downstream Measurements

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

```
stem.usb.getPortVoltage(channel, 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 off and latch off until the port is re-enabled. An overcurrent error flag is set in getPortState() bitfield. The voltage-current behavior is detailed in Figure 6.

## USB Downstream Enumeration Speed

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

```
stem.usb.getDownstreamDataSpeed [cpp] [python] [NET] [LabVIEW]
```

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

## USB Downstream Operational Mode

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

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

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

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

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

---

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

---

## USB Downstream Enumeration Delay

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

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

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

## USB Boost Mode

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

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

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

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

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

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

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

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

## USB Hub Upstream Channels

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

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

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

The mode parameter can be defined as the following:

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

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

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

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

## USB Hub Upstream State

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

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

This command returns a 32-bit value which indicates:

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

## USB Hub Operational Mode

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

### Auto VBus Toggle

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

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

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

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

## USB Port State

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

stem.usb.getPortState(state) [\[cpp\]](#) [\[python\]](#) [\[NET\]](#) [\[LabVIEW\]](#)

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

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

## USB Port Error Status Mapping

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

stem.usb.getPortError(channel) [\[cpp\]](#) [\[python\]](#) [\[NET\]](#) [\[LabVIEW\]](#)

where channel is [0-7].

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

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

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

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

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

## Complete list of Supported Entities and Functions

Entity Class	Entity Option	Variable(s) Notes
store[0-1]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getInputCurrent	
	getVersion	
	setHBInterval	
	getHBInterval	
	getModule	
	getSerialNumber	
	getModel	
temperature[0]	getTemperature	
timer[0-8]	getExpiration	
	setExpiration	
	getMode	
	setMode	
usb[0]	setPortEnable	Channels 0-7
	setPortDisable	Channels 0-7
	setDataEnable	Channels 0-7
	setDataDisable	Channels 0-7
	setHiSpeedDataEnable	Channels 0-7
	setHiSpeedDataDisable	Channels 0-7
	setSuperSpeedDataEnable	Channels 0-7
	setSuperSpeedDataDisable	Channels 0-7
	setPowerEnable	Channels 0-7
	setPowerDisable	Channels 0-7
	getPortVoltage	Channels 0-7
	getPortCurrent	Channels 0-7
	getPortCurrentLimit	Channels 0-7
	setPortCurrentLimit	Channels 0-7
	setPortMode	Channels 0-7
	getPortMode	Channels 0-7
	getDownstreamDataSpeed	Channels 0-7
	getHubMode	
	setHubMode	
	getPortState	Channels 0-7

continues on next page

Table 2 – continued from previous page

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

## 2.2 USBHub3c



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

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

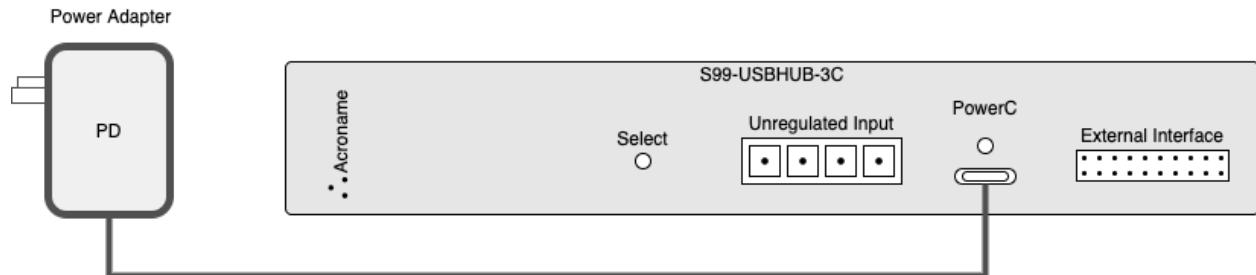
### 2.2.1 Quick Start Guide

#### 1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)<sup>2</sup> for your particular operating system and architecture.

#### 2. Connect Power

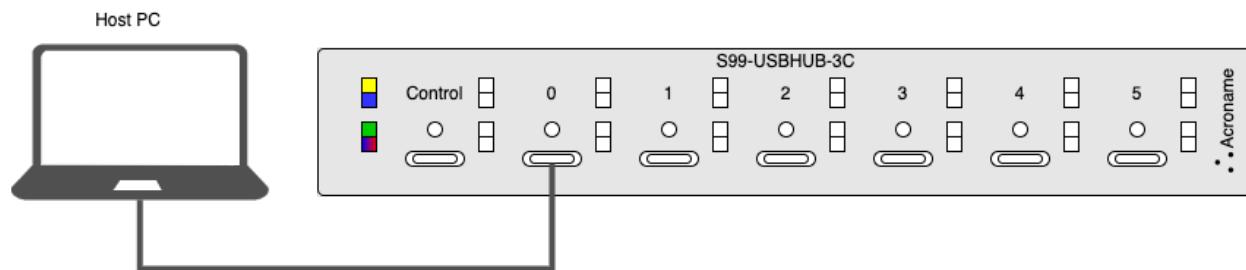
- Using the provided Power Delivery (PD) brick/wallwart and the USB 3.0 C-C cable make a connection between it and the Power-C port located on the back of the USBHub3c.
- Plug the PD Brick into a 120/240V AC outlet.



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

### 3. Connect Data

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



### 4. Play

- Unzip the downloaded package
- Navigate to the “/bin” folder
- Open HubTool
- On the bottom right side of the application select the USBHub3c device.

---

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

---

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

#### 2.2.2 Basic Example

This simple example shows briefly how instantiate, connect to, disable and re-enable a port on the USBHub3c. There are multiple examples shipped with the [BrainStem Development Kit \(BDK\)](#)<sup>3</sup> 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 {
```

(continues on next page)

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

(continued from previous page)

```

printf("Unable to discover device\r\n");
    return 1;
}

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

#####
//Do Stuff
#####

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

//Disconnect
device.disconnect();

return 0
}

```

## Python

```

import brainstem
from brainstem.result import Result
import sys

PORT = 5

#Create an instance of the USBHub3p
device = brainstem.stem.USBHub3c()

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

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

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

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

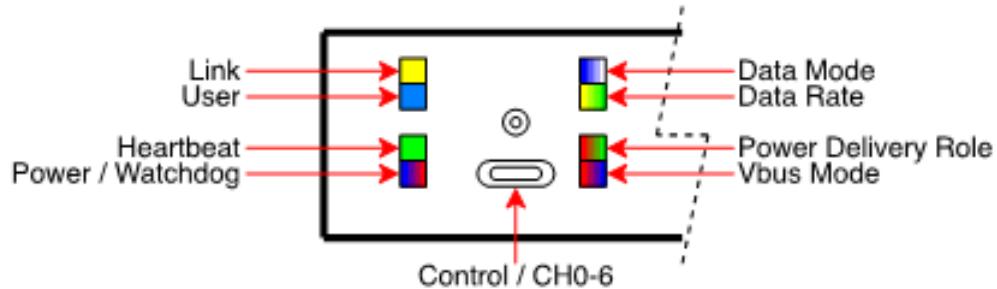
# Enable Port
device.hub.port[PORt].setEnabled(1)

#Close the connection
device.disconnect()

```

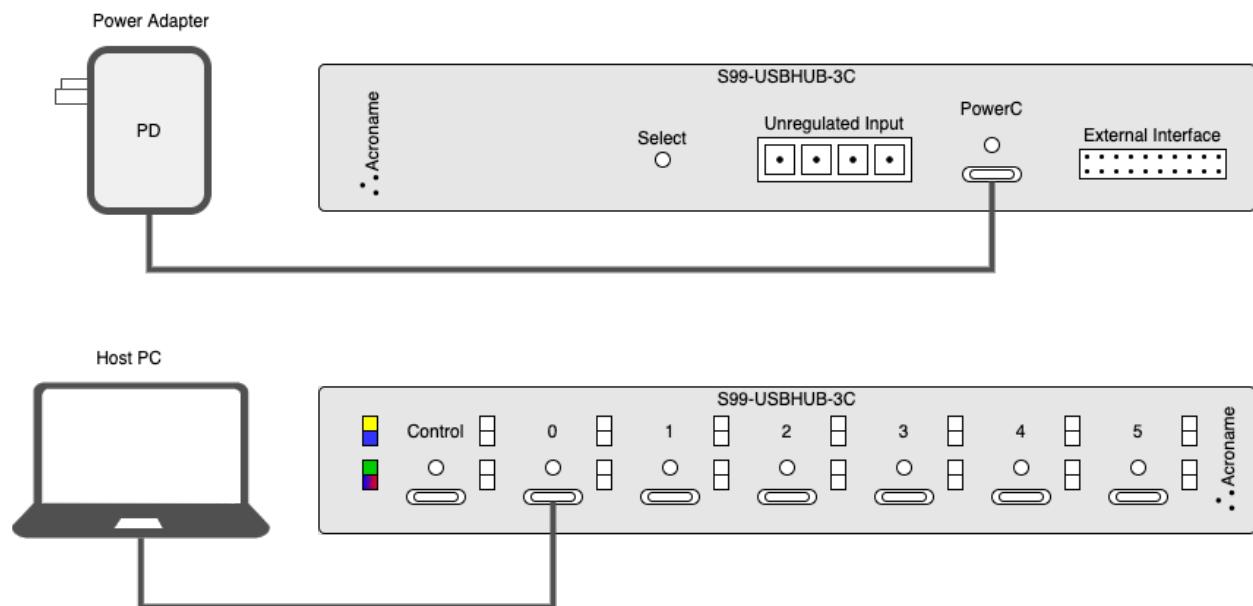
## 2.2.3 Indicators and Connections

### LEDs

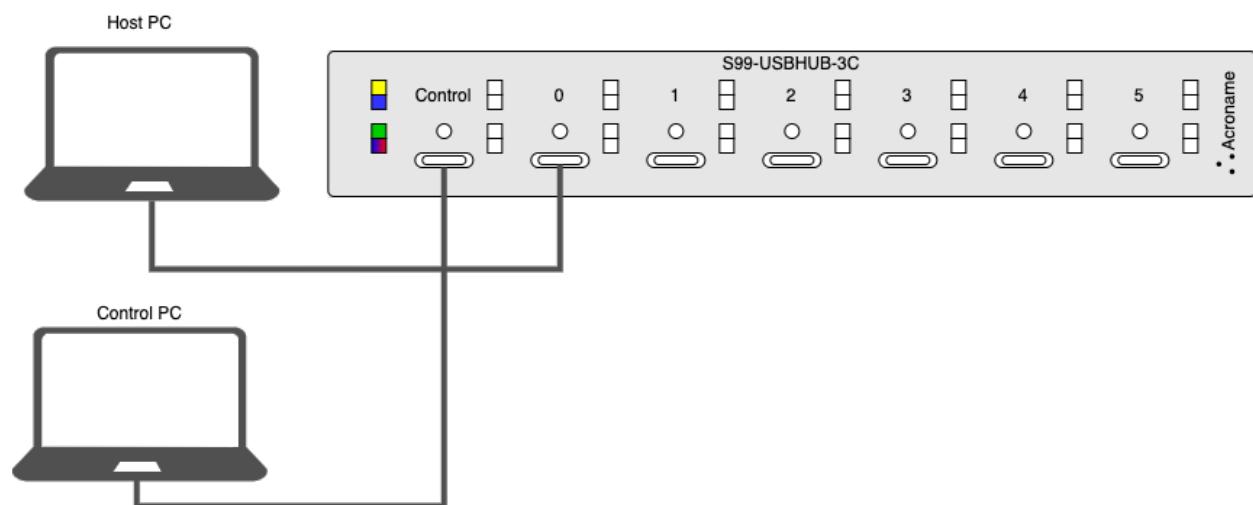


LED Name	Color	Description
Link Status LED	Yellow	On once a host device has enumerated the BrainStem controller
User LED	Blue	Can be manipulated through any of the available APIs
Heartbeat LED	Green	Indicates active BrainStem connection; pulses at a rate determined by the system heartbeat rate
Power/Watchdog LED	Red and flashing blue	Solid red indicates the system is powered. Flashing blue is indication the internal watchdog is running and the USBHub3c firmware is healthy
Data Mode	Green	Upstream Port
	Red	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

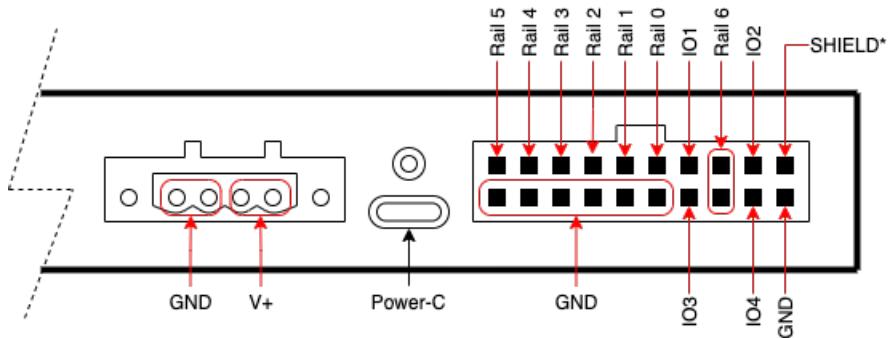
## Connections



## Separate Control Computer



## External Connector



## 2.2.4 Programming Interface

### Overview

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

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

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

### Module class

C++

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

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

(continues on next page)

(continued from previous page)

```

}

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

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

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

system.init(this, 0);

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

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

hub.init(this, 0);

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

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

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

};

```

## Python

```

class USBHub3c(Module):

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

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

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

class Hub(USBSysytem):
    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 Entities

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:

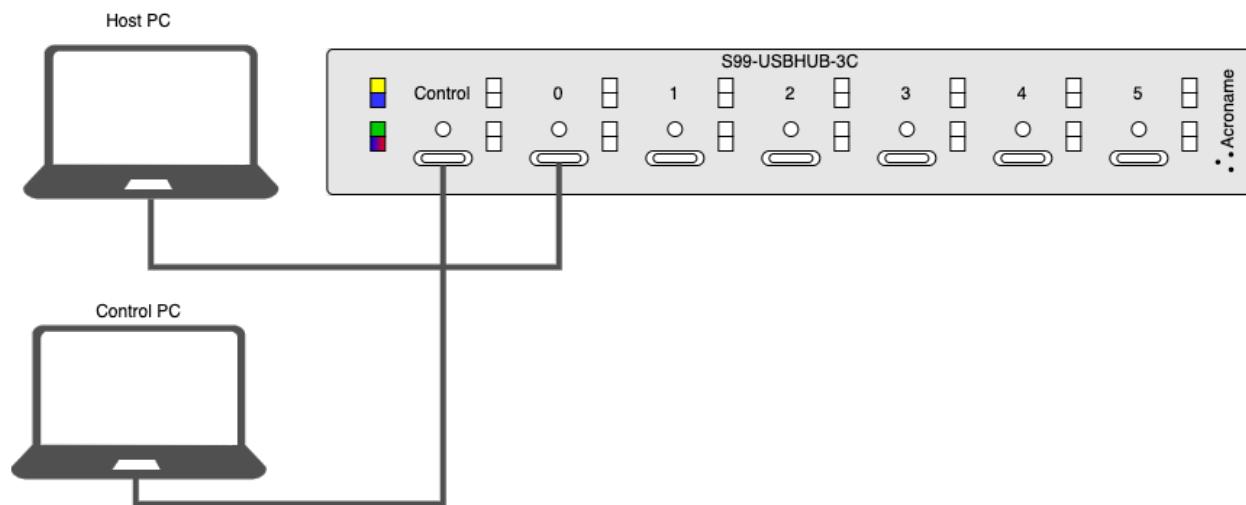
```
device.link.discoverAndConnect(USB)
```

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;

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

## BrainStem Control Port

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



## Using Multiple Hosts with USBHub3c

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

## Device Drivers

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

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

### 2.2.5 USBHub3c Module Entities

#### I2C

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

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

---

#### I2C Control

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

The I2C interface is controlled through the following APIs:

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

#### Port

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

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

---

## Port Control

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The CC lines can also be individually controlled.

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

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

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

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

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

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

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

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

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

## Voltage and Current Measurements

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

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

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

## Power Modes

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

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

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

---

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

---

## Port Mode

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

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

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

## Data Role

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

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

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

## Port Limits and Modes

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

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

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

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

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

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

## Available Power

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

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

## Accumulated Power

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

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

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

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

## Port Errors

### Power Delivery

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

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

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

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

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

Partner Type	Value	Define
Local	0	powerdeliveryPartnerLocal
Remote	1	powerdeliveryPartnerRemote

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

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

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

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

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

### Power Data Objects (PDO)

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

### Request Data Objects (RDO)

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

## Manipulating PDO's and RDO's

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

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

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

## Power Roles Preferred

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

## Connection State

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

## Requests

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

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

Request	Value	Define
Hard Reset	0	pdRequestHardReset
Soft Reset	1	pdRequestSoftReset
Data Reset	2	pdRequestDataReset
Power Role Swap	3	pdRequestPowerRoleSwap
Power Fast Role Swap	4	pdRequestPowerFastRoleSwap
Data Role Swap	5	pdRequestDataRoleSwap
Vconn Swap	6	pdRequestVconnSwap
Sink Go to Min	7	pdRequestSinkGoToMinimum
Remote Source PDOs	8	pdRequestRemoteSourcePowerDataObjects
Remote Sink PDOs	9	pdRequestRemoteSinkPowerDataObjects

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

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

### Cable Orientation

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

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

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

### Cable Type

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

## 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-degrees Celcius.

## Uart

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

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

---

## Uart Control

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

## Uart Protocols

The USBHub3c has two protocol values enumerated.

Value	Description
0	Disabled/Undefined
1	Extron Compatible Protocol

## Uart APIs

Uarts are controlled through the following APIs:

```
stem.uart[x].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 clever about how it allocates power. The method in which power is allocated across these ports is called the power behavior and it can be configured through

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

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

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

## Data Behavior

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

List of Available Data Behaviors for USBHub3c

Behavior	Value	Define
Hard Coded	0	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]
```

**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	
	getModule	
system[0]	getModuleBaseAddress	

continues on next page

Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	setRouter	
	getRouter	
	setHBInterval	
	getHBInterval	
	setLED	
	getLED	
	getVersion	
	getModel	
	getHardwareVersion	
	getSerialNumber	
	save	
	reset	
	logEvents	
	getUptime	
	getTemperature	
	getMinimumTemperature	
	getMaximumTemperature	
	getInputPowerSource	
	getInputVoltage	
	getInputCurrent	
	getModuleHardwareOffset	
	getModuleSoftwareOffset	
	getRouterAddressSetting	
	routeToMe	
	getUnregulatedVoltage	
	getUnregulatedCurrent	
	getPowerLimit	
	getPowerLimitMax	
	setPowerLimitMax	
	resetDeviceToFactoryDefaults	
temperature[0-2]	getValue	
	getValueMax	
	getValueMin	
port[0-7]	getVbusVoltage	
	getVbusCurrent	
	getVconnVoltage	
	getVconnCurrent	
	getPowerEnabled	
	setPowerEnabled	
	getPowerMode	
	setPowerMode	
	getEnabled	
	setEnabled	
	getDataEnabled	
	setDataEnabled	
	getDataHSEnabled	
	setDataHSEnabled	
	getDataHS1Enabled	
	setDataHS1Enabled	

continues on next page

Table 3 – continued from previous page

Entity Class	Entity Option	Variable(s) Notes
	getDataHS2Enabled	
	setDataHS2Enabled	
	getDataSSEnabled	
	setDataSSEnabled	
	getDataSS1Enabled	
	setDataSS1Enabled	
	getDataSS2Enabled	
	setDataSS2Enabled	
	getVconnEnabled	
	setVconnEnabled	
	getVconn1Enabled	
	setVconn1Enabled	
	getVconn2Enabled	
	setVconn2Enabled	
	getDataRole	
	getDataSpeed	
	getCCEnabled	
	setCCEnabled	
	getCC1Enabled	
	setCC1Enabled	
	getCC2Enabled	
	setCC2Enabled	
	getCCBias	
	setCCBias	
	getMode	
	setMode	
	getState	
	getCurrentLimit	
	setCurrentLimit	
	getAllocatedPower	
	getAvailablePower	
	getPowerLimit	
	setPowerLimit	
	getVbusAccumulatedPower	
	resetVbusAccumulatedPower	
	getVconnAccumulatedPower	
	resetVconnAccumulatedPower	
	getHSBoost	
	setHSBoost	
	getDataHSRoutingBehavior	
	setDataHSRoutingBehavior	
	getDataSSRoutingBehavior	
	setDataSSRoutingBehavior	
USBSystem [0]	getUpstream	
	setUpstream	
	setDataRoleBehavior	
	getDataRoleBehavior	
PowerDelivery [0-8]	getConnectionState	
	getNumberOfPowerDataObjects	

continues on next page

Table 3 – continued from previous page

<b>Entity Class</b>	<b>Entity Option</b>	<b>Variable(s) Notes</b>
	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	
	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

continues on next page

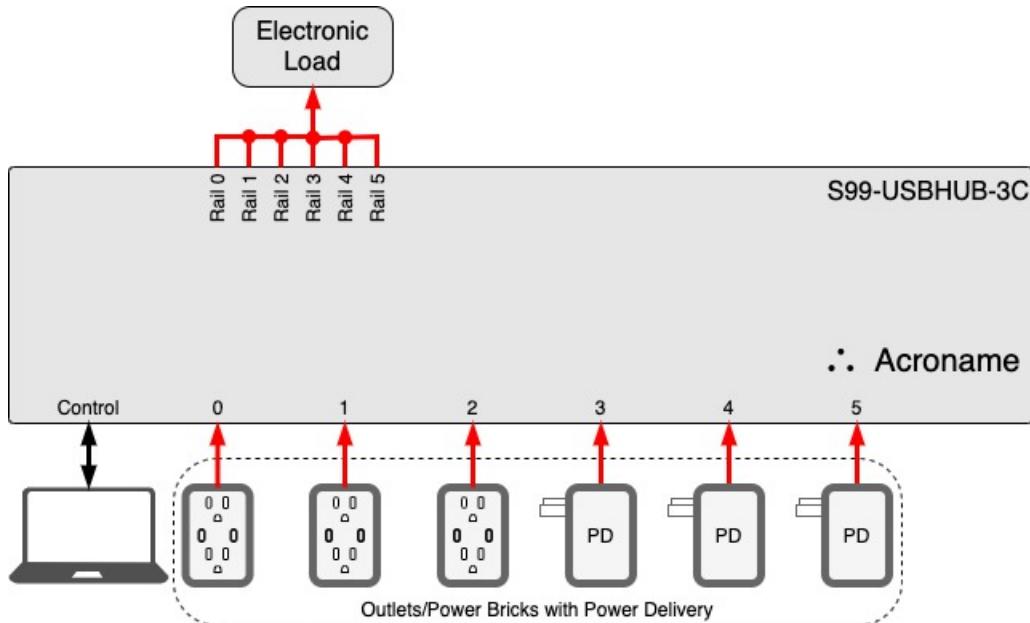
Table 3 – continued from previous page

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

## 2.2.6 USBHub3c Software Features

### External Load Testing

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



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

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

**Example:**

C++

```
static const int TEST_PORT = 1;

aUSBHub3c stem;
stem.discoverAndConnect(USB);

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

# 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 your own device
- Exposing DUTs to PDOs
- Restricting DUTs from PDOs

### Editable Items

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

## HubTool

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

The screenshot shows the HubTool application window. At the top, there are dropdown menus for 'Port' (set to 1) and 'Power Type' (set to 'Local Source Rules'). Below this is a table with columns: En, Rule, PDO Type, Min Voltage (mV), Max Voltage (mV), Power (mW), Current (mA), Raw, Set Rule, and Set Default. There are 7 rows in the table, each representing a PDO rule. Row 1: En checked, Rule 1, PDO Type Fixed, Min 5000, Max 5000, Power 3000, Current 0x3F01912C, Set, Reset. Row 2: En checked, Rule 2, PDO Type Fixed, Min 9000, Max 9000, Power 3000, Current 0x0002D12C, Set, Reset. Row 3: En checked, Rule 3, PDO Type Variable, Min 3300, Max 21000, Power 5000, Current 0x9A4109F4, Set, Reset. Row 4: En checked, Rule 4, PDO Type Battery, Min 3400, Max 21000, Power 100000, Current 0x5A410990, Set, Reset. Row 5: En checked, Rule 5, PDO Type ADPO, Min 3000, Max 21000, Power 5000, Current 0xC1A41E64, Set, Reset. Row 6: En unchecked, Rule 6, PDO Type Fixed, Min 0, Max 0, Power 0, Current 0x00000000, Set, Reset. Row 7: En unchecked, Rule 7, PDO Type Fixed, Min 0, Max 0, Power 0, Current 0x00000000, Set, Reset.

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

## Example

C++

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

aUSBHub3c stem;
stem.discoverAndConnect(USB);

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

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

//Do Stuff

stem.disconnect()
```

Python

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

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

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

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

#Do Stuff

stem.disconnect()
```

## Relevant API's

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

## PD Logging

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

### Use Cases

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

## HubTool

With HubTool you can easily visualize the Power Delivery log.

The screenshot shows the HubTool application window. At the top, there is a 'System Information' panel displaying hardware details: SN: 0x4F7A0C15, Model: 24, Firmware: 2.9.6, Module: 6. It also shows power metrics: Voltage: 20.0 VDC, Current: 0.5 A, Temperature: 30 C, and a 'Hardware Offset' section with Router: 6, SW Offset: 0, and a spinbox for adjusting it. Below this are tabs for Summary, Port 0, Port 2, Port 3, Port 4, Port 5, Control, Power C, IO Expander, Power, and PD Logging. The PD Logging tab is selected, showing a table of log entries. The table has columns for Timestamp (S:uS), Port, Direction, Spec, SOP, Power Role, Data Role, ID, Packet Type, Msg Type, and Raw hex data. The log entries show various transactions like Source Capabilities, Requests, Accepts, PS Ready, and Vendor Defined messages. At the bottom left, a status bar lists software features: Rail Enable, QC Mode, PDO Editing, VBUS Validation, and PD Logging, all enabled. It also shows the USBHub3c serial number 4F7A0C15 and the USBHub3p serial number AF62130D. The bottom right shows the device type and serial number again.

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

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

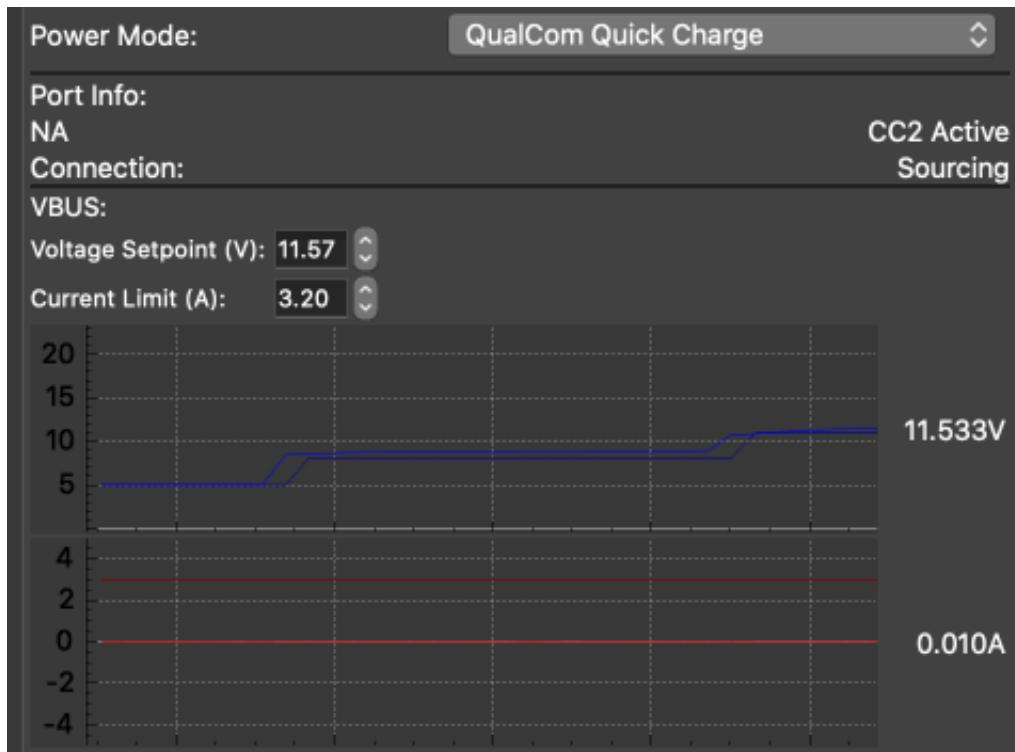
Device Type: Serial Number:  
USBHub3c 4F7A0C15  
USBHub3p AF62130D

## Quick Charge™

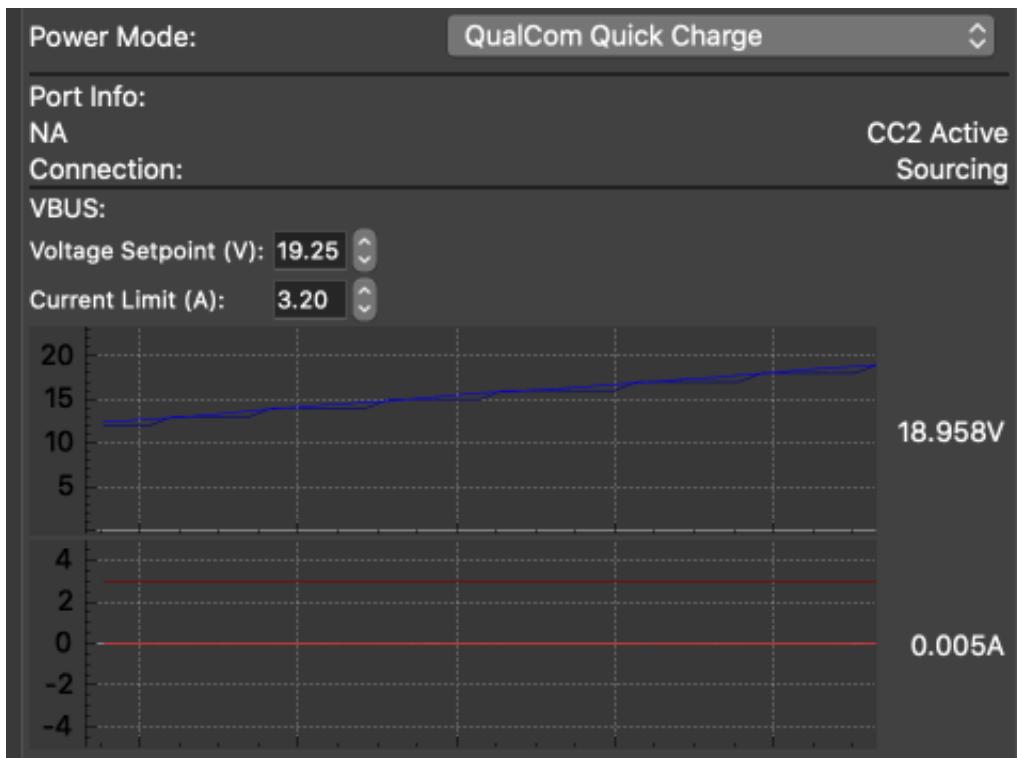
The USBHub3c Quick Charge™ feature adds the ability to enable the Quick Charge™ port power mode on one or more ports. This enables fast charging on devices that support Quick Charge™.

Version	Voltage	Current	Power
QC1	0-6.3V	2A	10W
QC2	Class A: 5V, 9V, 12V	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
```

(continues on next page)

(continued from previous page)

```
stem.disconnect()
```

## Relevant API's

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

## VBus Validation

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

### Use Cases

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

---

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

---

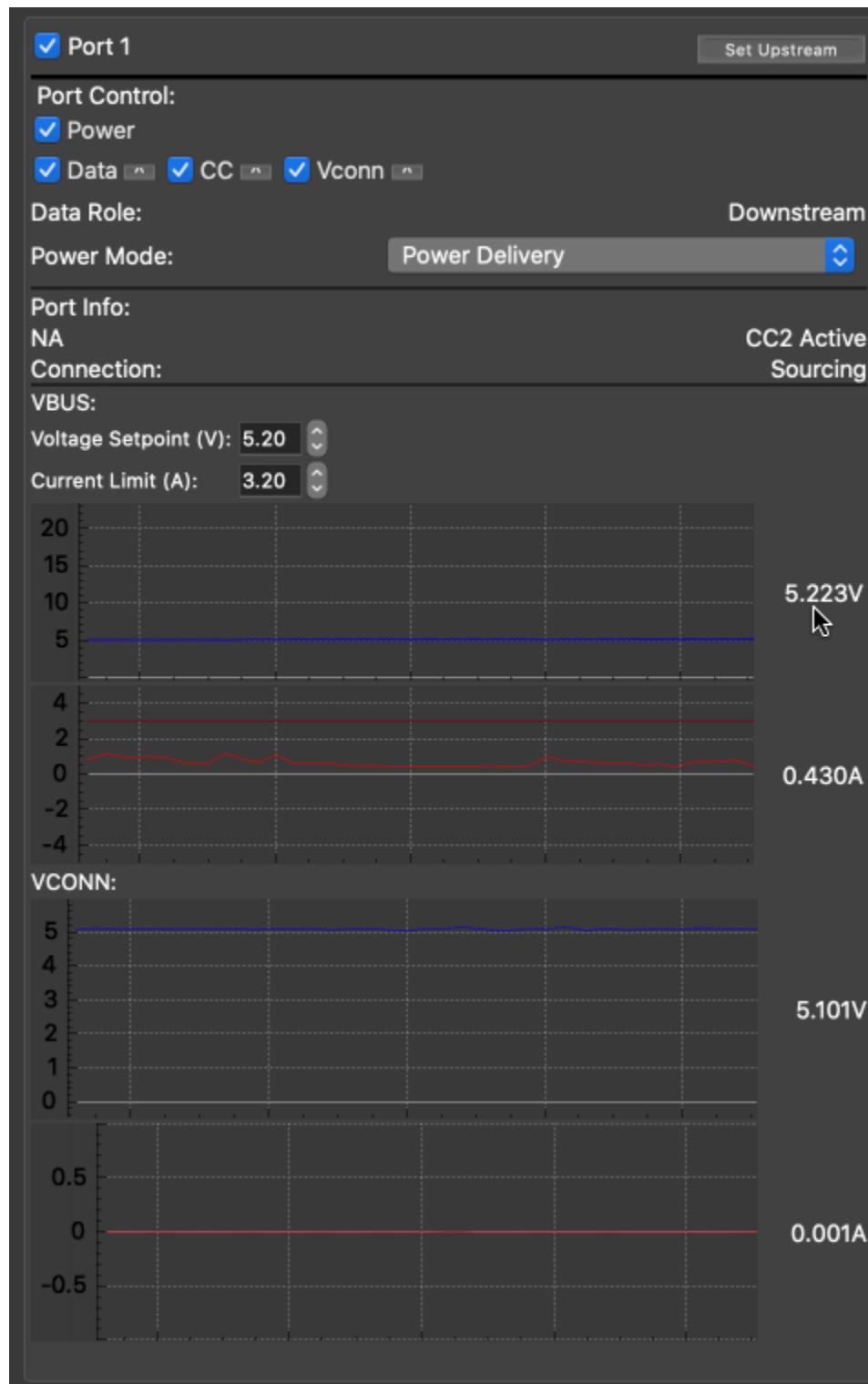
This software feature can be exploited through the *USBHub3c Port Entity*

### VBus Validation - Power Delivery Mode

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

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

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



### Example

C++

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

(continues on next page)

(continued from previous page)

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

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

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

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

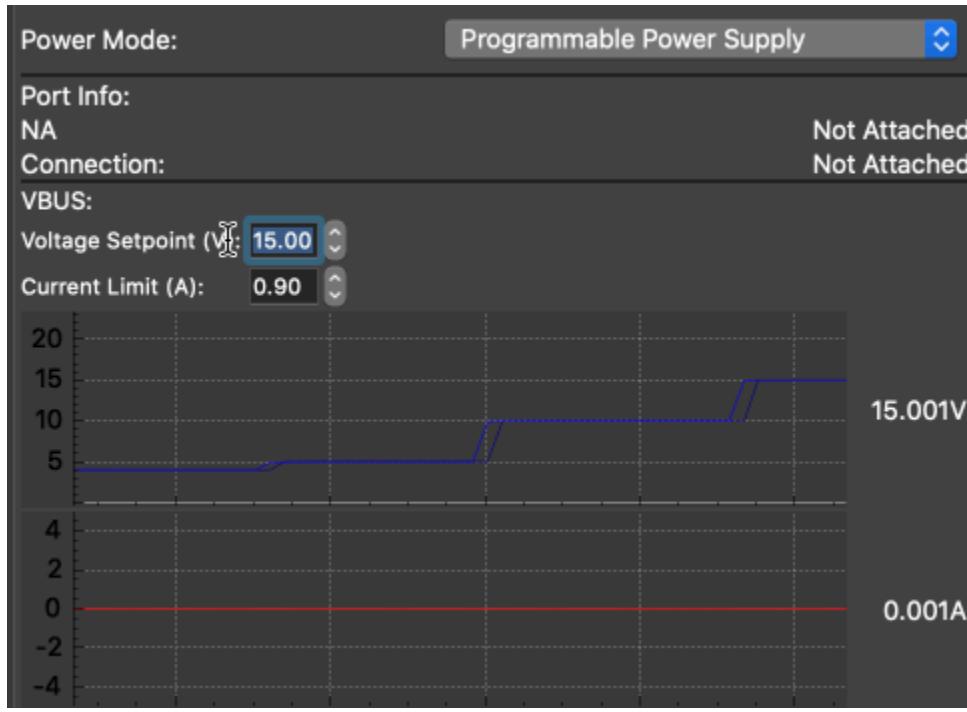
if ((connection_state_result.value == _BS_C.powerdeliveryPowerRoleSource) and
    (rdo_result.value > 0)):

    #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);

// 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();
```

(continues on next page)

(continued from previous page)

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]
```

## 2.3 USBHub2x4



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

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

### 2.3.1 Quick Start Guide

#### 1. Download The Development Kit

- Download the [BrainStem Development Kit \(BDK\)](#)<sup>4</sup> for your particular operating system and architecture.

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

## 2. Connect Power

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

## 3. Connect Data

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

## 4. Run System

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

---

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

---

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

### 2.3.2 Basic Example

C++

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

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

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

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

//Disconnect
hub.disconnect();
}

```

## Python

```

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

```

(continues on next page)

(continued from previous page)

```

import time
import sys

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

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

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

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

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

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

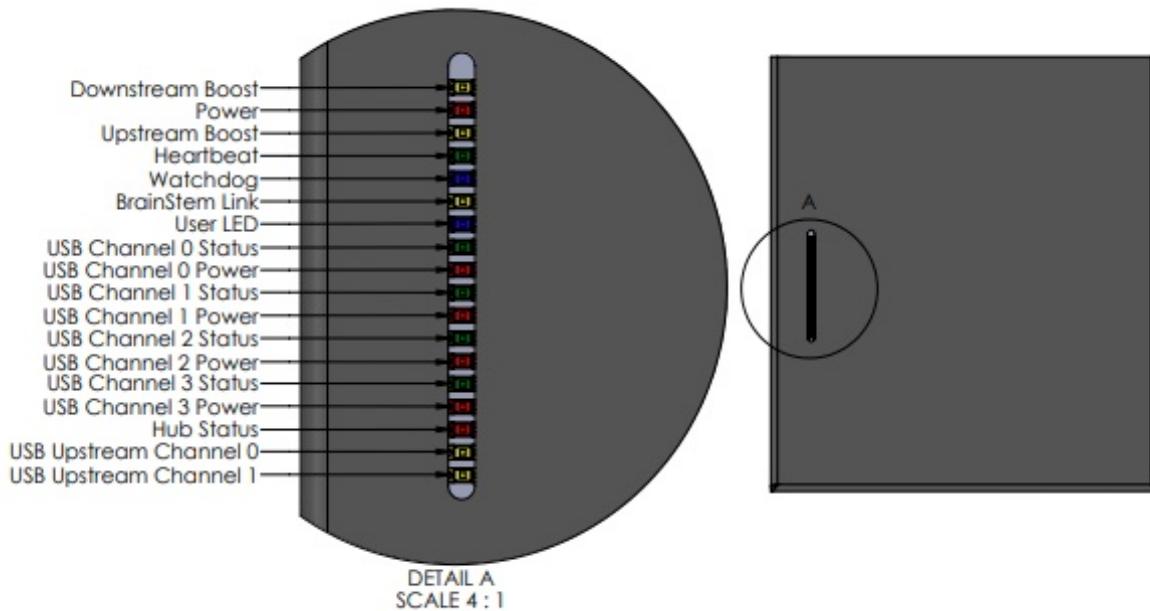
##Finished with testing.
##De-initialize.
hub.usb.setPortDisable(0)
hub.usb.setPortDisable(1)
hub.usb.setPortDisable(2)
hub.usb.setPortDisable(3)

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

```

### 2.3.3 Indicators and Connections

#### LEDs and Connections



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

### 2.3.4 Programming Interface

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

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

---

#### Software Control

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

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

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

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

## Using Multiple Hosts with USBHub2x4

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

## Device Drivers

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

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

### 2.3.5 USBHub2x4 Module Entities

#### System

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

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

---

#### Serial Number

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

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

#### Module Default Base Address

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

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

## Saved Settings

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

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

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

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

## Temperature

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

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

---

### System Temperature

The temperature of the USBHub2x4 can be measured with:

```
stem.temperature[0].getTemperature(µC) [cpp] [python] [.NET] [LabVIEW]
```

where temperature is in micro-degrees Celcius.

## USB

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

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

---

## Downstream Channel Control

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Downstream Measurements

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

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

### Hub Operational Mode

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

#### Auto Vbus Toggle

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

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

This command returns a 32-bit value which indicates:

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

## Hub Upstream Channels

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

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

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

The mode parameter can be defined as the following:

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

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

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

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

## Hub Upstream State

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

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

Value	Hub Upstream State Descriptions
0	Upstream port 0 is actively selected
1	Upstream port 1 is actively selected
2	No upstream port is selected

## Port State

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

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

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

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

## Port Error Status Mapping

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

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

where channel is [0-3].

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

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

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

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

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

## Boost Mode

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

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

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

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

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

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

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

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

### Complete list of Supported Entities and Functions

Entity Class	Entity Option	Non Standard Value
system[0]	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getInputCurrent	
	getVersion	
	setHBIInterval	
	getHBIInterval	
	getModule	
	getSerialNumber	
	getModel	
temperature[0]	getTemperature	
usb[0]	setPortEnable	Channels 0-3
	setPortDisable	Channels 0-3
	setDataEnable	Channels 0-3
	setDataDisable	Channels 0-3
	setHiSpeedDataEnable	Channels 0-3
	setHiSpeedDataDisable	Channels 0-3
	setPowerEnable	Channels 0-3
	setPowerDisable	Channels 0-3
	getPortVoltage	Channels 0-3
	getPortCurrent	Channels 0-3
	getPortCurrentLimit	Channels 0-3
	setPortCurrentLimit	Channels 0-3
	setPortMode	Channels 0-3
	getPortMode	Channels 0-3
	getDownstreamDataSpeed	Channels 0-3
	getHubMode	
	setHubMode	
	getPortState	Channels 0-3
	getPortError	
	getEnumerationDelay	
	setEnumerationDelay	
	clearPortErrorStatus	

continues on next page

Table 4 – continued from previous page

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

## 2.4 USB-C-Switch



### 2.4.1 Passive and Redriver Models

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

To get up to speed with the USB-C-Switch and quickly learn about its functionality follow the [quick start guide](#). Have a look at the [basic example](#) or dive into the [programming interface](#) of the USB-C-Switch for a more in depth view into the capabilities of each model.

### 2.4.2 Overview

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

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

### 2.4.3 Quick Start Guide

#### 1. Download The Development Kit

- Download the BrainStem Development Kit (BDK)<sup>5</sup> for your particular operating system and architecture.

#### 2. Connect Device(s)

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

#### 3. Run System

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

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

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

### 2.4.4 Basic Example

C++

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

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

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

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

    ///////////
}
```

(continues on next page)

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

(continued from previous page)

```

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

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

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

cswitch.mux.setChannel(1);

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

cswitch.mux.setChannel(2);

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

cswitch.mux.setChannel(3);

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

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

//Disconnect
cswitch.disconnect();
}

```

## Python

```

import brainstem
# For easy access to error constants
from brainstem.result import Result
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)
if result != Result.NO_ERROR:
    print ("Error %d encountered connecting to BrainStem Module.\n" % (result))
    sys.exit(1)
else:

```

(continues on next page)

(continued from previous page)

```
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)

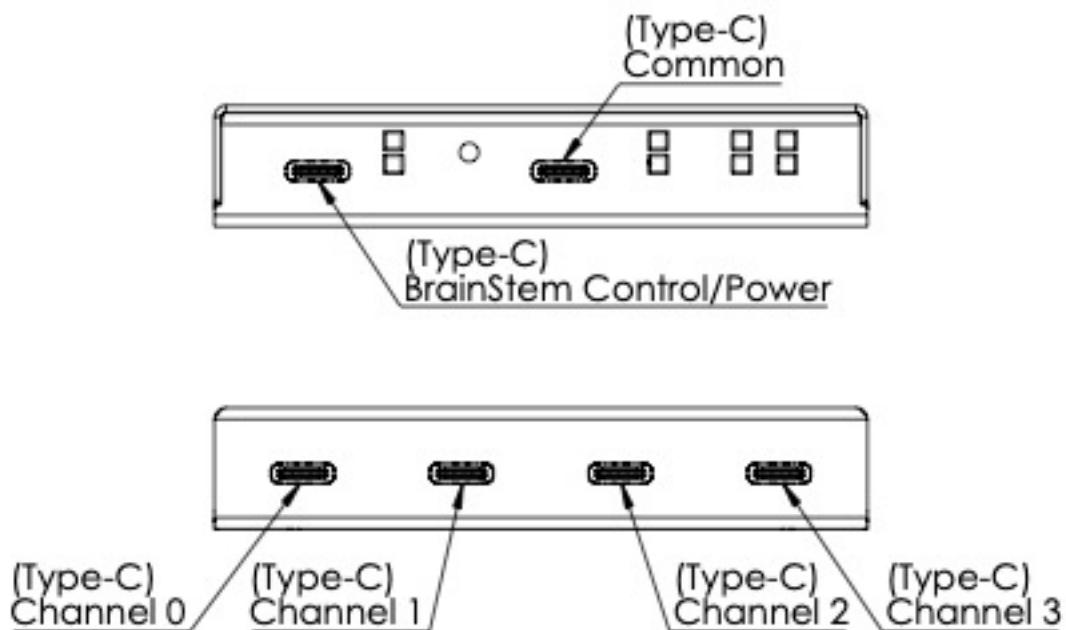
#####
##Do Stuff on Mux Channel 3
#####

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

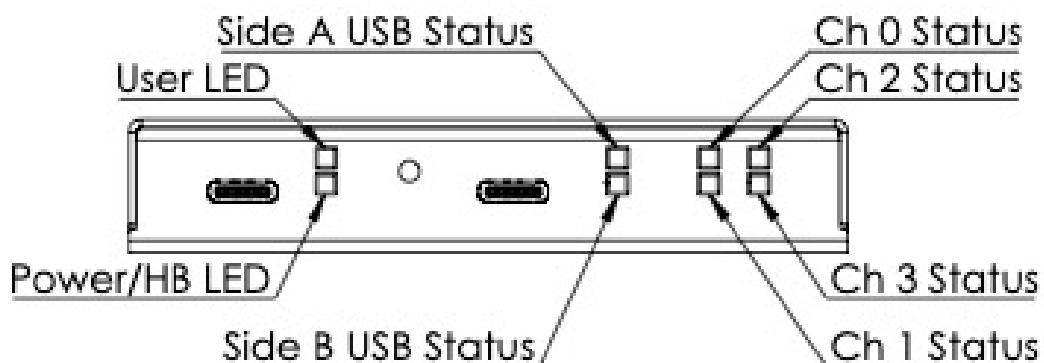
# Disconnect from device.
cswitch.disconnect()
```

## 2.4.5 Indicators and Connections

### USB Channels



### LED Indicators



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

## 2.4.6 Programming Interface

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

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

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

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

### Software Control

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

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

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

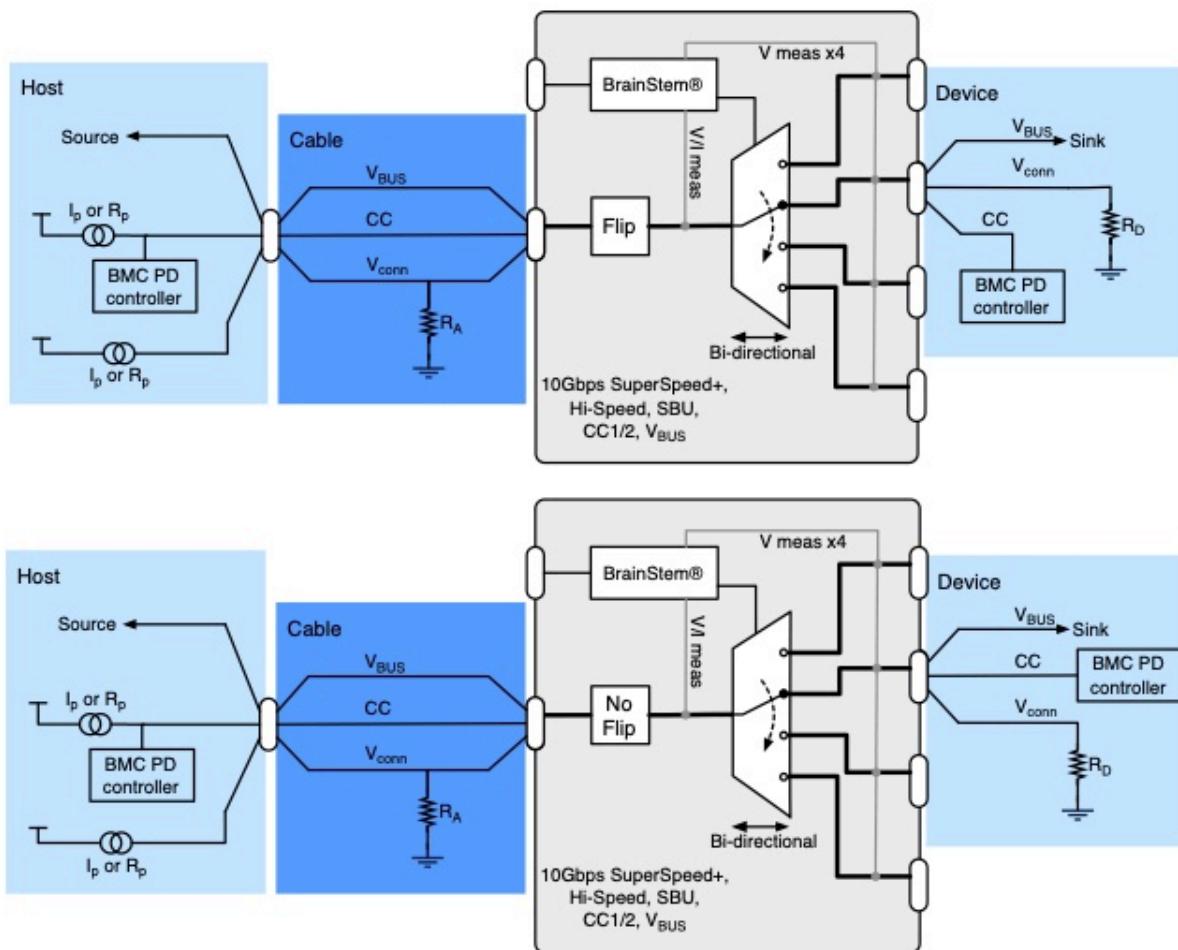
```
brainstem.discover.findAllModules(USB) (Python)
Acroname::BrainStem::Link::sDiscover() (C++)
```

## Cable Flip

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

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

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



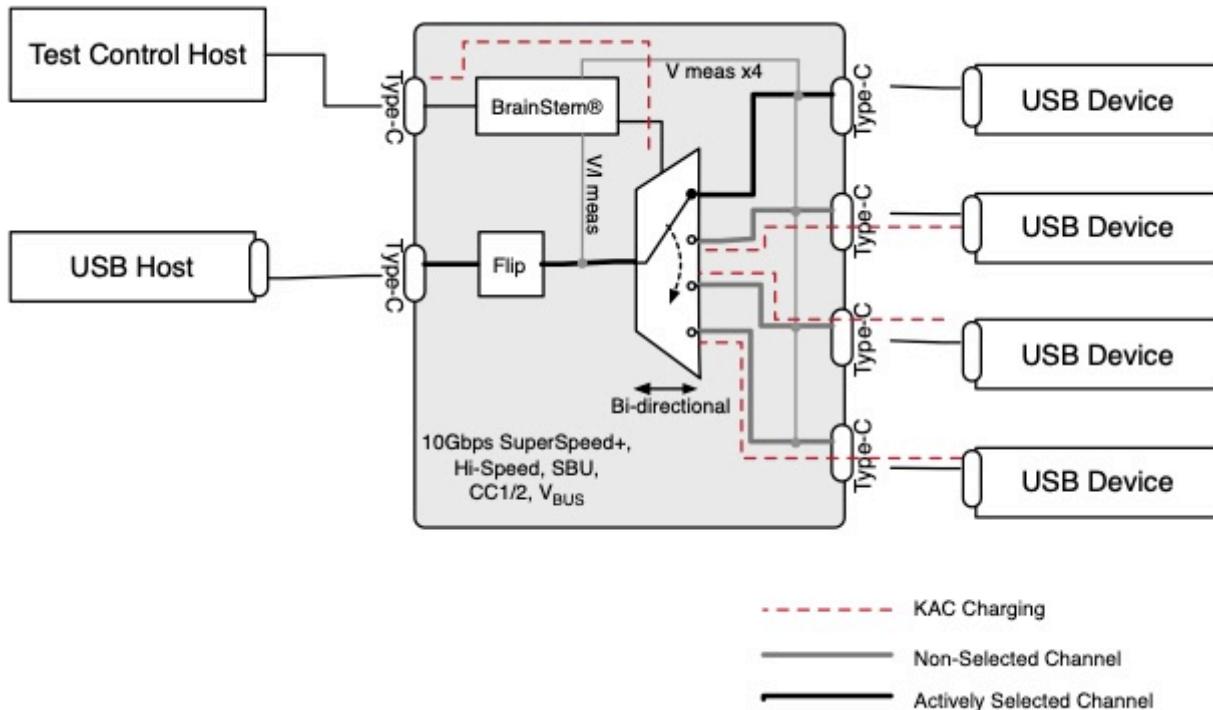
A UOC should be used on either the common port or mux port to enable automated cable flips. The UOC should be connected to the device under test.

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

## Keep-Alive Charging (KAC)

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

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



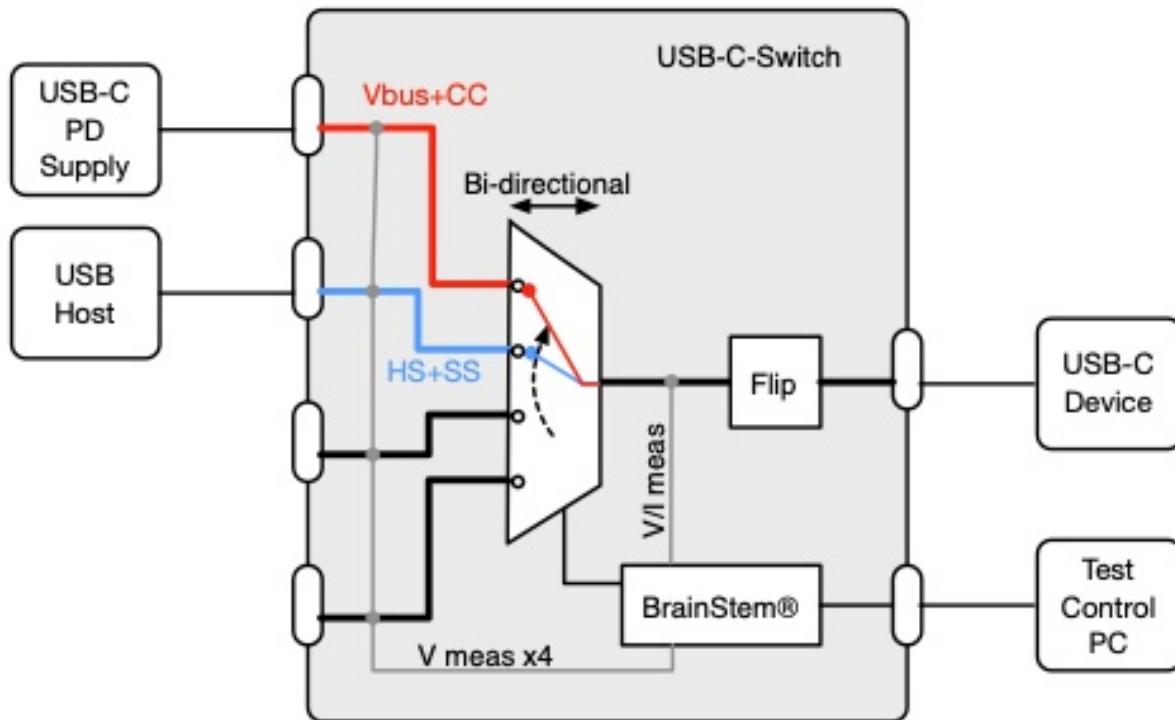
When KAC is enabled, the KAC circuit connects power from the control port VBUS to all non-selected mux channel VBUS lines. KAC power is applied only to inactive mux channels and is not applied to the actively selected mux channel since the actively selected channel has a power path to the common port.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

## Mux Split Mode

The default behavior of the USB-C-Switch is to act as a port selector, where all USB-C lines are connected between the common port and one selected mux channel. In some cases, it is desirable to split the connections in a USB-C cable and route them to different mux paths. A common application is to be able connect a USB device to a host machine for USB data while connecting VBUS charging from a device specific charger.

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

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



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

## Device Drivers

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

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

### 2.4.7 USB-C-Switch Module Entities

#### Equalizer

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

The Equalizer entity provides a concise interface for controlling equalizer and filter settings for receivers (inputs) and transmitters (outputs). Products supporting Equalizer are capable of applying frequency dependent gain to their signals. This can allow for compensation for signal loss and degradation due to cable quality, cable length and the number of connections. It can also act as a filter implemented in hardware or firmware. Products may implement one or more equalizers; each can be configured using the Equalizer index. Allowed index values are specified in the product data sheet.

---

**Note:** The Equalizer Entity is only functional on the Redriver Version of the USB-C-Switch.

---

## Equalizer Mapping and Entities

The redriver model of the switch provides two equalizer entities. They provide programmatic control over linear equalizers and amplifiers (aka: redrivers) connected to the HS and SS sata lines. These equalizer entities split the configuration between receiver-side and transmitter-side settings allowing for compensation of signal integrity loss due to cable quality, length, and insertion losses. However, some of the settings can have combined effects between receiver and transmitter modes. The two equalizer entities are indexed to their respective data lines as defined below:

Index	Equalizer Entity Mapping
0	USB2 High Speed
1	USB3 SuperSpeed

The transmitter is responsible for driving and selectively amplifying the signals traveling out the redriver hardware after any receiver-side equalization. Each equalizer entity has transmitter options of:

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

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

The receiver attempts to compensate for distortion of the incoming signal. Each equalizer entity has receiver options such as:

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

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

where (chan) parameters are defined below:

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

## High Speed Redriver Configuration

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

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

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

The HS Equalizer entity transmitter option controls the gain applied to HS signals only; USB Low Speed (LS) and Full Speed (FS) signals are unaffected and uncompensated. This option changes the DC boost applied to HS signals which can help achieve sharper rising edges. The allowed values are shown below:

Value	High Speed Transmitter Configuration
0	40mV DC Boost
1	60mV DC Boost
2	80mV DC Boost
3	0mV DC Boost (disabled)

The chan parameter of the HS equalizer 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 full-duplex SS data lines. Each configuration combines the transmitter gain and approximate peak-to-peak voltage for both the common and mux side transmitters. The available options are shown below.

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 equalizer 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:

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

## Mux

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

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexor) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything. Not every MUX has multiple inputs.

Some mux entities can simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

---

## Mux Channel

The mux entity primarily selects one active mux port to connect to the common port using the channel option:

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

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

where (channel) is an index 0-3.

## Mux Configuration

Default configuration of the mux is to switch all enabled USB-C lines to a single mux channel. If desired, the switch can split the USB-C functional group and route them to selected mux ports. This feature is referred to as "split mode". Default or split modes can be enabled with:

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

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

where (config) is 0 for default and 1 for Split Mode.

## Split Mode

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

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

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

where (splitMode) is a 32-bit word, defined below. Each bit pair is a 2-bit binary number from 0-3 representing the mux port to which to route the functional signal group. Vbus uses 4-bits to define which mux ports are connected to the common port Vbus lines.

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

## System

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

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

---

## Serial Number

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

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

## 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)	Mux Configuration (mux)
Mux Split Mode (mux)	Equalizer Configuration (equalizer)

## 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	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	Mux Port Pin Normal	Mux Port Pin Flipped
									Color Key	
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 (setting) parameter is an integer value of 0 or 1, where 0 is normal and 1 is flipped.

Individual functional group 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 of portState is used to interface with the actual state of the port.

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

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

where (mode) is a 32-bit word, defined below.

Bit	Port Mode Bit Map
0	Reserved
1	Reserved
2	Keep Alive Charging Enable
3	Reserved
4	HS Side A Data enable
5	HS Side B Data enable
6	Vbus enable
7	SS Lane 1 Data enable
8	SS Lane 2 Data enable
9:10	Reserved
11	Auto Connect enable
12	CC1 enable
13	CC2 enable
14	SBU enable
15	CC Flip enable
16	Super-Speed Flip enable
17	SBU Flip enable
18	Hi-Speed Flip enable
19	CC1 Current Injection enable
20	CC2 Current Injection enable
21:31	Reserved

## Port Operational State

The portState option provide an interface to the state of the common port and internals of the USB-C-Switch system.

```
stem.usb.getPortState(0, state) [cpp] [python] [NET] [LabVIEW]
```

where (state) is a 32-bit word, defined below.

Bit	Port State Bit Map
0	Vbus enable
1	HS Side A Data enable
2	HS Side B Data enable
3	SBU enable
4	SS Lane 1 Data enable
5	SS Lane 2 Data enable
6	CC1 enable
7	CC2 enable
8:9	Reserved
10:11	Reserved
12:13	Reserved
14	CC Flip enable
15	Super-Speed Flip enable
16	SBU Flip enable
17	Reserved
19:18	Daughter-Card status
22:20	Reserved
23	Connection Established
24:25	Reserved
26	CC1 Current Injection
27	CC2 Current Injection
28	CC1 Pulse detect
29	CC2 Pulse detect
30	CC1 Logic state
31	CC2 Logic state

## SBU Manipulation

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

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

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

## Complete List of Supported Entities and Functions

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

continues on next page

Table 5 – continued from previous page

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



## API Reference

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

## 3.1 BrainStem Entities

How does one describe the capabilities of an embedded system? Essentially this is the question that BrainStem entities answer. BrainStem modules are capable of I/O in the form of digitals, analogs, I<sup>2</sup>C, serial UARTs and other specialized interfaces. This section details how entities are referenced, and then describes the core Entities that BrainStem modules implement.

### 3.1.1 Entities

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

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

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

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

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

Indices for entities always start with a zero index.

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

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

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

- [Appendix: Brainstem Universal Entity Interface](#)
- [Appendix: The BrainStem Communication Protocol](#)

### 3.1.2 Analog Entity

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

BrainStem modules may have the ability to read an analog voltage (ADC) and convert this into a discrete digitized value or output a voltage value based on a desired discrete value (DAC). Analog voltage capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of analog entities and details for their capacities will be described in that module's datasheet.

## Value (Get/Set)

```
analog [ index ] . getValue <= (unsigned short) value
analog [ index ] . setValue => (unsigned short) value
```

### Getting Values

A BrainStem's A2D reading will always return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the measurement will get propagated up to 16 bits wide.

For example, if a 12-bit A2D engine exists in the target module's hardware, the reading will get promoted in the firmware layer by shifting up 4 bits to fill out the 16 bit value ( $0x0FFF =: 0x0FFF \ll 4 = 0xFFFF0$ ) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

### Setting Values

The reading resolution will return a 16 bit value. If the module hardware does not have full 16 bit wide analog to digital conversion capabilities, the value sent by the API will get propagated up to 16 bits wide.

For example, if a 10-bit DAC engine exists in the target module's hardware, the reading will get down shifted 5 bits to derive the 10 bit value ( $0x8000 =: 0x8000 \gg 5 = 0x0400$ ) in the module's firmware. This approach allows more portable API code to be generated independent of the target hardware.

## Configuration (Get/Set)

```
analog [ index ] . getConfiguration <= (unsigned char) configuration
analog [ index ] . setConfiguration => (unsigned char) configuration
```

### Getting Configuration

Some analog entities may be single purpose functionality or can be configured for multiple different behaviors depending on the hardware. Configuration information includes whether the entities is an input only, output only, or can be configured as either and input or output.

### Setting Configuration

Analog entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most analog entities are typically as inputs, but will vary by module hardware.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

stem.analog[0].getValue(value); // gets the value of A2D channel 0 into variable value
stem.analog[3].setValue(1234); // sets the DAC on channel 3 to a value of 1234
stem.analog[0].setConfiguration(analogConfigurationInput);
```

## Reflex

```
stem.analog[0].getValue(value); // gets the value of A2D channel 0 into variable value
stem.analog[3].setValue(1234); // sets the DAC on channel 3 to a value of 1234
```

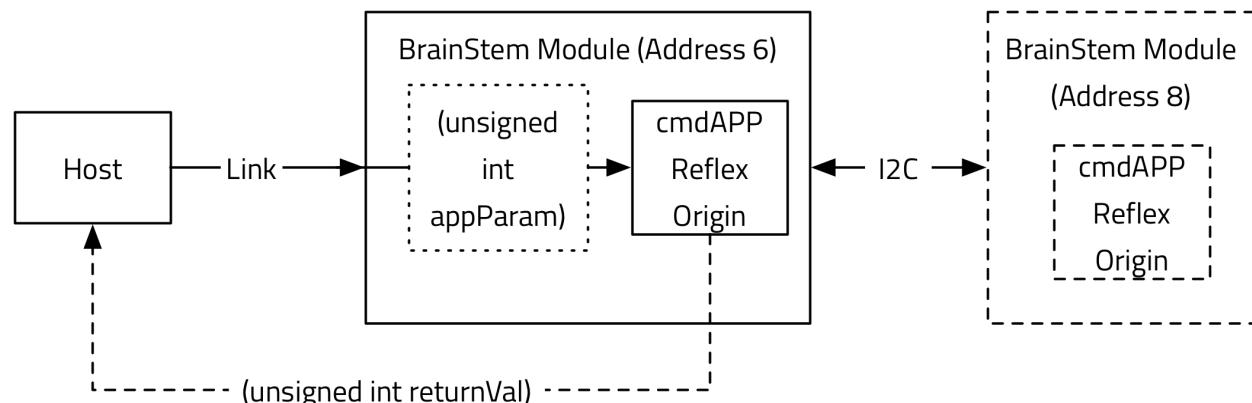
## Python

```
result = stem.analog[2].getValue() # gets the value of A2D channel 2 into variable
                                 ↳ result.
print result.value
err = stem.analog[3].setValue(1234) # sets the DAC on channel 3 to a value of 1234
print err
```

### 3.1.3 App Entity

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

BrainStem modules have a unique mechanism and communication method to send host-to-stem or stem-to-stem messages that can initiate a Reflex origin to trigger if one is defined on the target module. BrainStem modules may have up to 4 different (0-3) entity app instances.



Please be aware that a Reflex file must be enabled on the target module for a call to an App entity to be successful.

#### Execute (non-blocking)

```
app[0] . execute => (unsigned int) appParam
```

This entities will pass the data specified in appParam to be passed into the Reflex handle. The 4 bytes are up to the implementor to mean what ever one wants them to be.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

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

### Reflex

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

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

### Python

Implementation comming in future release.

## 3.1.4 Clock Entity

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

BrainStem modules may have a real time clock. This capability will be listed in the product datasheet. The clock entity allows the user to set and read the real time clock.

#### Year (Get/Set)

```
clock . getYear <= (unsigned short) year
clock . setYear => (unsigned short) year
```

Gets or sets the year value of the real time clock.

#### Month (Get/Set)

```
clock . getMonth <= (unsigned char) month
clock . setMonth => (unsigned char) month
```

Gets or sets the month value of the real time clock. Valid values are 1-12.

### Day (Get/Set)

```
clock . getDay <= (unsigned char) day  
clock . setDay => (unsigned char) day
```

Gets or sets the day value of the real time clock. Valid values are 1-31 depending on the month setting.

### Hour (Get/Set)

```
clock . getHour <= (unsigned char) hour  
clock . setHour => (unsigned char) hour
```

Gets or sets the hour value of the real time clock. Valid values are 0-23.

### Minute (Get/Set)

```
clock . getMinute <= (unsigned char) minute  
clock . setMinute => (unsigned char) minute
```

Gets or sets the minute value of the real time clock. Valid values are 0-59.

### Second (Get/Set)

```
clock . getSecond <= (unsigned char) second  
clock . setSecond => (unsigned char) second
```

Gets or sets the second value of the real time clock. Valid values are 0-59.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success. Get requests fill the variable with the current clock value.  
  
stem.clock.getYear(year);  
stem.clock.setYear(year);  
stem.clock.getMonth(month);  
stem.clock.setMonth(month);  
stem.clock.getDay(day);  
stem.clock.setDay(day);  
stem.clock.getHour(hour);  
stem.clock.setHour(hour);  
stem.clock.getMinute(minute);  
stem.clock.setMinute(minute);  
stem.clock.getSecond(second);  
stem.clock.setSecond(second);
```

## Reflex

```
// Get requests fill the variable with the value.

stem.clock.getYear(year);
stem.clock.setYear(year);
stem.clock.getMonth(month);
stem.clock.setMonth(month);
stem.clock.getDay(day);
stem.clock.setDay(day);
stem.clock.getHour(hour);
stem.clock.setHour(hour);
stem.clock.getMinute(minute);
stem.clock.setMinute(minute);
stem.clock.getSecond(second);
stem.clock.setSecond(second);
```

## Python

```
year = stem.clock.getYear();
stem.clock.setYear(year);
month = stem.clock.getMonth();
stem.clock.setMonth(month);
day = stem.clock.getDay();
stem.clock.setDay(day);
hour = stem.clock.getHour();
stem.clock.setHour(hour);
minute = stem.clock.getMinute();
stem.clock.setMinute(minute);
second = stem.clock.getSecond();
stem.clock.setSecond(second);
```

### 3.1.5 Digital Entity

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

BrainStem modules may have the ability to read, write or manipulate a digital pin. Digital I/O capabilities will be dictated by the module hardware being used. Module specifics that include the quantity of digital entities and details for their capacities will be described in that module's datasheet.

#### State (Get/Set)

```
digital [ index ] . getState <= (unsigned char) state
digital [ index ] . setState => (unsigned char) state
```

Gets or Sets the digital I/O Value.

For gets the digital input state will be reported in a boolean fashion. Voltage threshold tolerance details for the target module will be described in the datasheet.

For sets the digital output state will be asserted logic high or logic low. Voltage threshold details for the target module will be described in the datasheet.

## Configuration (Get/Set)

```
digital [ index ] . getConfiguration <= (unsigned char) configuration  
digital [ index ] . setConfiguration => (unsigned char) configuration
```

Gets or Sets the digital pin configuration.

Some digital entities may be single purpose functionality or can be configured for multiple behaviors depending on the hardware.

Digital entities that are capable of different operating configurations can be explicitly set to operate in a desired configuration mode when possible. Defaults for most digital entities are typically as inputs, but will vary by module hardware.

Available configurations for the digital entities:

Function	Typedef Constant (C++)	Typedef Constant (Python)	Val
Digital Input	digitalConfigurationInput	CONFIGURATION_INPUT	0
Digital Output	digitalConfigurationOutput	CONFIGURATION_OUTPUT	1
RCServo Input	digitalConfigurationRCServoInput	CONFIGURATION_RCSERVO_INPUT	2
RCServo Output	digitalConfigurationRCServoOutput	CONFIGURATION_RCSERVO_OUTPUT	3
High Z State	digitalConfigurationHiZ	CONFIGURATION_HIGHZ	4
Input Pull Up	digitalConfigurationInputPullUp	CONFIGURATION_INPUT_PULL_UP	0
Input No Pull	digitalConfigurationInputNoPull	CONFIGURATION_INPUT_NO_PULL	4
Input Pull Down	digitalConfigurationInputPullDown	CONFIGURATION_INPUT_PULL_DOWN	5
Signal Output	digitalConfigurationSignalOutput	CONFIGURATION_SIGNAL_OUTPUT	6
Signal Input	digitalConfigurationSignalInput	CONFIGURATION_SIGNAL_INPUT	7

---

**Note:** When using the High Z State configuration the pin and pull-ups are disconnected internally leaving the external pin floating. A get or set of the state will return in an error.

---

See the [RCServo Entity](#) for more information on its configuration.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success.  
  
stem.digital[0].getState(&state); // gets the current digital state for channel 0  
stem.digital[3].setState(1); // sets the digital output state to logic high on  
// channel 3  
stem.digital[0].setConfiguration(digitalConfigurationInput);
```

## Reflex

```
stem.digital[0].getState(state); // gets the current digital state for channel 0
stem.digital[3].setState(1); // sets the digital output state to logic high on
                            ↪channel 3
```

## Python

```
state = stem.digital[3].getState() # gets the value of digital channel 3 into
                                  ↪variable state
stem.digital[3].setState(1) # sets the digital on channel 3 to a logic high
```

### 3.1.6 Equalizer Entity

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

The Equalizer entity provides a concise interface for controlling equalizer and filter settings for receivers (inputs) and transmitters (outputs). Products supporting Equalizer are capable of applying frequency dependent gain to their signals. This can allow for compensation for signal loss and degradation due to cable quality, cable length and the number of connections. It can also act as a filter implemented in hardware or firmware. Products may implement one or more equalizers; each can be configured using the Equalizer index. Allowed index values are specified in the product data sheet.

#### Set/Get Transmitter Configuration

```
equalizer [ index ] . getTransmitterConfig <= (unsigned char) config
equalizer [ index ] . setTransmitterConfig => (unsigned char) config
```

The transmitter is the outgoing portion of the equalizer entity. It is responsible for generating the signal output. Generally, transmitters may have configurations which apply frequency dependent filters, broadband gain, and DC-offsets.

#### Set/Get Receiver Configuration

```
equalizer [ index ] . getReceiverConfig (unsigned char) channel <= (unsigned char) ↪
                                         ↪config
equalizer [ index ] . setReceiverConfig (unsigned char) channel => (unsigned char) ↪
                                         ↪config
```

The receiver is the incoming portion of the equalizer entity. The receiver equalizer may have configurations which apply frequency dependent filters or broadband gain. Products with more than one receiver may allow individual configuration of the receivers via the channel parameter. Allowed channel and config parameter values are specified in the product data sheet.

## Code Examples

### C++

```
//Set Transmitter and Receiver configurations
err = stem.equalizer[0].setTransmitterConfig(transmitterConfig);
err = stem.equalizer[1].setTransmitterConfig(transmitterConfig);
err = stem.equalizer[0].setReceiverConfig(eqReceiverChannel, receiverConfig);
err = stem.equalizer[1].setReceiverConfig(eqReceiverChannel, receiverConfig);

//Get Transmitter and Receiver configurations
err = stem.equalizer[0].getTransmitterConfig(&transmitterConfig);
err = stem.equalizer[1].getTransmitterConfig(&transmitterConfig);
err = stem.equalizer[0].getReceiverConfig(eqReceiverChannel, &receiverConfig);
err = stem.equalizer[1].getReceiverConfig(eqReceiverChannel, &receiverConfig);
```

### Python

```
#Set Transmitter and Receiver configurations
err = stem.equalizer[0].setTransmitterConfig(transmitterConfig);
err = stem.equalizer[1].setTransmitterConfig(transmitterConfig);
err = stem.equalizer[0].setReceiverConfig(eqReceiverChannel, receiverConfig);
err = stem.equalizer[1].setReceiverConfig(eqReceiverChannel, receiverConfig);

#Get Transmitter and Receiver configurations
result = stem.equalizer[0].getTransmitterConfig();
result = stem.equalizer[1].getTransmitterConfig();
result = stem.equalizer[0].getReceiverConfig(eqReceiverChannel);
result = stem.equalizer[1].getReceiverConfig(eqReceiverChannel);
```

## 3.1.7 I2C Entity

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

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

### Read

```
i2c [ index ] . read => (unsigned char) address, (unsigned char) length <= (unsigned char*) data
```

Reads up to 26 bytes from the i2c bus given by the index. The parameters are the I2C address of the device on the bus, and the number of bytes to read. The result is the data that was read or an error.

## Write

```
i2c [ index ] . write => (unsigned char) address, (unsigned char) length, (unsigned_
→char*) data <= (unsigned char) result
```

Writes up to 26 bytes to the i2c bus given by the index. The parameters are the I2C address of the device on the bus, the number of bytes to write, and the data to write. The result is the result error condition or none.

## Set Pullup

```
i2c [ index ] . setPullup => (unsigned char) bool
```

Sets software controlled pullup state on modules which have software controllable pullups. This setting is saved when a call to system.save is made so that Pullup settings on bus 0 can persist.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.
char buff[2];
stem.i2c[0].read(0x42, 0x02, buff); // reads two from device with address 0x42.
char wrbuff[] = {0xBE, 0xEF};
stem.i2c[0].write(0x42, 0x02, wrbuff); // writes 0xBEEF to the device with address
→ 0x42
stem.i2c[0].setPullup(true) //enables pullup on bus 0
```

### Reflex

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

### Python

```
result = stem.i2c[0].read(0x42, 0x02) # reads two bytes from the i2c bus. The value
→ is given in result.value
print result.value
err = stem.i2c[0].write(0x42,0x02, b'\xbe\xef') # writes b'\xbe\xef' to the i2c bus.
print err
err = stem.i2c[0].setPullup(True)
print err
```

### 3.1.8 Mux Entity

#### Channel (Set/Get)

```
mux [ index ] . setChannel => (unsigned char) channel  
mux [ index ] . getChannel <= (unsigned char) channel
```

Gets/Sets the currently selected channel

#### Enable/Disable (Set/Get)

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

Enables/Disables the mux.

#### Get Channel Voltage (Get)

```
mux [ index ] . getChannelVoltage <= ((unsigned char) channel, (unsigned char)  
→ voltage)
```

Returns the voltage of the supplied channel.

### Code Examples

#### C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success. Get calls will fill the variable with the returned value.  
  
err = stem.mux[0].getChannel(&channel);  
err = stem.mux[0].setChannel(1);  
err = stem.mux[0].setEnable(1);  
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)
```

### 3.1.9 Pointer Entity

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

Access the reflex pad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read or write of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not. This allows for multiple reads of the same pad value, or reads of multi-record values, via an incrementing pointer.

#### Offset (Get/Set)

```
pointer [ index ] . getOffset <= (unsigned char) Offset
pointer [ index ] . setOffset => (unsigned char) offset
```

Gets or sets the current cursor position for the pointer.

#### Mode (Get/Set)

```
pointer [ index ] . getMode <= (unsigned char) mode
pointer [ index ] . setMode => (unsigned char) mode
```

Get or set the pointer mode, static (0 default) or incrementing (1).

#### Char (Get/Set)

```
pointer [ index ] . getChar <= (unsigned char) value
pointer [ index ] . setChar => (unsigned char) value
```

Get or set a character value into the scratchpad at the current pointer offset. This will increment the pointer by 1 byte if the pointer mode is set to increment.

## Short (Get/Set)

```
pointer [ index ] . getShort <= (unsigned short) value  
pointer [ index ] . setShort => (unsigned short) value
```

Get or set a short value into the scratchpad at the current pointer offset. This will increment the pointer by 2 bytes if the pointer mode is set to increment.

## Int (Get/Set)

```
pointer [ index ] . getInt <= (unsigned int) value  
pointer [ index ] . setInt => (unsigned int) value
```

Get or set an int value into the scratchpad at the current pointer offset. This will increment the pointer by 4 bytes if the pointer mode is set to increment.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success. Get calls will fill the variable with the returned value.  
  
stem.pointer[0].getOffset(&offset);  
stem.pointer[0].setOffset(4);  
stem.pointer[0].getMode(&mode);  
stem.pointer[1].setMode(1);  
stem.pointer[1].getChar(&value);  
stem.pointer[1].setChar(6);  
stem.pointer[1].getShort(&value);  
stem.pointer[1].setShort(600);  
stem.pointer[1].getInt(&value);  
stem.pointer[1].setInt(600000);
```

### Reflex

```
//Get calls will fill the variable with the returned value.  
  
stem.pointer[0].getOffset(offset);  
stem.pointer[0].setOffset(4);  
stem.pointer[0].getMode(mode);  
stem.pointer[1].setMode(1);  
stem.pointer[1].getChar(value);  
stem.pointer[1].setChar(6);  
stem.pointer[1].getShort(value);  
stem.pointer[1].setShort(600);  
stem.pointer[1].getInt(value);  
stem.pointer[1].setInt(600000);
```

## Python

```
result = stem.pointer[0].getOffset()
print result.value
err = stem.pointer[0].setOffset(4)
result = stem.pointer[0].getMode(mode)
print result.value
err = stem.pointer[1].setMode(1)
result = stem.pointer[1].getChar()
print result.value
err = stem.pointer[1].setChar(6)
result = stem.pointer[1].getShort()
result.value
err = stem.pointer[1].setShort(600)
result = stem.pointer[1].getInt()
result.value
result = stem.pointer[1].setInt(600000)
```

### 3.1.10 Port Entity

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

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

#### Port Enable/Disable (Get/Set)

```
port [index] . getEnabled <= (unsigned char) enabled
port [index] . setEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the an entire port for a provided index (Power, Data, CC and Vconn). Values either passed in or returned are treated as boolean values.

#### Power Enable/Disable (Get/Set)

```
port [index] . getPowerEnabled <= (unsigned char) enabled
port [index] . setPowerEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the power for a provided index (Vbus). Values either passed in or returned are treated as boolean values.

## Data Enable/Disable (Get/Set)

```
port [index] . getDataEnabled <= (unsigned char) enabled
port [index] . setDataEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the data lines for a provided index (High Speed (HS) and Super Speed (SS)). Values either passed in or returned are treated as boolean values.

### High Speed (HS) Data Enable/Disable (Get/Set)

```
port [index] . getDataHSEnabled <= (unsigned char) enabled
port [index] . setDataHSEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the High Speed (HS) data lines for a provided index (HS1 and HS2). Values either passed in or returned are treated as boolean values.

#### High Speed 1 (HS1) Data Enable/Disable (Get/Set)

```
port [index] . getDataHS1Enabled <= (unsigned char) enabled
port [index] . setDataHS1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the High Speed 1 (HS1) data lines for a provided index. Values either passed in or returned are treated as boolean values.

#### High Speed 2 (HS2) Data Enable/Disable (Get/Set)

```
port [index] . getDataHS2Enabled <= (unsigned char) enabled
port [index] . setDataHS2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the High Speed 2 (HS2) data lines for a provided index. Values either passed in or returned are treated as boolean values.

#### Super Speed (SS) Data Enable/Disable (Get/Set)

```
port [index] . getDataSSEnabled <= (unsigned char) enabled
port [index] . setDataSSEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Super Speed (SS) data lines for a provided index (SS1 and SS2). Values either passed in or returned are treated as boolean values.

### Super Speed 1 (SS1) Data Enable/Disable (Get/Set)

```
port [index] . getDataSS1Enabled <= (unsigned char) enabled
port [index] . setDataSS1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Super Speed 1 (SS1) data lines for a provided index. Values either passed in or returned are treated as boolean values.

### Super Speed 2 (SS2) Data Enable/Disable (Get/Set)

```
port [index] . getDataSS2Enabled <= (unsigned char) enabled
port [index] . setDataSS2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Super Speed 2 (SS2) data lines for a provided index. Values either passed in or returned are treated as boolean values.

### Vconn Enable/Disable (Get/Set)

```
port [index] . getVconnEnabled <= (unsigned char) enabled
port [index] . setVconnEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Vconn lines for a provided index (Vconn1 and Vconn2 (only one ever exists)). Values either passed in or returned are treated as boolean values.

### Vconn 1 Enable/Disable (Get/Set)

```
port [index] . getVconn1Enabled <= (unsigned char) enabled
port [index] . setVconn1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Vconn 1 lines for a provided index. Values either passed in or returned are treated as boolean values.

### Vconn 2 Enable/Disable (Get/Set)

```
port [index] . getVconn2Enabled <= (unsigned char) enabled
port [index] . setVconn2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the Vconn 2 lines for a provided index. Values either passed in or returned are treated as boolean values.

### CC Enable/Disable (Get/Set)

```
port [index] . getCCEnabled <= (unsigned char) enabled
port [index] . setCCEnabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the CC lines for a provided index (CC1 and CC2). Values either passed in or returned are treated as boolean values.

### CC 1 Enable/Disable (Get/Set)

```
port [index] . getCC1Enabled <= (unsigned char) enabled
port [index] . setCC1Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the CC 1 lines for a provided index. Values either passed in or returned are treated as boolean values.

### CC 2 Enable/Disable (Get/Set)

```
port [index] . getCC2Enabled <= (unsigned char) enabled
port [index] . setCC2Enabled => (unsigned char) enabled
```

Provides control (Set) and monitoring (Get) over the CC 2 lines for a provided index. Values either passed in or returned are treated as boolean values.

### Vbus Voltage/Current (Get)

```
port [index] . getVbusVoltage <= (unsigned int) microvolts
port [index] . getVbusCurrent <= (unsigned int) microamps
```

Provides access to the last read values of Voltage (in microvolts) and Current (in microamps) for the Vbus lines.

### Vconn Voltage/Current (Get)

```
port [index] . getVconnVoltage <= (unsigned int) microvolts
port [index] . getVconnCurrent <= (unsigned int) microamps
```

Provides access to the last read values of Voltage (in microvolts) and Current (in microamps) for the Vconn lines.

### Vbus Accumulated Power (Get/Reset)

```
port [index] . getVbusAccumulatedPower <= (unsigned int) milliwatthours
port [index] . resetVbusAccumulatedPower => (void)
```

Returns the accumulated power (energy) sank or sourced by the Vbus line for the given port in units of milliWatt-hours.

### Vconn Accumulated Power (Get/Reset)

```
port [index] . getVconnAccumulatedPower <= (unsigned int) milliwatthours
port [index] . resetVconnAccumulatedPower => (void)
```

Returns the accumulated power (energy) sank or sourced by the Vconn line for the given port in units of milliWatt-hours.

### 3.1.11 Power Delivery Entity

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

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

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

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

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

Partner Type	Value	Define
Local	0	powerdeliveryPartnerLocal
Remote	1	powerdeliveryPartnerRemote

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

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

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

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

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

#### Power Data Objects (PDO)

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

#### Request Data Objects (RDO)

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

### Connection State (Get)

```
pd[x] . getConnectionState => (unsigned char) state
```

Gets the type of connection as defined by the Power Delivery Specification. The most common connections types are: Not Attached, Sourcing and Sinking.

### Power Data Object (Get/Set)

```
pd[x] . getPowerDataObject => (unsigned int) pdo
pd[x] . setPowerDataObject <= (unsigned int) pdo
```

Gets and Sets the PDO for a given pd[x] instance, partner and power role.

For any one connection there are 4 locations in which 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 possible configurations.

### Power Data Objects Enabled (Get/Set)

```
pd[x] . getPowerDataObjectEnabled => (unsigned char) enable
pd[x] . setPowerDataObjectEnabled <= (unsigned char) enable
```

Acroname products which support this feature can selectively enable and disable its local PDO's. In that, if the local source location has 7 PDO's, the user could disable all but the first PDO from being advertised by disabling them.

### Power Data Object Enabled List (Get)

```
pd[x] . getPowerDataObjectEnabledList => (unsigned char) enableList
```

Convenience function to getPowerDataObjectEnabled. Returns a bit packed representation of the PDO enabled status.

### Request Data Object (Get/Set)

```
pd[x] . getRequestDataObject => (unsigned int) rdo
pd[x] . setRequestDataObject <= (unsigned int) rdo
```

Gets and Sets the RDO for a given pd[x] instance and partner

Set calls are only possible on a local sinking partner assuming the BrainStem device supports this feature.

### Power Role (Get/Set)

```
pd[x] . getPowerRole => (unsigned char) role
pd[x] . setPowerRole <= (unsigned char) role
```

The power role defines the type of PD connections the device supports. Devices can be disabled, sinking, sourcing or dual role ports (capable of sinking or sourcing).

### Power Role Preferred (Get/Set)

```
pd[x] . getPowerRolePreferred => (unsigned char) role
pd[x] . setPowerRolePreferred <= (unsigned char) role
```

Dual role port typically have a preference of whether they are sinking or sourcing. For instance battery powered devices typically prefer to sink power since they have a finite amount of battery power; however, many of them can source power if requested to do so.

### Cable Voltage Maximum (Get)

```
pd[x] . getCableVoltageMax => (unsigned char) voltage
```

Returns the maximum amount of voltage the attached cable is capable of handling. This information is defined in the emark of the cable and is used during PD negotiations for PDO compatibility.

### Cable Current Maximum (Get)

```
pd[x] . getCableCurrentMax => (unsigned char) voltage
```

Returns the maximum amount of current the attached cable is capable of handling. This information is defined in the emark of the cable and is used during PD negotiations for PDO compatibility.

### Cable Speed Maximum (Get)

```
pd[x] . getCableSpeedMax => (unsigned char) speed
```

Returns the maximum speed the attached cable is capable of handling. This information is defined in the emark of the cable.

### Cable Type (Get)

```
pd[x] . getCableType => (unsigned char) cable
```

Returns whether the cable is active or passive and if it is emarked.

### Cable Orientation (Get)

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

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

### Request (Set)

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

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

Table 1: Flags

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

## 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	pdFlagUSBCComPossible
Unchunked Message Support	6	pdFlagUnchunkedMessageSupport
Higher Capability	7	pdFlagHigherCapability
Capability Mismatch	8	pdFlagCapabilityMismatch
Giveback Flag	9	pdFlagGivebackFlag

Table 3: Modes

Mode	Value	Description
Disabled	0	Flag will always report 0
Enabled	1	Flag will always report 1
Auto	2	Flag will show 0 or 1 correctly according to the rest of the hubs state/config

## 3.1.12 Rail Entity

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

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

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

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

## Current (Get)

```
rail[ index ] . getCurrent <= (int) microamps
```

Returns the current consumption of the device attached to the rail. This can be a positive or negative value, and is reported in microamps.

## Current Limit (Get/Set)

```
rail [ index ] . getCurrentLimit <= (int) microamps  
rail [ index ] . setCurrentLimit => (int) microamps
```

Available on some modules, check your module datasheet. This control gets or sets the maximum current draw for the given power rail in microamps.

## Temperature (Get)

```
rail [ index ] . getTemperature <= (int) microcelsius
```

Some modules have a rail temperature measurement. This command gets the current rail temperature in microcelsius.

## Enable (Get/Set)

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

### Setting Enable

Some rails can be enabled or disabled. The enable value is treated as a boolean 1 will enable the rail and 0 will disable it. Check the module datasheet to determine if this functionality is available for the given rail.

### Getting Enable

If a rail can be enabled or disabled, getting the Enable setting will return a 1 if the rail is enabled or 0 otherwise.

## Voltage (Get/Set)

```
rail [ index ] . setVoltage => (int) microvolts  
rail [ index ] . getVoltage <= (int) microvolts
```

Some rails are variable voltage rails, and users can set the rails to supply voltage at range of voltage values. Check the module datasheet for the rail voltage limits, and settings.

### Setting Rail Voltage

Setting this value will cause the rail to supply the requested voltage, if it is within the settings defined in the datasheet.

### Getting Rail Voltage

Getting this value will return the current voltage setpoint for the rail in microvolts. If the given rail is fixed, it returns the fixed voltage setting for the given rail.

## Kelvin Sensing (Get/Set)

```
rail [ index ] . setKelvinSensingEnable => (unsigned char) enable
rail [ index ] . getKelvinSensingEnable <= (unsigned char) enable
```

Some rails have kelvin sensing capabilities. See the module datasheet for more information about using kelvin sensing in your application.

### Setting Kelvin Sensing mode

Setting this value to 1 will enable Kelvin sensing on this rail.

### Getting Kelvin Sensing mode

Getting this value will return whether kelvin sensing is enabled on the rail. 1 is enabled 0 is disabled.

## Kelvin Sensing State (Get)

```
rail [ index ] . getKelvinSensingState <= (unsigned char) state
```

When a rail is capable of Kelvin sensing, under certain error conditions kelvin sensing may be disabled by the system. This command returns the current kelvin sensing state of the rail, either enabled or disabled.

## Operational Mode (Get/Set)

```
rail [ index ] . setOperationalMode => (unsigned char) mode
rail [ index ] . getOperationalMode <= (unsigned char) mode
```

Certain modules have multiple power regulation stages that can affect the behavior of the supplied rail voltage and current. This command sets and gets the preferred mode of operation for the given rail. Check the module datasheet for details on the capabilities and behavior of these operational modes.

## Operational State (Get)

```
rail [ index ] . getOperationalState <= (unsigned char) mode
```

When a rail is capable of multiple operational modes, getting this value will return the current operational state of the rail, this can indicate error conditions, or a certain operational mode if the rail is in an automatic behavior.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get commands fill the variable with the returned value.

stem.rail[0].getCurrent(microamps);
stem.rail[0].setCurrentLimit(limit);
stem.rail[0].getCurrentLimit(limit);
stem.rail[0].getTemperature(microcelsius);
stem.rail[0].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
```

### 3.1.13 RCservo Entity

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

The RCservo entity provides a pulsed signal based on the RC servo standard. This consist of a period lasting 20ms with a high pulse between 1-2ms. The time high corresponds to a specific position determined by the servo being used. For example if you are using a 90 degree servo a 1.5ms pulse will correspond to the 45 degrees. 1ms and 2ms pulses will correspond to 0 and 90 degree positions respectively.

The RCservo entity is an overload to the [Digital Entity](#) and therefor requires proper configuration of the Digital entity before the RCservo entity can be enabled.

---

**Note:** Not all BrainStem modules will have this capability.

---

#### Set/Get Enable

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

This functions gets/sets the RCservo function for a given pin (pending, it has been properly configured in the digital entity). At a firmware level this enables/disables the timers.

#### Set/Get Position

```
servo [ index ] . getPosition <= (unsigned char) position
servo [ index ] . setPosition => (unsigned char) position
```

This functions gets/sets the RCservo position. For outputs this will return the currently set position; however, for inputs it will return the value seen at the pin pending the pulse is valid. If the pulse or period are invalid a zero will be returned along with the error code aErrRange.

The default range is: 64 (1ms) - 192 (2ms). For example when working with a 90 degree servo setting the position to 64 will give you 0 degrees and 192 will give you 90 degrees.

---

**Note:** getPosition() will return the original setPosition() regardless of the reverse settings.

---

## Set/Get Reverse

```
servo [ index ] . getReverse <= (unsigned char) reverse  
servo [ index ] . setReverse => (unsigned char) reverse
```

This functions gets/sets the reverse (invert) option in the RCServo Class.

Given a setPosition of 64 the servo pulse will be 1ms; however, if you reverse it the value will now be treated as 192.

## Aligning the Digital and RCServo Entities

Digital Entity	Servo Entity	Pin Number	Assignment
digital[0]	servo[0]	Pin 0	RCServo Input
digital[1]	servo[1]	Pin 1	RCServo Input
digital[2]	servo[2]	Pin 2	RCServo Input
digital[3]	servo[3]	Pin 3	RCServo Input
digital[4]	servo[4]	Pin 4	RCServo Output
digital[5]	servo[5]	Pin 5	RCServo Output
digital[6]	servo[6]	Pin 6	RCServo Output
digital[7]	servo[7]	Pin 7	RCServo Output

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success.  
  
//Output  
//Set digital pin 8 as an RCServo output.  
err = stem.digital[8].setConfiguration(digitalConfigurationRCServoOutput);  
//Enable the servo channel  
err = stem.servo[4].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)
```

### 3.1.14 Relay Entity

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

The Relay entity is a simple class which allows the enabling and disabling of a specified relay.

#### Channel Enable (Get/Set)

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

Enables the relay channel for the specified index

#### Get Voltage (Get)

```
relay [ index ] . getVoltage <= (unsigned char) voltage
```

Returns the voltage of the specified index.

#### Code Examples

##### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get calls will fill the variable with the returned value.

err = stem.relay[0].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);

```

### 3.1.15 Signal Entity

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

SignalClass. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

Signal entity to Digital entity mapping varies from device to device. Please refer to the datasheet.

#### Timing

##### Set/Get enable

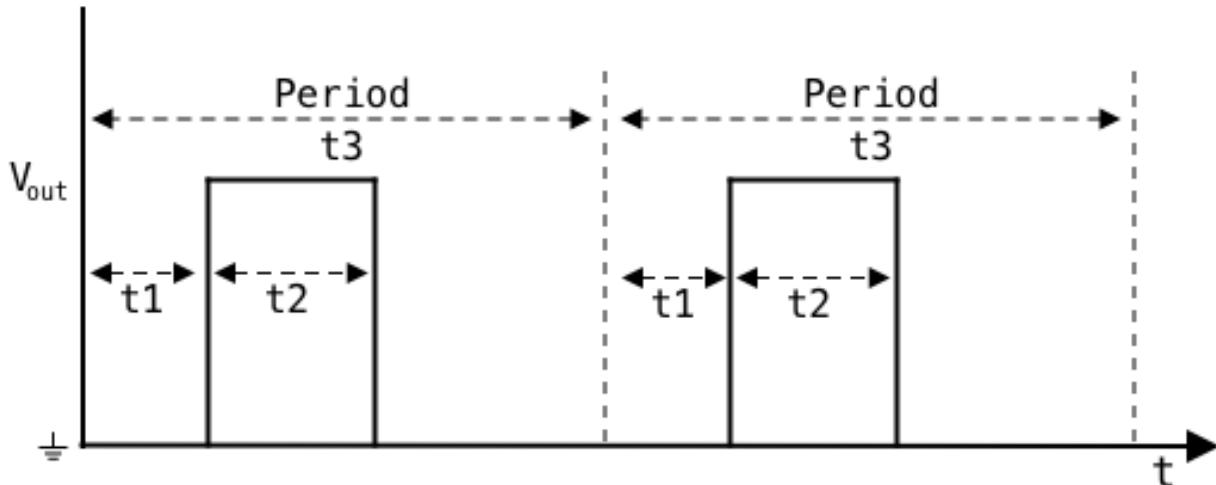
```

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

```

Enables the Signal Entity for a given index.

## Signal Entity Timing Diagram



### Set/Get T3 Time

```
signal [ index ] . getT3Time <= (unsigned int) t3_nsec
signal [ index ] . setT3Time => (unsigned int) t3_nsec
```

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

### Set/Get T2 Time

```
signal [ index ] . getT2Time <= (unsigned int) t2_nsec
signal [ index ] . setT2Time => (unsigned int) t2_nsec
```

The T2 time defines the high period of the waveform in nano seconds.

### Set/Get invert

```
signal [ index ] . getInvert <= (unsigned char) invert
signal [ index ] . setInvert => (unsigned char) invert
```

Inverts the meaning of the T2 time. When inverted the T2 time will represent the time in nano seconds that the waveform is low.

## Code Examples

### C++

```
//Setup 10Hz Signal Output with 50% Duty Cycle
err = stem.digital[0].setConfiguration(digitalConfigurationSignalOutput);
err = stem.signal[0].setT2Time(50000000);
err = stem.signal[0].setT3Time(100000000);
err = stem.signal[0].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;
```

### 3.1.16 Store Entity

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

Every BrainStem module has one or more stores. Stores are the BrainStem equivalent of a filesystem. Stores are broken up into a number of slots, each of which can be thought of as a file. A Store generally represents a specific type of storage. Flash or internal, RAM, or SD if the BrainStem includes an SD slot. The most common usage of slots and stores is for the storage of reflex code that will run on the BrainStem module. Additionally Bulk capture of Analog data can write to a slot within a store. Slots within the internal store can be set up as boot slots by setting the appropriate slot number in the system configuration. See the :doc:`System <system>` entity for more information about setting a boot slot.

The number and type of stores is Model specific. Details about the number of slots per store, and available stores can be found in the data sheets for specific models.

There are a number of commands for manipulating stores, which are detailed below. Many of the store commands are only accessible from host API's and UI applications, however commands relating to enabling reflex files in slots are accessible from the reflex language.

### Get Slot State (Get)

```
store [ index ] . getSlotState <= (unsigned char) state
```

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

### Load Slot (Write)

```
store [ index ] . loadSlot => (slot, byte buffer, buffer length)
```

This command writes a data buffer into a slot for the given store. It is only available from host side API's.

### Unload Slot (Read)

```
store [ index ] . unloadSlot <= (slot, byte buffer, max buffer size, length read)
```

#### **This command reads the slot in the given store into the byte buffer. The length**

will never be more than the max buffer size given, but may be less if the slot contents were shorter than max buffer length.

### Slot Enable (Set)

```
store [ index ] . slotEnable => (unsigned char) slot
```

This command enables the reflex file in the given store and slot. This command is accessible from the reflex language.

### Slot Disable (Set)

```
store [ index ] . slotDisable => (unsigned char) slot
```

This command disables the reflex file in the given store and slot. This command is accessible from the reflex language.

### Slot Capacity (Get)

```
store [ index ] . slotCapacity (unsigned char) slot <= (unsigned short) capacity
```

This command gets the maximum capacity of the given slot for the store. This command is accessible from the reflex language.

## Slot Size (Get)

```
store [ index ] . slotSize (unsigned char) slot <= (unsigned short) size
```

This command gets the current size of the data in the given slot for the store. This can be the size in bytes of the reflex byte code file, or the data size for a bulk capture.

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success.

stem.store[0].getSlotState(3, state); // gets the state of slot 3 in the internal
    ↪store.
stem.store[0].loadSlot(3, buffer, length); // loads the data in buffer.
stem.store[1].unloadSlot(0, buffer, 300, length); // unloads at most 300 bytes from
    ↪the 1st RAM slot.
stem.store[0].enableSlot(1);
stem.store[0].disableSlot(1);
stem.store[0].getSlotCapacity(1, size); // gets the max size of the slot.
Stem.store[0].getSlotSize(1, size); // gets the current size of the data in the slot.
```

### Reflex

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

### Python

```
res = stem.store[0].getSlotState(3) #res.value is the state of slot 3 in the internal
    ↪store
stem.store[0].loadSlot(3, buffer, length) # loads length bytes from buffer to slot 3
res = stem.store[1].unloadSlot(0) # res.value is a tuple of (strgetBytes/int) of the
    ↪data in slot 0 and the length
stem.store[0].enableSlot(3)
stem.store[0].disableSlot(3)
res = stem.store[0].getCapacity(1) #res.value is the max size of the slot
res = stem.store[0].getSize(1) #res.value is the current size of the data in slot 1
```

### 3.1.17 System Entity

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

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

#### System save

```
system . save => (void)
```

BrainStem configuration settings are stored in volatile memory until the save command is executed. Settings such as the BootSlot, and changes to the Module or Router address will not persist across resets unless followed by a call to:

#### System reset (Set)

```
system . reset => (void)
```

Calling `system.reset()` will reset the BrainStem module just as if the reset button were pressed.

#### User LED

```
system . setLED => (unsigned char) state
system . getLED <= (unsigned char) state
```

Gets or Sets the state of the User LED. Setting LED with a value of 1 turns the User LED on and setting it to 0 turns it off.

#### Boot Slot (Get/Set)

```
system . setBootSlot => (unsigned char) slot
system . getBootSlot <= (unsigned char) slot
```

BrainStem modules can be configured to enable a reflex file at boot. The reflex file must be loaded into a slot in the internal store. Setting the boot slot to the value 255 will disable on boot functionality. For more information about stores and slots please see the store section of the reference manual. For more information about reflexes please see the Reflex section of the manual.

#### Input Voltage (Get)

```
system . getInputVoltage <= (unsigned int) inputVoltage
```

The input voltage system command is a read only command and will return the input supply voltage of the BrainStem module in micro volts.

### **Serial Number (Get)**

```
system . getSerialNumber <= (unsigned int) serialNumber
```

Read only command that returns the unique module serial number. The returned value is an unsigned int. In Acroname UI applications the serial number is generally represented as an 8 character Hexadecimal number.

### **BrainStem Model (Get)**

```
system . getModel <= (unsigned char) BrainStem model.
```

Read only command that returns the model of the BrainStem module.

### **Hardware Version (Get)**

```
system . getHardwareVersion <= (unsigned int) Hardware Version.
```

Read only command that returns the hardware version of the module. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

### **Version (Get)**

```
system . getVersion <= (unsigned int) version number.
```

Read only command that returns the version number of the BrainStem firmware. This is a packed format. The aVersion.h C API can represent this version in a human readable manner. The format of the version number is 3 digits separated by ..

```
major . minor . patch
```

### **Module Address (Get)**

```
system . getModule <= (unsigned char) module
```

The module address is the number used to address the module on the BrainStem network and from the host. This is a combination of the module base address, any software offset that is applied and any hardware module offset.

### **Module Base Address (Get)**

```
system . getModuleBaseAddress <= (unsigned char) module
```

The module base address is the default or base address of the module, before any offsets are applied.

### Module Software Offset (Set/Get)

```
system . getModuleSoftwareOffset <= (unsigned char) software offset
system . setModuleSoftwareOffset => (unsigned char) software offset
```

The module software offset is added to the module's base address and any hardware offsets to determine the final module address of the module. This setting is not applied until saved and the module has been reset.

### Module Hardware Offset (Get)

```
system . getModuleHardwareOffset <= (unsigned char) module hardware offset
```

MTM BrainStems have a set of module offset pins which will adjust the module address via hardware. See the data sheet for your MTM module for more information about these hardware settings. The module offset command is a read only command that returns the offset that will be added to the base module address and any software offset to determine the operating address of the MTM BrainStem module. Changes to the hardware offset are applied when the Device is reset.

### Router Address (Get/Set)

```
system . setRouter => (unsigned char) module
system . getRouter <= (unsigned char) module
```

The BrainStem router address refers to the BrainStem module address of the module that will coordinate communication with the host system. This setting is not applied until it is saved and the module has been reset.

Changing the router address can have negative consequences for communicating with the BrainStem network. Please see the appendix on the BrainStem Network setup for more information.

- [Appendix: Brainstem Universal Entity Interface](#)
- [Appendix: The BrainStem Communication Protocol](#)

### HeartBeat Interval (Get/Set)

```
system . setHBInterval => (unsigned char) interval
system . getHBInterval <= (unsigned char) interval
```

Gets or sets the heartbeat interval to control the amount of heartbeat traffic. This value is set at approximately 1/50th of a second resolution. Heartbeat packets are handled by the underlying system, and are indicated on the brainstem by the blinking green heartbeat LED. UI applications also have Heartbeat indicators. Default value is 12.

## Code Examples

### C++

```
// Get requests fill the parameter with the current system value upon success.  
// All commands return aErr values when errors are encountered and aErrNone on  
// success.  
  
stem.system.save();  
stem.system.reset();  
stem.system.setLED(1);  
stem.system.getLED(state);  
stem.system.setBootSlot(5);  
stem.system.getBootSlot(slot);  
stem.system.getInputVoltage(voltage);  
stem.system.getModule(address);  
stem.system.getRouter(address);  
stem.system.setRouter(6);  
stem.system.getModuleBaseAddress(address);  
stem.system.setModuleSoftwareOffset(16);  
stem.system.getModuleSoftwareOffset(offset);  
stem.system.getModuleHardwareOffset(offset);  
stem.system.getSerialNumber(serialNumber);  
stem.system.getModel(model);  
stem.system.getHardwareVersion(hardwareVersion);  
stem.system.getVersion(version);  
stem.system.getHBInterval(interval);  
stem.system.setHBInterval(interval);
```

### Python

```
stem.system.save()  
stem.system.reset()  
stem.system.setLED(1)  
state = stem.system.getLED()  
print state.value  
stem.system.setBootSlot(5)  
slot = stem.system.getBootSlot()  
print slot.value  
inputVoltage = stem.system.getInputVoltage()  
print inputVoltage.value  
module = stem.system.getModule()  
print module.value  
address = stem.system.getModuleBaseAddress();  
print address.value  
stem.system.setModuleSoftwareOffset(16);  
offset = stem.system.getModuleSoftwareOffset();  
print offset.value  
offset = stem.system.getModuleHardwareOffset();  
print offset.value  
serialNumber = stem.system.getSerialNumber()  
print serialNumber.value  
model = stem.system.getModel()  
hardwareVersion = stem.system.getHardwareVersion()  
print hardwareVersion.value  
version = stem.system.getVersion()  
print brainstem.version.get_version_string(version.value)
```

(continues on next page)

(continued from previous page)

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

## Reflex

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

stem.system.save();
stem.system.reset();
stem.system.setLED(1);
stem.system.getLED(state);
stem.system.setBootSlot(5);
stem.system.getBootSlot(slot);
stem.system.getInputVoltage(voltage);
stem.system.getModule(address);
stem.system.getRouter(address);
stem.system.setRouter(6);
stem.system.getModuleBaseAddress(address);
stem.system.setModuleSoftwareOffset(16);
stem.system.getModuleSoftwareOffset(offset);
stem.system.getModuleHardwareOffset(offset);
stem.system.getSerialNumber(serialNumber);
stem.system.getModel(model);
stem.system.getHardwareVersion(hardwareVersion);
stem.system.getVersion(version);
stem.system.getHBInterval(interval);
stem.system.setHBInterval(interval);
```

### 3.1.18 Temperature Entity

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

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

#### Temperature (Get)

```
temperature [ index ] . getTemperature => (int) microcelsius
```

Returns a temperature measurement in microcelsius.

#### Code Examples

##### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get commands fill the variable with the returned value.

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

## Reflex

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

## Python

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

### 3.1.19 Timer Entity

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

The Timer entity provides simple scheduling for events in the reflex system. BrainStem modules generally contain between 4 and 8 timers depending on the module. The most common usage is to write a timer reflex and load and enable it on the BrainStem module, then an expiration can be set for the timer, and this reflex code will be executed when the timer expires.

Timers have two modes, single which executes just once and repeat which executes until the expiration is set to zero or the mode is changed to single.

#### Expiration (Get/Set)

```
timer [ index ] . getExpiration <= (unsigned int) microseconds  
timer [ index ] . setExpiration => (unsigned int) microseconds
```

Gets or sets the next expiration for this timer in microseconds. If zero, the timer is not currently set to expire in the future.

#### Mode (Get/Set)

```
timer [ index ] . getMode <= (unsigned char) mode  
timer [ index ] . setMode => (unsigned char) mode
```

Gets or sets the current timer mode. 1 for repeat mode and 0 for single mode.

When in repeat mode an expiration will occur every n microseconds when n is the expiration setting of the timer. To stop a repeat timer, set its expiration to 0.

When in single mode (The default) setting a non-zero expiration will cause the timer to trigger a single time after the expiration setting in microseconds. If a timer is set, resetting its expiration to zero will clear the timer, and no reflex code will be triggered.

#### Reflex example

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

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

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on
// success. Get commands fill the variable with the returned value.

stem.timer[0].getExpiration(uSecs);
stem.timer[0].setExpiration(1000000); // Sets the timer for 1 second in the future.
stem.timer[0].getMode(mode);
stem.timer[0].setMode(timerModeRepeat) // timerModeRepeat is a convenience define.
```

### Reflex

```
// Get commands fill the variable with the returned value.

stem.timer[0].getExpiration(uSecs);
stem.timer[0].setExpiration(1000000); // Sets the timer for 1 second in the future.
stem.timer[0].getMode(mode);
stem.timer[0].setMode(timerModeRepeat) // timerModeRepeat is a convenience define.
```

### Python

```
uSecs = stem.timer[3].getExpiration()
stem.timer[3].setExpiration(1000000) # Sets the timer for 1 second in the future.
mode = stem.timer[3].getMode()
stem.timer[3].setMode(timerModeRepeat) // timerModeRepeat is a convenience define.
```

## 3.1.20 UART Entity

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

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

### Channel Enable (Get/Set)

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

Enables the uart channel for the specified index.

### Change Baudrate (Get/Set)

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

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

### Change Protocol (Get/Set)

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

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

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success. Get calls will fill the variable with the returned value.  
  
err = stem.uart[0].setEnable(1);  
err = stem.uart[1].setEnable(1);  
  
err = stem.uart[0].getEnable(&enable);  
err = stem.uart[1].getEnable(&enable);  
  
err = stem.uart[0].setEnable(0);  
err = stem.uart[1].setEnable(0);
```

### Python

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

### 3.1.21 USB Entity

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

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

#### Port Enable/Disable (Set)

```
usb . setPortEnable => (unsigned char) channel
usb . setPortDisable => (unsigned char) channel
```

Enables or Disables the given downstream channel. This call enables or disables data and power together for the given channel.

#### Data Enable/Disable (Set)

```
usb . setDataEnable => (unsigned char) channel
usb . setDataDisable => (unsigned char) channel
```

Enables or Disables data only for given downstream channel. This call enables or disables the usb data (+) and data (-) lines for the given channel.

Calls to this command have no side effects on the power connections for the channel. If power was enabled before the call then it will still be enabled after the call to setDataEnable/Disable.

#### High Speed Data Enable/Disable (Set)

```
usb . setHiSpeedDataEnable => (unsigned char) channel
usb . setHiSpeedDataDisable => (unsigned char) channel
```

Enables or Disables Hi Speed data only for given downstream channel. This call enables or disables the usb data (+) and data (-) lines for the given channel.

Calls to this command have no side effects on the power connections for the channel. If power was enabled before the call then it will still be enabled after the call to setSuperSpeedDataEnable/Disable.

#### Super Speed Data Enable/Disable (Set)

```
usb . setSuperSpeedDataEnable => (unsigned char) channel
usb . setSuperSpeedDataDisable => (unsigned char) channel
```

Enables or Disables Super Speed (3.0) data only for given downstream channel. This call enables or disables the usb data (+) and data (-) lines for the given channel.

Calls to this command have no side effects on the power connections for the channel. If power was enabled before the call then it will still be enabled after the call to setSuperSpeedDataEnable/Disable.

### Power Enable/Disable (Set)

```
usb . setPowerEnable => (unsigned char) channel  
usb . setPowerDisable => (unsigned char) channel
```

Enables or Disables power only for given downstream channel. This call enables or disables the usb power connection for the given channel.

Calls to this command have no side effects on the data connections for the channel. If data was enabled before the call then it will still be enabled after the call to setPowerEnable/Disable.

### port Voltage/Current (Get)

```
usb . getPortVoltage (unsigned char) channel <= (unsigned int) microvolts  
usb . getPortCurrent (unsigned char) channel <= (unsigned int) microamps
```

Returns the last read values for Voltage (in microvolts) and Current (in microamps) for the given channel.

### Hub Mode (Get/Set)

```
usb . getHubMode <= (unsigned int) state  
usb . setHubMode => (unsigned int) state
```

Gets/Sets the hubs mode in the form of a big mapped representation. See the product datasheet for state mapping. Usually represents the downstream ports power and data lines enable/disable state.

### Hub State (Get)

---

**Note:** This function has been removed in version 2.5. This functionality is moved to [Port State](#).

---

### Hub Error Status (Get)

---

**Note:** This function has been removed in version 2.5. This functionality is moved to [Port Error](#).

---

### Clear Port Error Status (Set)

```
usb . clearPortErrorStatus => (unsigned char) channel
```

Clears the error status for the given channel

### Upstream Mode (Get/Set)

```
usb . getUpstreamMode <= (unsigned char) mode
usb . setUpstreamMode => (unsigned char) mode
```

Gets/Sets the mode of the upstream USB ports. Options are Auto, 0 or 1

### Upstream State (Get)

```
usb . getUpstreamState <= (unsigned char) state
```

Gets the upstream switch state for the USB upstream ports. Returns none if no ports are plugged in, port 0 if the mode is set correctly and a cable is plugged into port 0, and port 1 if the mode is set correctly and a cable is plugged into port 1

### Enumeration Delay (Get/Set)

```
usb . getEnumerationDelay <= (unsigned int) ms_delay
usb . setEnumerationDelay => (unsigned int) ms_delay
```

Gets/Sets the inter-port enumeration delay in milliseconds. The enumeration delay sequentially enables data and power to downstream ports after the defined delay time. After setting and saving this parameter all downstream ports will be initially disabled upon system power-on or reset. Similarly, if there is no upstream connection, all downstream ports will be disabled. When an upstream connection is applied, or after the system boots, the system will wait for the defined delay time and enable the lowest port number. The system will then wait for the defined delay time and then enable the next highest port. This behavior repeats until all ports are enabled.

Inconsistent behavior from race conditions may occur if enumeration delay is used in conjunction with Reflex programs which also manipulate the downstream port states. Care should be taken to ensure no conflicts between the enumeration delay and Reflex programs.

---

**Note:** This setting should be saved with a `stem.system.save()` call.

---

### Upstream Boost Mode (Get/Set)

```
usb . getUpstreamBoostMode <= (unsigned char) setting
usb . setUpstreamBoostMode => (unsigned char) setting
```

Gets/Sets the upstream boost mode. Boost mode increase the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Modes: 0 = no boost, 1 = 4% boost, 2 = 8% boost, 3 = 12% boost.

---

**Note:** This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

---

### Down Stream Boost Mode (Get/Set)

```
usb . getDownstreamBoostMode <= (unsigned char) setting  
usb . setDownstreamBoostMode => (unsigned char) setting
```

Gets/Sets the Downstream boost mode. Boost mode increase the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Modes: 0 = no boost, 1 = 4% boost, 2 = 8% boost, 3 = 12% boost.

---

**Note:** This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

---

### Port Current Limit (Get/Set)

```
usb . setPortCurrentLimit => (unsigned char) channel, (unsigned int) microamps  
usb . getPortCurrentLimit (unsigned char) channel <= (unsigned int) microamps
```

Gets/Sets the current limit for the downstream channel. There are a number of settings for current limits ranging from 100 mAmps to 2.5 amps. See the USB hub datasheet for specific settings information.

### Port Mode setting (Get/Set)

```
usb . setPortMode => (unsigned char) channel, (unsigned char) mode  
usb . getPortMode (unsigned char) channel <= (unsigned char) mode
```

Gets/Sets the Port mode for the channel specified. The portmode is a bitmapped setting. Device specific mode options are listed in the data-sheet. There is a unified listing of all port mode bits at `usbPortMode` within [USB Entity](#).

### Port State (Get)

```
usb . getPortState (unsigned char) channel <= (unsigned char) mode
```

Gets the Port state for the channel specified. State options for the device are listed in the device data-sheet.

### Port Error (Get)

```
usb . getPortError (unsigned char) channel <= (unsigned char) mode
```

Gets the Port error status for the channel specified. Error status for the device are listed in the device data-sheet.

## System Temperature (Get)

---

**Note:** This function has been removed in version 2.5. This functionality is moved to [temperature](#).

---

## Connect Mode setting (Get/Set)

```
usb . setConnectMode => (unsigned char) channel, (unsigned char) mode
usb . getConnectMode (unsigned char) channel <= (unsigned char) mode
```

Gets/Sets the connect mode for the channel specified. Check the device datasheet for more information regarding the use of this function.

## CC1/CC2 Enable/Disable setting (Get/Set)

```
usb . setCC[1|2]Enable => (unsigned char) channel, (unsigned char) bEnable
usb . getCC[1|2]Enable (unsigned char) channel <= (unsigned char) bEnable
```

Gets or sets the enabled status of the CC1/CC2 lines.

## CC1/CC2 Current (Get)

```
usb . getCC[1|2]Current (unsigned char) channel <= (unsigned char) microAmps
```

Gets the current on the CC1/CC2 line in microAmps.

## CC1/CC2 Voltage (Get)

```
usb . getCC[1|2]Voltage (unsigned char) channel <= (unsigned char) microVolts
```

Gets the voltage on the CC1/CC2 lines in microVolts.

## SBU Enable/Disable setting (Get/Set)

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

Gets or sets the enabled status of the SBU lines.

## Cable Flip (Get/Set)

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

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

## Code Examples

### C++

```
// All commands return aErr values when errors are encountered and aErrNone on  
// success. Get commands fill the variable with the returned value.  
  
stem.usb.setPortEnable(1);  
stem.usb.setPortDisable(2);  
stem.usb.setDataEnable(0);  
...
```

### Reflex

```
// Get commands fill the variable with the returned value.  
  
stem.usb.setPortEnable(1);  
stem.usb.setPortDisable(2);  
stem.usb.setDataEnable(0);  
...
```

### Python

```
stem.usb.setPortEnable(1)  
stem.usb.setPortDisable(2)  
stem.usb.setDataEnable(0)  
stem.usb.setDataDisable(1)  
stem.usb.setPowerEnable(0)  
stem.usb.setPowerDisable(0)  
microamps = stem.usb.getPortCurrent(0)  
print microamps.value  
microvolts = stem.usb.getPortVoltage(0)  
print microvolts.value  
stem.usb.setPortCurrentLimit(0, limit_setting)  
state = stem.usb.getHubState()  
print state.value  
...
```

### 3.1.22 USB System Entity

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

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

#### Upstream Connection (Get/Set)

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

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

#### Enumeration Delay (Get/Set)

```
usbsystem . getEnumerationDelay <= (unsigned int) ms_delay
usbsystem . setEnumerationDelay => (unsigned int) ms_delay
```

Gets/Sets the inter-port enumeration delay in milliseconds. The enumeration delay sequentially enables data and power to downstream ports after the defined delay time. After setting and saving this parameter all downstream ports will be initially disabled upon system power-on or reset. Similarly, if there is no upstream connection, all downstream ports will be disabled. When an upstream connection is applied, or after the system boots, the system will wait for the defined delay time and enable the lowest port number. The system will then wait for the defined delay time and then enable the next highest port. This behavior repeats until all ports are enabled.

Inconsistent behavior from race conditions may occur if enumeration delay is used in conjunction with Reflex programs which also manipulate the downstream port states. Care should be taken to ensure no conflicts between the enumeration delay and Reflex programs.

#### Enabled List (Get/Set)

```
usbsystem . getEnabledList <= (unsigned int) list
usbsystem . setEnabledList => (unsigned int) list
```

The enabled list function provides state and control over all lower ports enables. It is equivalent to calling calling get/set enabled from the *PortClass* on all ports at once. The returned variable is in a bit mapped format. Please see the product data sheet for specific bit meanings.

#### Mode List (Get/Set)

```
usbsystem . getModeList <= (unsigned int [NUM_PORTS]) list
usbsystem . setModeList => (unsigned int [NUM_PORTS]) list
```

The mode list function gives you access and control to all lower level port modes. It is equivalent to calling get/set mode from the *PortClass* on all ports at once.

## State List (Get)

```
usbsystem . getModeList <= (unsigned int [NUM_PORTS]) list  
usbsystem . setModeList => (unsigned int [NUM_PORTS]) list
```

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

## Power Behavior (Get/Set)

```
usbsystem . getPowerBehavior <= (unsigned char) behavior  
usbsystem . setPowerBehavior => (unsigned char) behavior
```

The power behavior controls how power will be allocated to each lower level port. This behavior comes into play when the requested power of the system exceeds the available power. i.e. first come first serve, even distribution, priority list. See the product datasheet for specific implementations.

## Power Behavior Config (Get/Set)

```
usbsystem . getPowerBehaviorConfig <= (unsigned int []) config  
usbsystem . setPowerBehaviorConfig => (unsigned int []) config
```

Some power behaviors require a list of parameters in order to operate. For instance in priority list mode the user can supply a list of port indexes to priorities for power. This feature is product specific and users should consult the manual for further details.

## Data Role Behavior (Get/Set)

```
usbsystem . getDataRoleBehavior <= (unsigned char) behavior  
usbsystem . setDataRoleBehavior => (unsigned char) behavior
```

Some Type-C ports are capable of being dual role ports (DRP). Meaning they are capable of being either a host or a device. The behavior defined here will determine if that is allowed, what happens if it is, and what occurs when a host goes away. Examples are: first come first serve, priority list, static/fixed selection, etc. See the product datasheet for specific implementations.

## Data Role Behavior Config (Get/Set)

```
usbsystem . getDataRoleBehaviorConfig <= (unsigned int []) config  
usbsystem . setDataRoleBehaviorConfig => (unsigned int []) config
```

Many of the data role behaviors require a list of parameters in order to operate. For instance in a static/fixed mode the config would indicate what port is the upstream connection.

### Selector Mode (Get/Set)

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

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

## 3.2 Python API Reference

Welcome to the BrainStem Python API reference documentation. This documentation covers the Python Acroname BrainStem module. This reference assumes that you understand the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation.

- *BrainStem Overview*
- *BrainStem Terminology*
- *Getting Started with the BrainStem*.

Next check out the python *Getting Started* section.

### 3.2.1 Getting (Quickly) Started

The BrainStem python package allows you to interact with a collection of BrainStem modules from python. The API is similar to both the C++ and Reflex API's, with a few significant differences. The remainder of this section details the structure and functionality of the python API.

Most modern operating systems come with all the tools needed to immediately install the BrainStem python libraries and create python based applications. As such, simply download the [latest development package](#)<sup>6</sup>, 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>

<sup>6</sup> <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](mailto:support@acroname.com)

### 3.2.2 Acroname Modules

#### Quick Access:

- [USBHub3p](#)
- [USBHub2x4](#)
- [USBCSwitch](#)
- [MTMDAQ2](#)
- [MTMEtherStem](#)
- [MTMIOSerial](#)
- [MTMLOAD1](#)
- [MTMPM1](#)
- [MTMRelay](#)
- [MTMUSBStem](#)
- [MTMDAQ1](#)
- [EtherStem](#)
- [USBStem](#)

Each type of BrainStem module is represented by a corresponding concrete Module implementation. The following classes are instantiated to allow communication through to the corresponding BrainStem module hardware.

The instantiation and subsequent connection to each module is as follows

```
>>> stem = USBStem()
# 0xFFFFFFFF is the serial number of the module.
>>> stem.connect(0xFFFFFFFF)
```

Connecting to the BrainStem module can take multiple forms, the simplest way to connect when you know the module's serial number is to call connect with the serial number, as in the above code snippet. If you don't know the serial number of the module, you can perform a discovery of the modules currently connected and print that information. For details of the connection functions API please see [Connections](#) in this reference.

## USBHub3p

```
class brainstem.stem.USBHub3p(address=6, enable_auto_networking=True, model=19)
Concrete Module implementation for the USBHub3p.

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

#### Entities:

- system
- app[0-3]

- pointers[0-3]
- usb
- store[0-1]
- temperature
- timer[0-7]

**Useful Constants:**

- BASE\_ADDRESS (6)
- NUMBER\_OF\_STORES (2)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_DOWNSTREAM\_USB (8)
- NUMBER\_OF\_UPSTREAM\_USB (2)

Bit defines for port state UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if  
(state & brainstem.BIT(aUSBHUB3P\_USB\_VBUS\_ENABLED))

- aUSBHUB3P\_USB\_VBUS\_ENABLED (0)
- aUSBHUB3P\_USB2\_DATA\_ENABLED (1)
- aUSBHUB3P\_USB3\_DATA\_ENABLED (3)
- aUSBHUB3P\_USB\_SPEED\_USB2 (11)
- aUSBHUB3P\_USB\_SPEED\_USB3 (12)
- aUSBHUB3P\_USB\_ERROR\_FLAG (19)
- aUSBHUB3P\_USB2\_BOOST\_ENABLED (20)
- aUSBHUB3P\_DEVICE\_ATTACHED (23)

Bit defines for port error UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if  
(error & brainstem.BIT(aUSBHUB3P\_ERROR\_VBUS\_OVERCURRENT))

- aUSBHUB3P\_ERROR\_VBUS\_OVERCURRENT (0)
- aUSBHUB3P\_ERROR\_VBUS\_BACKDRIVE (1)
- aUSBHUB3P\_ERROR\_HUB\_POWER (2)
- aUSBHUB3P\_ERROR\_OVER\_TEMPERATURE (3)

**connect (serial\_number, \*\*kwargs)**

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

**Parameters**

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

**Returns**

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

**Return type**  
Result.error

[Back to the top](#)

## USBHub2x4

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

Concrete Module implementation for the USBHub2x4.

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

### Entities:

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

### Useful Constants:

- BASE\_ADDRESS (6)
- NUMBER\_OF\_STORES (3)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_DOWNSTREAM\_USB (4)
- NUMBER\_OF\_UPSTREAM\_USB (2)

Bit defines for port error UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if  
(error & brainstem.BIT(aUSBHUB2X4\_USB\_VBUS\_ENABLED))

- aUSBHUB2X4\_USB\_VBUS\_ENABLED (0)
- aUSBHUB2X4\_USB2\_DATA\_ENABLED (1)
- aUSBHUB2X4\_USB\_ERROR\_FLAG (19)
- aUSBHUB2X4\_USB2\_BOOST\_ENABLED (20)
- aUSBHUB2X4\_DEVICE\_ATTACHED (23)
- aUSBHUB2X4\_CONSTANT\_CURRENT (24)

Bit defines for port error UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if  
(error & brainstem.BIT(aUSBHUB3P\_ERROR\_VBUS\_OVERCURRENT))

- aUSBHUB2X4\_ERROR\_VBUS\_OVERCURRENT (0)
- aUSBHUB2X4\_ERROR\_OVER\_TEMPERATURE (3)
- aUSBHub2X4\_ERROR\_DISCHARGE (4)

**connect** (*serial\_number*, *\*\*kwargs*)

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

*Back to the top*

## USBCSwitch

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

Concrete Module implementation for the USBC-Switch.

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

**Entities:**

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

**Useful Constants:**

- BASE\_ADDRESS (6)
- NUMBER\_OF\_STORES (3)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_USB (1)

- NUMBER\_OF\_MUXS (1)
- NUMBER\_OF\_EQUALIZERS (2)

Bit defines for port state UInt32 use brainstem.BIT(X) from aDefs.h to get bit value. i.e if  
(state & brainstem.BIT(usbPortStateVBUS))

- usbPortStateVBUS (0)
- usbPortStateHiSpeed (1)
- usbPortStateSBU (2)
- usbPortStateSS1 (3)
- usbPortStateSS2 (4)
- usbPortStateCC1 (5)
- usbPortStateCC2 (6)
- usbPortStateCCFlip (13)
- usbPortStateSSFlip (14)
- usbPortStateSBUFlip (15)
- usbPortStateErrorFlag (19)
- usbPortStateUSB2Boost (20)
- usbPortStateUSB3Boost (21)
- usbPortStateConnectionEstablished (22)
- usbPortStateCC1Inject (26)
- usbPortStateCC2Inject (27)
- usbPortStateCC1Detect (28)
- usbPortStateCC2Detect (29)
- usbPortStateCC1LogicState (30)
- usbPortStateCC2LogicState (31)
- usbPortStateOff (0)
- usbPortStateSideA (1)
- usbPortStateSideB (2)
- usbPortStateSideUndefined (3)
- TRANSMITTER\_2P0\_40mV (0)
- TRANSMITTER\_2P0\_60mV (1)
- TRANSMITTER\_2P0\_80mV (2)
- TRANSMITTER\_2P0\_0mV (3)
- MUX\_1db\_COM\_0db\_900mV (0)
- MUX\_0db\_COM\_1db\_900mV (1)
- MUX\_1db\_COM\_1db\_900mV (2)
- MUX\_0db\_COM\_0db\_900mV (3)

- MUX\_0db\_COM\_0db\_1100mV (4)
- MUX\_1db\_COM\_0db\_1100mV (5)
- MUX\_0db\_COM\_1db\_1100mV (6)
- MUX\_2db\_COM\_2db\_1100mV (7)
- MUX\_0db\_COM\_0db\_1300mV (8)
- LEVEL\_1\_2P0 (0)
- LEVEL\_2\_2P0 (1)
- LEVEL\_1\_3P0 (0)
- LEVEL\_2\_3P0 (1)
- LEVEL\_3\_3P0 (2)
- LEVEL\_4\_3P0 (3)
- LEVEL\_5\_3P0 (4)
- LEVEL\_6\_3P0 (5)
- LEVEL\_7\_3P0 (6)
- LEVEL\_8\_3P0 (7)
- LEVEL\_9\_3P0 (8)
- LEVEL\_10\_3P0 (9)
- LEVEL\_11\_3P0 (10)
- LEVEL\_12\_3P0 (11)
- LEVEL\_13\_3P0 (12)
- LEVEL\_14\_3P0 (13)
- LEVEL\_15\_3P0 (14)
- LEVEL\_16\_3P0 (15)
- EQUALIZER\_CHANNEL\_BOTH (0)
- EQUALIZER\_CHANNEL\_MUX (1)
- EQUALIZER\_CHANNEL\_COMMON (2)
- NO\_DAUGHTERCARD (0)
- PASSIVE\_DAUGHTERCARD (1)
- REDRIVER\_DAUGHTERCARD (2)
- UNKNOWN\_DAUGHTERCARD (3)

**connect** (*serial\_number*, *\*\*kwargs*)

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

---

[Back to the top](#)

## MTMDAQ2

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

Concrete Module implementation for MTM-DAQ-2 module

**MTM-DAQ-2 modules contain contain the following entities:**

- system
- app[0-3]
- digital[0-1]
- analog[0-19]
- i2c[0]
- pointer[0-3]
- store[0-1]
- timer[0-7]

**Useful Constants:**

- BASE\_ADDRESS (14)
- NUMBER\_OF\_STORES (2)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_DIGITALS (2)
- NUMBER\_OF\_ANALOGS (20)
- NUMBER\_OF\_I2C (1)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- ANALOG\_RANGE\_P0V064N0V064 (0)
- ANALOG\_RANGE\_P0V64N0V64 (1)
- ANALOG\_RANGE\_P0V128N0V128 (2)
- ANALOG\_RANGE\_P1V28N1V28 (3)
- ANALOG\_RANGE\_P1V28N0V0 (4)
- ANALOG\_RANGE\_P0V256N0V256 (5)
- ANALOG\_RANGE\_P2V56N2V56 (6)
- ANALOG\_RANGE\_P2V56N0V0 (7)
- ANALOG\_RANGE\_P0V512N0V512 (8)
- ANALOG\_RANGE\_P5V12N5V12 (9)
- ANALOG\_RANGE\_P5V12N0V0 (10)

- ANALOG\_RANGE\_P1V024N1V024 (11)
- ANALOG\_RANGE\_P10V24N10V24 (12)
- ANALOG\_RANGE\_P10V24N0V0 (13)
- ANALOG\_RANGE\_P2V048N0V0 (14)
- ANALOG\_RANGE\_P4V096N0V0 (15)
- ANALOG\_BULK\_CAPTURE\_MAX\_HZ (500000)
- ANALOG\_BULK\_CAPTURE\_MIN\_HZ (1)

**connect** (*serial\_number*, *\*\*kwargs*)

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

[Back to the top](#)

## MTMEtherStem

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

Concrete Module implementation for MTM EtherStem modules

**USBStem modules contain the following entities:**

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]
- servo[0-7]
- store[0-2]
- timer[0-7]

**Useful Constants:**

- BASE\_ADDRESS (4)
- NUMBER\_OF\_STORES (3)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_SD\_SLOTS (255)

- NUMBER\_OF\_ANALOGS (4)
- DAC\_ANALOG\_INDEX (3)
- FIXED\_DAC\_ANALOG (False)
- NUMBER\_OF\_DIGITALS (15)
- NUMBER\_OF\_I2C (2)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_SERVOS (8)
- NUMBER\_OF\_SERVO\_OUTPUTS (4)
- NUMBER\_OF\_SERVO\_INPUTS (4)
- ANALOG\_BULK\_CAPTURE\_MAX\_HZ (200000)
- ANALOG\_BULK\_CAPTURE\_MIN\_HZ (7000)

**connect (serial\_number, \*\*kwargs)**

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

[Back to the top](#)

## MTMIOSerial

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

Concrete Module implementation for MTM-IO-Serial module

**MTM-IO-SERIAL modules contain contain the following entities:**

- system
- app[0-3]
- digital[0-8]
- i2c[0]
- pointer[0-3]
- servo[0-7]
- signal[0-4]
- store[0-1]
- temperature
- timer[0-7]

- uart[0-3]
- rail[0-2]

**Useful Constants:**

- BASE\_ADDRESS (8)
- NUMBER\_OF\_STORES (2)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_DIGITALS (8)
- NUMBER\_OF\_I2C (1)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_UART (1)
- NUMBER\_OF\_RAILS (3)
- NUMBER\_OF\_SERVOS (8)
- NUMBER\_OF\_SERVO\_OUTPUTS (4)
- NUMBER\_OF\_SERVO\_INPUTS (4)
- NUMBER\_OF\_SIGNALS (5)
- 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)

**connect** (*serial\_number*, *\*\*kwargs*)

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

[Back to the top](#)

**MTMLOAD1**

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

Concrete Module implementation for MTM-LOAD-1 module

**MTM-LOAD-1 modules contain contain the following entities:**

- system
- app[0-3]
- digital[0-3]
- i2c[0]
- pointer[0-3]
- store[0-1]
- timer[0-7]
- rail[0]
- temperature

**Useful Constants:**

- BASE\_ADDRESS (14)
- NUMBER\_OF\_STORES (2)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_DIGITALS (2)
- NUMBER\_OF\_I2C (1)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_RAILS (2)
- NUMBER\_OF\_TEMPERATURES (1)

`connect(serial_number, **kwargs)`

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

**Parameters**

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

**Returns**

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

**Return type**

`Result.error`

[Back to the top](#)

## MTMPM1

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

Concrete Module implementation for MTM-PM-1 module

**MTM-PM-1 modules contain contain the following entities:**

- system
- app[0-3]
- digital[0-1]
- i2c[0]
- pointer[0-3]
- store[0-1]
- timer[0-7]
- rail[0-1]
- temperature

**Useful Constants:**

- BASE\_ADDRESS (6)
- NUMBER\_OF\_STORES (2)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_DIGITALS (2)
- NUMBER\_OF\_I2C (1)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_RAILS (2)
- NUMBER\_OF\_TEMPERATURES (1)

**connect (serial\_number, \*\*kwargs)**

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

[Back to the top](#)

## MTMRelay

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

Concrete Module implementation for MTM-RELAY module

### MTM-RELAY modules contain contain the following entities:

- system
- app[0-3]
- digital[0-3]
- i2c[0]
- pointer[0-3]
- store[0-1]
- timer[0-7]
- relay[0-3]
- temperature

### Useful Constants:

- BASE\_ADDRESS (12)
- NUMBER\_OF\_STORES (2)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_DIGITALS (4)
- NUMBER\_OF\_I2C (1)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_RELAYS (4)

`connect(serial_number, **kwargs)`

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

#### Parameters

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

#### Returns

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

#### Return type

`Result.error`

[Back to the top](#)

**MTMUSBStem**

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

Concrete Module implementation for MTM USBStem modules

**MTMUSBStem modules contain the following entities:**

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]
- servo[0-7]
- signal[0-4]
- store[0-2]
- timer[0-7]

**Useful Constants:**

- BASE\_ADDRESS (4)
- NUMBER\_OF\_STORES (3)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_SD\_SLOTS (255)
- NUMBER\_OF\_ANALOGS (4)
- DAC\_ANALOG\_INDEX (3)
- FIXED\_DAC\_ANALOG (True)
- NUMBER\_OF\_DIGITALS (15)
- NUMBER\_OF\_I2C (2)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_SERVOS (8)
- NUMBER\_OF\_SERVO\_OUTPUTS (4)
- NUMBER\_OF\_SERVO\_INPUTS (4)
- NUMBER\_OF\_SIGNALS (5)
- ANALOG\_BULK\_CAPTURE\_MAX\_HZ (200000)
- ANALOG\_BULK\_CAPTURE\_MIN\_HZ (7000)

---

`connect (serial_number, **kwargs)`

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

[Back to the top](#)

## MTMDAQ1

```
class brainstem.stem.MTMDAQ1 (address=10, enable_auto_networking=True, model=20)
Concrete Module implementation for MTM-DAQ-1 module
```

**MTM-DAQ-1 modules contain contain the following entities:**

- system
- app[0-3]
- digital[0-1]
- analog[0-19]
- i2c[0]
- pointer[0-3]
- store[0-1]
- timer[0-7]

**Useful Constants:**

- BASE\_ADDRESS (10)
- NUMBER\_OF\_STORES (2)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_DIGITALS (2)
- NUMBER\_OF\_ANALOGS (20)
- NUMBER\_OF\_I2C (1)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- ANALOG\_RANGE\_P0V064N0V064 (0)
- ANALOG\_RANGE\_P0V64N0V64 (1)
- ANALOG\_RANGE\_P0V128N0V128 (2)
- ANALOG\_RANGE\_P1V28N1V28 (3)
- ANALOG\_RANGE\_P1V28N0V0 (4)

- ANALOG\_RANGE\_P0V256N0V256 (5)
- ANALOG\_RANGE\_P2V56N2V56 (6)
- ANALOG\_RANGE\_P2V56N0V0 (7)
- ANALOG\_RANGE\_P0V512N0V512 (8)
- ANALOG\_RANGE\_P5V12N5V12 (9)
- ANALOG\_RANGE\_P5V12N0V0 (10)
- ANALOG\_RANGE\_P1V024N1V024 (11)
- ANALOG\_RANGE\_P10V24N10V24 (12)
- ANALOG\_RANGE\_P10V24N0V0 (13)
- ANALOG\_RANGE\_P2V048N0V0 (14)
- ANALOG\_RANGE\_P4V096N0V0 (15)

**connect** (*serial\_number*, \*\**kwargs*)

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

[Back to the top](#)

## EtherStem

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

Concrete Module implementation for 40Pin EtherStem modules

### EtherStem modules contain the following entities:

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]
- servo[0-7]
- store[0-2]
- timer[0-7]

### Useful Constants:

- BASE\_ADDRESS (2)

- NUMBER\_OF\_STORES (3)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_SD\_SLOTS (255)
- NUMBER\_OF\_ANALOGS (4)
- DAC\_ANALOG\_INDEX (3)
- FIXED\_DAC\_ANALOG (False)
- NUMBER\_OF\_DIGITALS (15)
- NUMBER\_OF\_I2C (2)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_SERVOS (8)
- NUMBER\_OF\_SERVO\_OUTPUTS (4)
- NUMBER\_OF\_SERVO\_INPUTS (4)
- ANALOG\_BULK\_CAPTURE\_MAX\_HZ (200000)
- ANALOG\_BULK\_CAPTURE\_MIN\_HZ (7000)

**connect (serial\_number, \*\*kwargs)**

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

*Back to the top*

## USBStem

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

Concrete Module implementation for 40Pin USBStem modules

**USBStem modules contain contain the following entities:**

- system
- analog[0-3]
- app[0-3]
- clock
- digital[0-14]
- i2c[0-1]
- pointer[0-3]

- servo[0-7]
- store[0-2]
- timer[0-7]

**Useful Constants:**

- BASE\_ADDRESS (2)
- NUMBER\_OF\_STORES (3)
- NUMBER\_OF\_INTERNAL\_SLOTS (12)
- NUMBER\_OF\_RAM\_SLOTS (1)
- NUMBER\_OF\_SD\_SLOTS (255)
- NUMBER\_OF\_ANALOGS (4)
- DAC\_ANALOG\_INDEX (3)
- FIXED\_DAC\_ANALOG (False)
- NUMBER\_OF\_DIGITALS (15)
- NUMBER\_OF\_I2C (2)
- NUMBER\_OF\_POINTERS (4)
- NUMBER\_OF\_TIMERS (8)
- NUMBER\_OF\_APPS (4)
- NUMBER\_OF\_SERVOS (8)
- NUMBER\_OF\_SERVO\_OUTPUTS (4)
- NUMBER\_OF\_SERVO\_INPUTS (4)
- ANALOG\_BULK\_CAPTURE\_MAX\_HZ (200000)
- ANALOG\_BULK\_CAPTURE\_MIN\_HZ (7000)

**connect** (*serial\_number*, *\*\*kwargs*)

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

[Back to the top](#)

### 3.2.3 Package Structure

The BrainStem package consists of a number of modules, which together form the BrainStem python API.

#### **brainstem.module**

A module that provides base classes for BrainStem Modules and Entities.

The Module and Entity classes are designed to be extended for specific types of BraiStem Modules and Entities. For more information about Brainstem Modules and Entities, please see the [Terminology](#)<sup>7</sup> section of the [Acroname BrainStem Reference](#)<sup>8</sup>

#### **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](#)<sup>9</sup> section of the [Acroname BrainStem Reference](#)<sup>10</sup>

#### **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](#)<sup>11</sup>

#### **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](#)<sup>12</sup>

<sup>7</sup> <https://acroname.com/reference/terms.html>

<sup>8</sup> <https://acroname.com/reference>

<sup>9</sup> <https://acroname.com/reference/entities>

<sup>10</sup> <https://acroname.com/reference>

<sup>11</sup> <https://acroname.com/reference>

<sup>12</sup> <https://acroname.com/reference>

### **brainstem.defs**

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

### **brainstem.result**

A module that provides a result class for returning results of UEI commands.

Results consist of an error attribute and a value attribute. If the error attribute is set to NO\_ERROR, then the result value is the response to the UEI command that was sent.

For more information about return values for commands and UEI's see the [Acroname BrainStem Reference](#)<sup>13</sup>

### **brainstem.version**

Provides version access utilities.

## **3.2.4 Analog**

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

The AnalogClass is the interface to analog entities on BrainStem modules.

Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

#### **Useful constants:**

- CONFIGURATION\_INPUT (0)
- CONFIGURATION\_OUTPUT (1)
- HERTZ\_MINIMUM (7,000)
- HERTZ\_MAXIMUM (200,000)
- BULK\_CAPTURE\_IDLE (0)
- BULK\_CAPTURE\_PENDING (1)
- BULK\_CAPTURE\_FINISHED (2)
- BULK\_CAPTURE\_ERROR (3)

```
getBulkCaptureNumberOfSamples ()
```

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

#### **Returns**

**Result object, containing NO\_ERROR and sample number**  
or a non zero Error code.

#### **Return type**

*Result*

---

<sup>13</sup> <https://acroname.com/reference>

```
getBulkCaptureSampleRate ()  
Get the current sample rate setting for this analog when bulk capturing.  
Sample rate is in samples per second (Hertz).  
Returns  
Result object, containing NO_ERROR and sample rate  
or a non zero Error code.  
Return type  
Result  
getBulkCaptureState ()  
Get the current bulk capture state for this analog.  
Possible states of the bulk capture operation are; idle = 0 pending = 1 finished = 2 error = 3  
Returns  
Result object, containing NO_ERROR and bulk capture state  
or a non zero Error code.  
Return type  
Result  
getConfiguration ()  
Get the analog configuration.  
If the configuraton is 1 the analog is configured as an output, if the configuration is 0, the analog is set as an input.  
Returns  
Result object, containing NO_ERROR and analog configuration  
or a non zero Error code.  
Return type  
Result  
getEnable ()  
Get the enable state an analog output  
Get a boolean value corresponding to on/off  
Returns  
Result object, containing NO_ERROR and enable state  
or a non zero Error code.  
Return type  
Result  
getRange ()  
Get the range setting of an analog input  
Get a value corresponding to a discrete range option.  
Returns  
Result object, containing NO_ERROR and analog range  
or a non zero Error code.  
Return type  
Result  
getValue ()  
Get the raw ADC value in bits.  
Get a 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.
```

---

**Note:** Not all modules provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit ADC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

---

**Returns**

**Result object, containing NO\_ERROR and analog value**  
or a non zero Error code.

**Return type**

*Result*

**getVoltage ()**

Get the scaled micro volt value with reference to ground.

Get a 32 bit signed integer (in microVolts) based on the boards ground and reference voltages.

---

**Note:** Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

---

**Returns**

**Result object, containing NO\_ERROR and microVolts value**  
or a non zero Error code.

**Return type**

*Result*

**initiateBulkCapture ()**

Initiate a BulkCapture on this analog. Captured measurements are stored in the module's RAM store (RAM\_STORE) slot 0. Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure. When the bulk capture is complete getBulkCaptureState() will return either finished or error.

**Return type**

*Result.error*

**setBulkCaptureNumberOfSamples (value)**

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

Minimum # of Samples: 0 Maximum # of Samples: (BRAINSTEM\_RAM\_SLOT\_SIZE / 2) = (3FFF / 2) = 1FFF = 8191

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

**setBulkCaptureSampleRate (value)**

Set the sample rate for this analog when bulk capturing.

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

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

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**`setConfiguration (value)`**

Set the analog configuration.

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

**Param:**

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

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**`setEnable (enable)`**

Set the enable state of an analog output.

Set a boolean value corresponding to on/off

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**`setRange (value)`**

Set the range of an analog input.

Set a value corresponding to a discrete range option.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**`setValue (value)`**

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

Set a 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

---

**Note:** Not all modules 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**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setVoltage (value)**

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

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

---

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

---

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

## 3.2.5 App

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

The AppClass calls defined app reflexes on brainstem modules.

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

**execute (param)**

Execute an App reflex on a module.

**Param:**

param (int): App routine parameter.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**executeAndWaitForResult (param, msTimeout)**

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

**Param:**

param (int): App routine parameter.

**Param:**

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

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

### 3.2.6 Clock

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

The ClockClass is the interface to the realtime clock.

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

**getDay()**

Get the current day of the month

**Returns**

**Result object, containing NO\_ERROR and current day or  
a non zero Error code.**

**Return type**

*Result*

**getHour()**

Get the current hour

**Returns**

**Result object, containing NO\_ERROR and current hour or  
a non zero Error code.**

**Return type**

*Result*

**getMinute()**

Get the current minute

**Returns**

**Result object, containing NO\_ERROR and current minute or  
a non zero Error code.**

**Return type**

*Result*

**getMonth()**

Get the current month

**Returns**

**Result object, containing NO\_ERROR and current month or  
a non zero Error code.**

**Return type**

*Result*

**getSecond()**

Get the current second

**Returns**

**Result object, containing NO\_ERROR and current second or  
a non zero Error code.**

**Return type**

*Result*

**getYear()**

Get the current year

**Returns**

**Result object, containing NO\_ERROR and current year or  
a non zero Error code.**

**Return type**

*Result*

**setDay** (*day*)

Set the current day of the month

**Param:**

value (int): Current 2 digit day.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setHour** (*hour*)

Set the current hour

**Param:**

value (int): Current 2 digit hour.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setMinute** (*minute*)

Set the current minute

**Param:**

value (int): Current 2 digit minute.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setMonth** (*month*)

Set the current month

**Param:**

value (int): Current 2 digit month.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setSecond** (*second*)

Set the current second

**Param:**

value (int): Current 2 digit second.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setYear** (*year*)

Set the current year

**Param:**

value (int): Current 4 digit year.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

### 3.2.7 Definitions

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

```
brainstem.defs.MODEL_ETHERSTEM = 5
EtherStem Model number

brainstem.defs.MODEL_MTM_DAQ_1 = 20
MTM-DAQ-1 Model number

brainstem.defs.MODEL_MTM_DAQ_2 = 22
MTM-DAQ-2 Model number

brainstem.defs.MODEL_MTM_ETHERSTEM = 15
MTM EtherStem Model number

brainstem.defs.MODEL_MTM_IOSERIAL = 13
MTM-IO-Serial Model number

brainstem.defs.MODEL_MTM_PM_1 = 14
MTM-PM-1 Model number

brainstem.defs.MODEL_MTM_RELAY = 18
MTM-Relay Model number

brainstem.defs.MODEL_MTM_USBSTEM = 16
MTM USBStem Model number

brainstem.defs.MODEL_USBHUB_2X4 = 17
USBHub 2x4 Model number

brainstem.defs.MODEL_USBHUB_3C = 24
USBHub3c Model number

brainstem.defs.MODEL_USBHUB_3P = 19
USBHub 3+ Model number

brainstem.defs.MODEL_USBSTEM = 4
USBStem Model number

brainstem.defs.MODEL_USB_C_SWITCH = 21
USBC-Switch Model number

brainstem.defs.model_info(model)
Get Model information.
```

**Parameters**

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

**Returns**

Return a string containing model information.

**Return type**

str

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

Get Model Name.

**Parameters**

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

**Returns**

Return a string containing model name.

**Return type**

str

## 3.2.8 Digital

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

The DigitalClass is the interface to digital entities on BrainStem modules.

Digital entities have the following 5 possibilities: Digital Input, Digital Output, RC Servo Input, RC Servo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

**Useful Constants:**

- VALUE\_LOW (0)
- VALUE\_HIGH (1)
- CONFIGURATION\_INPUT (0)
- CONFIGURATION\_OUTPUT (1)
- CONFIGURATION\_RCSERVO\_INPUT (2)
- CONFIGURATION\_RCSERVO\_OUTPUT (3)
- CONFIGURATION\_HIGHZ (4)
- CONFIGURATION\_INPUT\_PULL\_UP (0)
- CONFIGURATION\_INPUT\_NO\_PULL (4)
- CONFIGURATION\_INPUT\_PULL\_DOWN (5)
- CONFIGURATION\_SIGNAL\_OUTPUT (6)
- CONFIGURATION\_SIGNAL\_INPUT (7)

```
getConfiguration()
```

Get the digital configuration.

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

**Returns**

Result object, containing NO\_ERROR and digital configuration or a non zero Error code.

**Return type**

*Result*

**getState()**

Get the digital state.

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

**Returns**

Result object, containing NO\_ERROR and digital state or a non zero Error code.

**Return type**

*Result*

**getStateAll()**

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

**Returns**

Result object, containing NO\_ERROR and the digital state of all digits where bit 0 = digital 0 and bit 1 = digital 1 etc. 0 = logic low and 1 = logic high. A non zero Error code is returned on error.

**Return type**

*Result*

**setConfiguration(*configuration*)**

Set the digital configuration.

**Param:****configuration (int):**

- Digital Input: CONFIGURATION\_INPUT = 0
- Digital Output: CONFIGURATION\_OUTPUT = 1
- RCservo Input: CONFIGURATION\_RCSERVO\_INPUT = 2
- RCservo Output: CONFIGURATION\_RCSERVO\_OUTPUT = 3
- High Z State: CONFIGURATION\_HIGHZ = 4
- Digital Input with pull up: CONFIGURATION\_INPUT\_PULL\_UP = 0 (Default)
- Digital Input with no pull up or pull down: CONFIGURATION\_INPUT\_NO\_PULL = 4
- Digital Input with pull down: CONFIGURATION\_INPUT\_PULL\_DOWN = 5
- Digital Signal Output: CONFIGURATION\_SIGNAL\_OUTPUT = 6
- Digital Signal Input: CONFIGURATION\_SIGNAL\_INPUT = 7

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

**setState(*state*)**

Set the digital state.

**Param:****state (int):**

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

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

**setStateAll (state)**

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

**Param:****state (uint):**

The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc. Configuration must be set to output.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

## 3.2.9 Discovery

A module that provides methods for discovering brainstem modules over USB and TPCIP.

The discovery module provides an interface for locating BrainStem modules accross multiple transports. It provides a way to find all modules for a give transport as well as specific modules by serial number, or first found. The result of a call to one of the discovery functions is either a list of brainstem.link.Spec objects, or a single brainstem.link.Spec.

The Discovery module allows users to find specific brainstem devices via their serial number, or a list of all devices connected to the host via usb or on the same subnet via TCP/IP. In all cases a *Spec* object is returned with connection details for the device. In addition do connection details, the BrainStem model is returned. This model is one of a list of BrainStem device model numbers which are accessible via the *defs* module.

A typical interactive python session finding all connected USB modules might look like the following.

```
>> import brainstem >> module_list = brainstem.discover.findAllModules(brainstem.link.Spec.USB)
>> print [str(s) for s in module_list] ['Model: 4 LinkType: USB(serial: 0xCB4A3B25, module: 0)',
   'Model: 13 LinkType: USB(serial: 0x40F5849A, module: 0)']
```

For an overview of links, discovery and the Brainstem network see the [Acroname BrainStem Reference](#)<sup>14</sup>

```
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)

<sup>14</sup> <https://acroname.com/reference>

- Super Speed Plus (5)
- `product_name` (string): USB string descriptor.
- `manufacture` (string): USB string descriptor.
- `serial_number` (string): USB string descriptor.

`brainstem.discover.findAllModules(transports)`

Return a list of Specs for all modules found on the transports given.

Transports can be presented as a list, and the results would be a list of all modules found for those transports. TCPIP modules take a little longer to find due to the Multicast and gather necessary for finding modules on the local network segment.

#### Parameters

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

#### Returns

A list of the Specs for all modules found.

#### Return type

`list(Spec)`

`brainstem.discover.findFirstModule(transport)`

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

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

#### args:

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

#### return:

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

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

Return the Spec for the module with the given serial number.

Transports can be presented as a list. TCPIP modules take a little longer to find due to the Multicast and gather necessary for finding modules on the local network segment.

#### Parameters

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

#### Returns

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

#### Return type

`Spec`

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

**Gets downstream device USB information for all Acroname hubs.**

#### args:

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

**Return:****Result: Result object, containing NO\_ERROR and a tuple of DeviceNode's**

containing the detected downstream devices.  
- aErrParam: Passed in values are not valid.  
(NULL, size etc).  
- aErrMemory: No more room in the list  
- aErrNotFound: No Acroname devices were found.

### 3.2.10 Entities

**Quick Access:**

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

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

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

#### System

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

Acccess system controls configuration and information.

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

**Useful Constants:**

- BOOT\_SLOT\_DISABLE (255)

*Back to the top*

## Analog

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

The AnalogClass is the interface to analog entities on BrainStem modules.

Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

### Useful constants:

- CONFIGURATION\_INPUT (0)
- CONFIGURATION\_OUTPUT (1)
- HERTZ\_MINIMUM (7,000)
- HERTZ\_MAXIMUM (200,000)
- BULK\_CAPTURE\_IDLE (0)
- BULK\_CAPTURE\_PENDING (1)
- BULK\_CAPTURE\_FINISHED (2)
- BULK\_CAPTURE\_ERROR (3)

[Back to the top](#)

## Digital

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

The DigitalClass is the interface to digital entities on BrainStem modules.

Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

### Useful Constants:

- VALUE\_LOW (0)
- VALUE\_HIGH (1)
- CONFIGURATION\_INPUT (0)
- CONFIGURATION\_OUTPUT (1)
- CONFIGURATION\_RCSERVO\_INPUT (2)
- CONFIGURATION\_RCSERVO\_OUTPUT (3)
- CONFIGURATION\_HIGHZ (4)
- CONFIGURATION\_INPUT\_PULL\_UP (0)
- CONFIGURATION\_INPUT\_NO\_PULL (4)
- CONFIGURATION\_INPUT\_PULL\_DOWN (5)
- CONFIGURATION\_SIGNAL\_OUTPUT (6)
- CONFIGURATION\_SIGNAL\_INPUT (7)

[Back to the top](#)

## Digital

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

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

[Back to the top](#)

## I2C

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

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

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

### Useful Constants:

- I2C\_DEFAULT\_SPEED (0)
- I2C\_SPEED\_100Khz (1)
- I2C\_SPEED\_400Khz (2)
- I2C\_SPEED\_1000Khz (3)

[Back to the top](#)

## Mux

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

Access MUX specialized entities on certain BrainStem modules.

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexor) can direct that input to on or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

### Useful Constants:

- UPSTREAM\_STATE\_ONBOARD (0)
- UPSTREAM\_STATE\_EDGE (1)
- UPSTREAM\_MODE\_AUTO (0)
- UPSTREAM\_MODE\_ONBOARD (1)
- UPSTREAM\_MODE\_EDGE (2)
- DEFAULT\_MODE (UPSTREAM\_MODE\_AUTO)

[Back to the top](#)

## Pointer

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

Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via `setOffset`. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

### Useful Constants:

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

*Back to the top*

## Rail

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

Provides power rail functionality on certain modules.

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

### Useful Constants:

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

- OPERATIONAL\_STATE\_OVER\_CURRENT\_FAULT (18)
- OPERATIONAL\_STATE\_OVER\_POWER\_FAULT (19)
- OPERATIONAL\_STATE\_REVERSE\_POLARITY\_FAULT (20)
- OPERATIONAL\_STATE\_OVER\_TEMPERATURE\_FAULT (21)
- OPERATIONAL\_STATE\_OPERATING\_MODE (24)
- OPERATIONAL\_STATE\_CONSTANT\_CURRENT (0)
- OPERATIONAL\_STATE\_CONSTANT\_VOLTAGE (1)
- OPERATIONAL\_STATE\_CONSTANT\_POWER (2)
- OPERATIONAL\_STATE\_CONSTANT\_RESISTANCE (3)

[Back to the top](#)

## Digital

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

The Signal Class is the interface to digital pins configured to produce square wave signals.

This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

[Back to the top](#)

## Store

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

Access the store on a BrainStem Module.

The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure.

Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

### Useful Constants:

- INTERNAL\_STORE (0)
- RAM\_STORE (1)
- SD\_STORE (2)
- EEPROM\_STORE (3)

[Back to the top](#)

## Temperature

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

Provide interface to temperature sensor.

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

[Back to the top](#)

## Timer

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

Schedules events to occur at future times.

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

### Useful Constants:

- SINGLE\_SHOT\_MODE (0)
- REPEAT\_MODE (1)
- DEFAULT\_MODE (SINGLE\_SHOT\_MODE)

[Back to the top](#)

## USB

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

USBClass provides methods to interact with a USB hub and USB switches.

Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

### Useful Constants:

- UPSTREAM\_MODE\_AUTO (2)
- UPSTREAM\_MODE\_PORT\_0 (0)
- UPSTREAM\_MODE\_PORT\_1 (1)
- UPSTREAM\_MODE\_NONE (255)
- DEFAULT\_UPSTREAM\_MODE (UPSTREAM\_MODE\_AUTO)
- UPSTREAM\_STATE\_PORT\_0 (0)
- UPSTREAM\_STATE\_PORT\_1 (1)
- BOOST\_0\_PERCENT (0)
- BOOST\_4\_PERCENT (1)
- BOOST\_8\_PERCENT (2)
- BOOST\_12\_PERCENT (3)
- PORT\_MODE\_SDP (0)
- PORT\_MODE\_CDP (1)

- PORT\_MODE\_CHARGING (2)
- PORT\_MODE\_PASSIVE (3)
- PORT\_MODE\_USB2\_A\_ENABLE (4)
- PORT\_MODE\_USB2\_B\_ENABLE (5)
- PORT\_MODE\_VBUS\_ENABLE (6)
- PORT\_MODE\_SUPER\_SPEED\_1\_ENABLE (7)
- PORT\_MODE\_SUPER\_SPEED\_2\_ENABLE (8)
- PORT\_MODE\_USB2\_BOOST\_ENABLE (9)
- PORT\_MODE\_USB3\_BOOST\_ENABLE (10)
- PORT\_MODE\_AUTO\_CONNECTION\_ENABLE (11)
- PORT\_MODE\_CC1\_ENABLE (12)
- PORT\_MODE\_CC2\_ENABLE (13)
- PORT\_MODE\_SBU\_ENABLE (14)
- PORT\_MODE\_CC\_FLIP\_ENABLE (15)
- PORT\_MODE\_SS\_FLIP\_ENABLE (16)
- PORT\_MODE\_SBU\_FLIP\_ENABLE (17)
- PORT\_MODE\_USB2\_FLIP\_ENABLE (18)
- PORT\_MODE\_CC1\_INJECT\_ENABLE (19)
- PORT\_MODE\_CC2\_INJECT\_ENABLE (20)
- PORT\_SPEED\_NA (0)
- PORT\_SPEED\_HISPEED (1)
- PORT\_SPEED\_SUPERSPEED (2)

[Back to the top](#)

## RCServo

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

Provides RCServo functionality on certain modules.

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

### Useful Constants:

- SERVO\_DEFAULT\_POSITION (128)
- SERVO\_DEFAULT\_MIN (64)
- SERVO\_DEFAULT\_MAX (192)

[Back to the top](#)

**USB**

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

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

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

**Useful Constants:**

- VALUE\_LOW (0)
- VALUE\_HIGH (1)

*Back to the top*

**3.2.11 Equalizer**

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

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

**getReceiverConfig(*channel*)**

Gets the receiver configuration for a given channel.

**Parameters**

*channel* – The equalizer receiver channel.

**Returns**

Result object, containing NO\_ERROR and the receiver configuration of the supplied channel or a non zero Error code.

**getTransmitterConfig()**

Gets the transmitter configuration

**Returns**

Result object, containing NO\_ERROR and the current transmitter config or a non zero Error code.

**setReceiverConfig(*channel, config*)**

Sets the receiver configuration for a given channel.

**Parameters**

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

**Returns**

Result.error Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**setTransmitterConfig(*config*)**

Sets the transmitter configuration

**Parameters**

*config* – Configuration to be applied to the transmitter.

**Returns**

Result.error Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

### 3.2.12 I2C

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

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

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

#### Useful Constants:

- I2C\_DEFAULT\_SPEED (0)
- I2C\_SPEED\_100Khz (1)
- I2C\_SPEED\_400Khz (2)
- I2C\_SPEED\_1000Khz (3)

**getSpeed()**

Get the current speed setting of the I2C object.

#### Returns

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

**read(address, length)**

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

#### Param:

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

#### Param:

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

#### Returns

Result object, containing NO\_ERROR and read data or a non zero Error code.

#### Return type

*Result*

**setPullup(bEnable)**

Set software controlled I2C pullup state.

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

#### Param:

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

#### Returns

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

#### Return type

*Result.error*

**setSpeed(value)**

Set the current speed setting of the I2C object.

#### Param:

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

#### Returns

returns NO\_ERROR on success or PARAMETER\_ERROR on failure.

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

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

**Param:**

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

**Param:**

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

**Param:**

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

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

### 3.2.13 Link

A module that provides a Spec class for specifying a connection to a BrainStem module.

A Spec instance fully describes a connection to a brainstem module. In the case of USB based stems this is simply the serial number of the module. For TCPIP based stems this is an IP address and TCP port.

For more information about links and the Brainstem network see the [Acroname BrainStem Reference](#)<sup>15</sup>

**class** `brainstem.link.Spec(transport, serial_number, module, model, **keywords)`

Spec class for specifying connection details

Instances of Spec represent the connection details for a brainstem link. The Spec class also contains constants representing the possible transport types for BrainStem modules.

**Parameters**

- **transport** (*int*) – One of USB or TCPIP.
- **serial\_number** (*int*) – The module serial number.
- **module** – The module address on the Brainstem network.
- **model** – The device model number of the Brainstem module.
- **\*\*keywords** – For TCPIP modules the possibilities are,
  - *ip\_address*: (*int*) The IP address of the module.
  - *tcp\_port*: (*int*) The TCP port of the module.

**transport**

instance attribute holding transport type.

**Type**

USB/TCPIP

**serial\_number**

Module serial number.

**Type**

int

<sup>15</sup> <https://acroname.com/reference>

```
module
    Module address.

    Type
        int

model
    Brainstem device type: See Defs module for listing.

    Type
        int

ip_address
    IP address for TCP/IP based modules.

    Type
        Optional[int]

tcp_port
    TCP port for TCP/IP based modules.

    Type
        Optional[int]

TCP_IP = 2
    TCPIP transport type.

USB = 1
    USB transport type.

class brainstem.link.Status
    Status variables represent the link status possibilities for Brainstem Links.

    Status States:
        • STOPPED (0)
        • INITIALIZING (1)
        • RUNNING (2)
        • STOPPING (3)
        • SYNCING (4)
        • INVALID_LINK_STREAM (5)
        • IO_ERROR (6)
        • UNKNOWN_ERROR (7)
```

### 3.2.14 Module

A module that provides base classes for BrainStem Modules and Entities.

The Module and Entity classes are designed to be extended for specific types of BraiStem Modules and Entities. For more information about Brainstem Modules and Entities, please see the [Terminology<sup>16</sup>](#) section of the [Acroname BrainStem Reference<sup>17</sup>](#)

---

<sup>16</sup> <https://acroname.com/reference/terms.html>

<sup>17</sup> <https://acroname.com/reference>

---

```
class brainstem.module.Entity(module, command, index)
```

Base class for BrainStem Entity.

Provides the default implementation for a functional entity within the BrainStem. This can include IO like GPIOs, Analogs etc. For a more detailed description of Entities see the [Terminology<sup>18</sup>](#) section of the brainstem reference for more information.

**call\_UEI** (*option*)

Result.error: Call a set UEI on this entity.

**Parameters**

**option** (*int*) – The command option.

**Returns**

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

**Return type**

Result.error

**property command**

Return the entitiy command.

**Type**

*int*

**drain\_UEI** (*option*)

drain UEI packets matchin option.

**Parameters**

**option** (*int*) – The command option.

**Returns**

**Returns a result object, whose value is the number of**  
packets drained, and the error value set to NO\_ERROR

**Return type**

*Result*

**get\_UEI** (*option*)

Result.error: Get a UEI value.

**Parameters**

**option** (*int*) – The command option.

**Returns**

**Returns a result object, whose value is set,**  
or with the requested value when the results error is set to NO\_ERROR

**Return type**

*Result*

**get\_UEIBytes32** (*option*)

Get a UEI Bytes 32 bit value.

**Parameters**

**option** (*int*) – The command option.

**Returns**

**Returns a result object, whose value is set,**  
or with the requested value when the results error is set to NO\_ERROR

**Return type***Result***get\_UEIBytes8 (option)**

Get a UEI Bytes 8 bit value.

**Parameters****option** (*int*) – The command option.**Returns****Returns a result object, whose value is set,**

or with the requested value when the results error is set to NO\_ERROR

**Return type***Result***get\_UEI\_with\_param (option, param)**

Result.error: Get a UEI value based on a parameter.

**Parameters**

- **option** (*int*) – The command option.
- **param** (*byte*) – The command parameter

**Returns****Returns a result object, whose value is set,**

or with the requested value when the results error is set to NO\_ERROR

**Return type***Result***property index**

Return the entity index

**Type***int***property module**

Return this entities module.

**Type***Module***set\_UEI16 (option, value)**

Result.error: Call a set UEI with short param on this entity.

**Parameters**

- **option** (*int*) – The command option.
- **value** (*short*) – The short parameter to send.

**Returns**

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

**Return type**

Result.error

**set\_UEI32 (option, value)**

Result.error: Call a set UEI with int param on this entity.

**Parameters**

- **option** (*int*) – The command option.
- **value** (*int*) – The int parameter to send.

**Returns**

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

**Return type**

Result.error

**set\_UEI32\_with\_subindex** (*option*, *subindex*, *value*)

Result.error: Call a set UEI with a subindex.

**Parameters**

- **option** (*int*) – The command option.
- **subindex** (*byte*) – The subindex of the entity.
- **param** (*byte*) – The byte parameter to send.

**Returns**

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

**Return type**

Result.error

**set\_UEI8** (*option*, *value*)

Result.error: Call a set UEI with byte param on this entity.

**Parameters**

- **option** (*int*) – The command option.
- **value** (*byte*) – The byte parameter to send.

**Returns**

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

**Return type**

Result.error

**set\_UEI8\_with\_subindex** (*option*, *subindex*, *value*)

Result.error: Call a set UEI with a subindex.

**Parameters**

- **option** (*int*) – The command option.
- **subindex** (*byte*) – The subindex of the entity.
- **param** (*byte*) – The byte parameter to send.

**Returns**

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

**Return type**

Result.error

**class** `brainstem.module.Module` (*address*, *enable\_auto\_networking=True*, *model=0*)

Base class for BrainStem Modules.

Provides default implementations for connecting and disconnecting from BrainStem modules via the module's serial number, a `Spec` object or through another module.

<sup>18</sup> <https://acroname.com/reference/terms.html>

**property address**

Return the Brainstem module address.

**Type**

int

**property bAutoNetworking**

Return the current networking mode.

**Type**

bool

**connect (*transport*, *serial\_number*)**

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

**Parameters**

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

**Returns**

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

**Return type**

Result.error

**connectFromSpec (*spec*)**

Result.error: Connect to a BrainStem module with a Spec.

**Parameters**

**spec** (*Spec*) – The specifier for the connection.

**Returns**

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

**Return type**

Result.error

**connectThroughLinkModule (*module*)**

Result.error: Connect to network module.

Connects to a Brainstem module on a BrainStem network, through the module given as an argument.  
The module passed in must have an active valid connection.

**Parameters**

**module** (*Module*) – The brainstem module to connec through.

**Returns**

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

**Return type**

Result.error

**disconnect ()**

Disconnect from the Brainstem module.

**discoverAndConnect (*transport*, *serial\_number=None*)**

Discover and connect from the Module level.

A discover-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that

serial number. Passing 0 or None as the serial number will create a link to the first link module found on the specified transport.

#### Parameters

- **transport** (*int*) – The module address to switch to for this module instance.
- **serial\_number** (*int*) – The module serial\_number to look for.

#### Returns

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

#### Return type

Result.error

#### **getStatus ()**

Returns the status of the BrainStem connection

See brainstem.link.Status for the possible states.

#### **isConnected ()**

Returns true if the Module has an active connection or false otherwise

#### **property link**

return the current link or None.

#### Type

Link

#### **property model**

returns the model number of the object.

#### Type

Model

#### **reconnect ()**

Reconnect a lost connection to a Brainstem module.

#### **setModuleAddress (address)**

Set the address of the module object.

This method changes the local address of the module, not of the device. It is possible to set the module address of the device via system.setModuleSoftwareOffset().

#### Parameters

- **address** (*int*) – The module address to switch to for this module instance.

#### Returns

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

#### Return type

Result.error

#### **setNetworkingMode (mode)**

Set the networking mode of the module object.

By default the module object is configured to automatically adjust its address based on the devices current module address. So that, if the device has a software or hardware offset it will still be able to communicate with the device. If advanced networking is required the auto networking mode can be turned off.

#### Parameters

- **mode** (*bool*) – True or 1 = Auto networking False or 0 = Manual networking

**Returns**

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

**Return type**

Result.error

**property spec**

Return the current spec object.

**Type**

*Spec*

### 3.2.15 Mux

**class** brainstem.entity.**Mux** (*module*, *index*)

Access MUX specialized entities on certain BrainStem modules.

A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexor) can direct that input to on or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

**Useful Constants:**

- UPSTREAM\_STATE\_ONBOARD (0)
- UPSTREAM\_STATE\_EDGE (1)
- UPSTREAM\_MODE\_AUTO (0)
- UPSTREAM\_MODE\_ONBOARD (1)
- UPSTREAM\_MODE\_EDGE (2)
- DEFAULT\_MODE (UPSTREAM\_MODE\_AUTO)

**getChannel()**

Gets the current selected channel.

**Param:**

*channel* (int): The channel of the mux to enable.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**getConfiguration()**

Gets the configuration of the Mux.

**Returns**

Return result object with NO\_ERROR set and the current mux voltage setting in the Result.value or an Error.

**Return type**

*Result*

**getEnable ()**

Gets the enable/disable status of the mux.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**getSplitMode ()**

Gets the bit packed mux split configuration.

**Returns**

Return result object with NO\_ERROR set and the current mux voltage setting in the Result.value or an Error.

**Return type**

*Result*

**getVoltage (channel)**

Gets the voltage of the specified channel.

On some modules this is a measured value so may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Return result object with NO\_ERROR set and the current mux voltage setting in the Result.value or an Error.

**Return type**

*Result*

**setChannel (channel)**

Enables the specified channel of the mux.

**Param:**

channel (int): The channel of the mux to enable.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setConfiguration (config)**

Sets the configuration of the mux.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setEnable (bEnable)**

Enables or disables the mux based on the param.

**Param:**

bEnable (bool): True = Enable, False = Disable

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setSplitMode** (*splitMode*)

Sets the mux split configuration

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

### 3.2.16 Pointer

**class brainstem.entity.Pointer** (*module, index*)

Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

**Useful Constants:**

- **POINTER\_MODE\_STATIC** (0)
- **POINTER\_MODE\_INCREMENT** (1)

**getChar ()**

Get the value of the pad at the current cursor position.

If the mode is increment this read will increment the cursor by 1 byte.

**Returns**

**Result object, containing NO\_ERROR and the value**  
or a non zero Error code.

**Return type**

*Result*

**getInt ()**

Get the value of the pad at the current cursor position.

If the mode is increment this read will increment the cursor by 4 bytes.

**Returns**

**Result object, containing NO\_ERROR and the value**  
or a non zero Error code.

**Return type**

*Result*

**getMode ()**

Get the pointer offset for this pointer.

**Returns**

**Result object, containing NO\_ERROR and the current mode**  
or a non zero Error code.

**Return type**

*Result*

**getOffset ()**

Get the pointer offset for this pointer.

**Returns**

**Result object, containing NO\_ERROR and the current offset**  
or a non zero Error code.

**Return type**

*Result*

**getShort ()**

Get the value of the pad at the current cursor position.

If the mode is increment this read will increment the cursor by 2 bytes.

**Returns**

**Result object, containing NO\_ERROR and the value**  
or a non zero Error code.

**Return type**

*Result*

**getTransferStore ()**

Get the open slot handle for this pointer.

**Returns**

**Result object, containing NO\_ERROR and the handle**  
or a non zero Error code.

**Return type**

*Result*

**setChar (*charVal*)**

Set a value at the current cursor position within the pad.

If the mode is increment this write will increment the cursor by 1 byte.

**Param:**

**charVal (char): The value to set into the pad at the current**  
pointer position.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error  
codes on failure.

**Return type**

*Result.error*

**setInt (*intVal*)**

Set a value at the current cursor position within the pad.

If the mode is increment this write will increment the cursor by 4 bytes.

**Param:**

**short (short): The value to set into the pad at the current**  
pointer position.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error  
codes on failure.

**Return type**

*Result.error*

**setMode (*mode*)**

Set the pointer offset for this pointer.

**Param:**

mode (char): The mode. One of POINTER\_MODE\_STATIC or  
POINTER\_MODE\_INCREMENT

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setOffset (offset)**

Set the pointer offset for this pointer.

**Param:**

offset (char): The byte offset within the pad (0 - 255).

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setShort (shortVal)**

Set a value at the current cursor position within the pad.

If the mode is increment this write will increment the cursor by 2 bytes.

**Param:**

**shortVal (short): The value to set into the pad at the current pointer position.**

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setTransferStore (handle)**

Set store slot handle for the pad to store and store to pad transfer.

**Param:**

handle (char): The handle. Open slot handle id.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**transferToStore (length)**

Transfer length bytes from the pad cursor position into the open store handle.

If the mode is increment the transfer will increment the cursor by length bytes.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**transerFromStore (length)**

Transfer length bytes from the open store handle to the cursor position in the pad.

If the mode is increment the transfer will increment the cursor by length bytes.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

### 3.2.17 Port

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

The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

**getAllocatedPower()**

Gets the currently allocated power. This value is determined by the power manager which is responsible for budgeting the systems allocated power envelope.

**Returns**

value (int): Allocated power in milli-watts (mW) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getAvailablePower()**

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

**Returns**

value (int): Available power in milli-watts (mW) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getCC1Enabled()**

Gets the current enable value of the CC1 lines. Sub-component (CC1) of getEnabled.

**Returns**

**value:**

- 1 = CC1 enabled
- 0 = CC1 disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getCC2Enabled()**

Gets the current enable value of the CC2 lines. Sub-component (CC2) of getEnabled.

**Returns**

**value:**

- 1 = CC2 enabled
- 0 = CC2 disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getCCEnabled()**

Gets the current enable value of the CC lines. Sub-component (CC) of getEnabled.

**Returns****value (int):**

- 1 = CC enabled
- 0 = CC disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getCurrentLimit()**

Gets the current limit of the port.

**Returns**

value: limit of the port in microAmps (uA) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getCurrentLimitMode()**

Gets the current limit mode. The mode determines how the port will react to an over current condition.

**Returns**

value: An enumerated representation of the active current limit mode. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getDataEnabled()**

Gets the current enable value of the data lines. Sub-component (Data) of getEnable.

**Returns****value:**

- 1 = Data enabled
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getDataHS1Enabled()**

Gets the current enable value of the High Speed 1 side (HS1) data lines.: Sub-component of getDataHSEnable.

**Returns****value:**

- 1 = Data enabled;
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getDataHS2Enabled()**

Gets the current enable value of the High Speed B side (HSB) data lines.: Sub-component of getDataHSEnable.

**Returns****value:**

- 1 = Data enabled
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getDataHSEnabled()**

Gets the current enable value of the High Speed (HS) data lines. Sub-component of `getDataEnable`.

**Returns**  
**value:**

- 1 = Data enabled
- 0 = Data disabled

error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getDataHSRoutingBehavior()**

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Returns**  
value: An enumerated representation of the routing behavior. error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getDataRole()**

Gets the Port Data Role.

**Returns**  
value: The current data role. See product datasheet for details. error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getDataSS1Enabled()**

Gets the current enable value of the Super Speed 1 side (SS1) data lines.: Sub-component of `getDataSSEnable`.

**Returns**  
**value:**

- 1 = Data enabled
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getDataSS2Enabled()**

Gets the current enable value of the Super Speed 2 side (SS2) data lines. Sub-component of `getDataSSEnable`.

**Returns**  
**value:**

- 1 = Data enabled;
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getDataSSEnable()**

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of

`getDataEnable.`

**Returns**

**value:**

- 1 = Data enabled;
- 0 = Data disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

`getDataSSRoutingBehavior()`

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Returns**

**value:** An enumerated representation of the routing behavior. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

`getDataSpeed()`

Gets the speed of the enumerated device.

**Returns**

**value:** Bit mapped value representing the devices speed. See product datasheet for details. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

`getEnabled()`

Gets the current enable value of the port.

**Returns**

**value:**

- 1 = Fully enabled port
- 0 = One or more disabled sub-components.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

`getErrors()`

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

**Returns**

**value:** Bit mapped value representing the errors of the port See product datasheet for details. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

`getHSBoost()`

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Returns**

**value:** An enumerated representation of the boost range. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

`getMode()`

Gets current mode of the port

**Returns**

value: Bit mapped value representing the ports mode. See product datasheet for details. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getName ()**

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

**Returns**

value: The current name of the port on success. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerEnabled ()**

Gets the current enable value of the power lines. Sub-component (Power) of getEnable.

**Returns****value:**

- 1 = Power enabled
- 0 = Power disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerLimit ()**

Gets the power limit of the port.

**Returns**

value: Active power limit in milli-watts (mW) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerLimitMode ()**

Gets the power limit mode. The mode determines how the port will react to an over power condition.

**Returns**

value: An enumerated representation of the power limit mode Available modes are product specific. See the reference documentation. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerMode ()**

Gets the Port Power Mode: Convenience Function of get/setPortMode

**Returns**

value: The current power mode. See product datasheet for details. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getState ()**

A bit mapped representation of the current state of the port. Reflects what he port IS which may differ from what was requested.

**Returns**

value: Bit mapped value representing the ports state. See product datasheet for details. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getVbusAccumulatedPower ()**

Gets the accumulated Vbus Power.

**Returns**

value: The accumulated power in mWattHours ( $1 == 1e-3$  Wh) currently consumed on VBUS. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getVbusCurrent ()**

Gets the Vbus Current.

**Returns**

value: The current in microamps ( $1 == 1e-6A$ ) currently present on VBUS. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getVbusVoltage ()**

Gets the Vbus Voltage.

**Returns**

value: The voltage in microvolts ( $1 == 1e-6V$ ) currently present on Vbus. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getVconn1Enabled ()**

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled.

**Returns****value:**

- 1 = Vconn1 enabled
- 0 = Vconn1 disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getVconn2Enabled ()**

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled.

**Returns****value:**

- 1 = Vconn2 enabled
- 0 = Vconn2 disabled.

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getVconnAccumulatedPower ()**

Gets the accumulated Vconn Power.

**Returns**

value: The accumulated power in mWattHours ( $1 == 1e-3$  Wh) currently consumed on Vconn. error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getVconnCurrent ()**  
Gets the Vconn Current.  
**Returns**  
value: The current in microamps ( $1 == 1e-6A$ ) currently present on Vconn. error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getVconnEnabled ()**  
Gets the current enable value of the Vconn lines. Sub-component (Vconn) of getEnabled.  
**Returns**  
value:

- 1 = Vconn enabled
- 0 = Vconn disabled.

error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getVconnVoltage ()**  
Gets the Vconn Voltage.  
**Returns**  
value: The voltage in microvolts ( $1 == 1e-6v$ ) currently present on Vconn. error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getVoltageSetpoint ()**  
Gets the current voltage setpoint value of the port.  
**Returns**  
value:  
The voltage setpoint in uV  
error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**resetEntityToFactoryDefaults ()**  
Resets the PortClass Entity to it factory default configuration.  
**Returns (int):**  
Non-zero BrainStem error code on failure.

**resetVbusAccumulatedPower ()**  
Resets the Vbus accumulated power.  
**Returns (int):**  
Non-zero BrainStem error code on failure.

**resetVconnAccumulatedPower ()**  
Resets the Vconn accumulated power.  
**Returns (int):**  
Non-zero BrainStem error code on failure.

**setCC1Enabled (enable)**  
Enables or disables the CC1 lines. Sub-component (CC1) of setEnabled.

**Parameters**

- enable** (*bool*) -
- 1 = Enable CC1 lines
  - 0 = Disable CC1 lines.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setCC2Enabled** (*enable*)

Enables or disables the CC2 lines. Sub-component (CC2) of setEnabled.

**Parameters**

- enable** (*bool*) -
- 1 = Enable CC2 lines
  - 0 = Disable CC2 lines.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setCCEnabled** (*enable*)

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

**Parameters**

- enable** (*bool*) -
- 1 = Enable CC lines
  - 0 = Disable CC lines.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setCurrentLimit** (*limit*)

Sets the current limit of the port.

**Parameters**

- limit** (*int*) – limit to be applied in microAmps (uA)

**Returns (int):**

Non-zero BrainStem error code on failure.

**setCurrentLimitMode** (*mode*)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

**Parameters**

- mode** (*int*) – An enumerated representation of the current limit mode.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataEnabled** (*enable*)

Enables or disables the data lines. Sub-component (Data) of setEnable.

**Parameters**

- enable** (*bool*) -
- 1 = Enable data
  - 0 = Disable data

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataHS1Enabled** (*enable*)

Enables or disables the High Speed 1 side (HS1) data lines. Sub-component of setDataH-SEnable.

**Parameters**

**enable** (*bool*) -  
• 1 = Enable data  
• 0 = Disable data

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataHS2Enabled** (*enable*)

Enables or disables the High Speed 2 side (HS2) data lines. Sub-component of setDataH-SEnable.

**Parameters**

**enable** (*bool*) -  
• 1 = Enable data  
• 0 = Disable data.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataHSEnabled** (*enable*)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnable.

**Parameters**

**enable** (*bool*) -  
• 1 = Enable data  
• 0 = Disable data.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataHSRoutingBehavior** (*mode*)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** (*int*) - An enumerated representation of the routing behavior.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataSS1Enabled** (*enable*)

Enables or disables the Super Speed 1 side (SS1) data lines. Sub-component of setDataEnable.

**Parameters**

**enable** (*bool*) -  
• 1 = Enable data  
• 0 = Disable data.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataSS2Enabled** (*enable*)

Enables or disables the Super Speed 2 side (SS2) data lines. Sub-component of setDataSEnable.

**Parameters**

**enable** (*bool*) -  
• 1 = Enable data  
• 0 = Disable data.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataSSEnabled (enable)**

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnable.

**Parameters**

- enable** (*bool*) -
- 1 = Enable data
  - 0 = Disable data.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataSSRoutingBehavior (mode)**

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

- mode** (*int*) - An enumerated representation of the routing behavior.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setEnabled (enable)**

Enables or disables the entire port.

**Parameters**

- enable** (*bool*) -
- 1 = Enable the port
  - 0 = Disable the port

**Returns (int):**

Non-zero BrainStem error code on failure.

**setHSBoost (boost)**

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Parameters**

- boost** (*int*) - An enumerated representation of the boost range.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setMode (mode)**

Sets the mode of the port.

**Parameters**

- mode** (*int*) - Port mode to be set. See product datasheet for details.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setName (name)**

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

**Parameters**

- name** (*string*) - User defined name to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerEnabled (enable)**

Enables or Disables the power lines. Sub-component (Power) of setEnable.

**Parameters**

- enable** (*bool*) -
- 1 = Enable power
  - 0 = Disable disable.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerLimit** (*limit*)

Sets the power limit of the port.

**Parameters**

- limit** (*int*) - Limit to be applied in milli-watts (mW)

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerLimitMode** (*mode*)

Sets the power limit mode. The mode determines how the port will react to an over power condition.

**Parameters**

- **mode** (*int*) - An enumerated representation of the power limit mode to be applied.
- **documentation**. (*Available modes are product specific. See the reference*) -

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerMode** (*powerMode*)

Sets the Port Power Mode: Convenience Function of get/setPortMode

**Parameters**

- powerMode** (*int*) - Power mode to be set. See product datasheet for details.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setVconn1Enabled** (*enable*)

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

**Parameters**

- enable** (*bool*) -
- 1 = Enable Vconn1 lines
  - 0 = Disable Vconn1 lines.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setVconn2Enabled** (*enable*)

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

**Parameters**

- enable** (*bool*) -
- 1 = Enable Vconn2 lines
  - 0 = Disable Vconn2 lines

**Returns (int):**

Non-zero BrainStem error code on failure.

**setVconnEnabled** (*enable*)

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

**Parameters**

- enable** (*bool*) -
- 1 = Enable Vconn lines
  - 0 = Disable Vconn lines.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setVoltageSetpoint** (*value*)

Sets the current voltage setpoint value of the port

**Parameters**

- value** (*int*) - The voltage setpoint in uV

**Returns (int):**

Non-zero BrainStem error code on failure.

### 3.2.18 Power Delivery

**class brainstem.entity.PowerDelivery** (*module, index*)

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

**get Cable Current Max ()**

Gets the maximum current capability report by the e-mark of the attached cable

**Returns****value (int): An enumerated representation of current**

- Unknown/Unattached (0)
- 3 Amps (1)
- 5 Amps (2)

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**get Cable Orientation ()**

Gets the current orientation being used for PD communication

**Returns****value (int): An enumerated representation of the cables max speed**

- Unconnected = 0
- CC1 (1)
- CC2 (0)

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**get Cable Speed Max ()**

Gets the maximum data rate capability reported by the e-mark of the attached cable.

**Returns****value (int): An enumerated representation of the cables max speed**

- Unknown/Unattached (0)
- USB 2.0 (1)
- USB 3.2 gen 1 (2)
- USB 3.2 / USB 4 gen 2 (3)
- USB 4 gen 3 (4)

error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getCableType ()**

Gets the cable type reported by the e-mark of the attached cable.

**Returns****value (int): An enumerated representation of the cables max speed**

- Invalid, no e-mark and not Vconn powered (0)
- Passive cable with e-mark (1)
- Active cable (2)

error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getCableVoltageMax ()**

Gets the maximum voltage capability reported by the e-mark of the attached cable.

**Returns****value (int): An enumerated representation of voltage**

- Unknown/Unattached (0)
- 20 Volts DC (1)
- 30 Volts DC (2)
- 40 Volts DC (3)
- 50 Volts DC (4)

error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getConnectionState ()**

Gets the current state of the connection in the form of an enumeration.

**Returns**

value (int): An enumerated representation of the current state. error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getFastRoleSwapCurrent ()**

Gets the Fast Role Swap Current. The fast role swap current refers to the amount of current required by the Local Sink in order to successfully perform the swap.

**Returns****value (int): An enumerated value referring to current swap value**

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getFlagMode (*flag*)**

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters***flag* (int) – Flag/Advertisement to be modified

**Returns****value (int): The current mode of the provided flag**

- Disabled (0)
- Enable (1)
- Auto (2) default

error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getNumberOfPowerDataObjects (partner, powerRole)**

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

**Parameters**

- **partner (int)** -
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole (int)** -
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink

**Returns****value (int): The number of of Power Data Objects (PDO) error: Non-zero Brain-Stem error code on failure.****Return type***Result* (object)**getOverride ()**

Gets the current enabled overrides

**Returns****value (int): Bit mapped representation of the current override configuration. error: Non-zero BrainStem error code on failure.****Return type***Result* (object)**getPeakCurrentConfiguration ()**

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

**Returns****value (int): An enumerated value referring to the current configuration.**

- Allowable values are 0 - 4

error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getPowerDataObject (partner, powerRole, ruleIndex)**

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

**Parameters**

- **partner (int)** -
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole (int)** -
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex (int)** -

**Returns****value (int): Power Data Object (PDO) for the given partner and power role.**

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerDataObjectEnabled** (*powerRole*, *ruleIndex*)

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

**Parameters**

- **powerRole** (*int*) –
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*int*) – The index of the PDO in question. Valid index are 1-7.

**Returns**

value (bool): Represents the enabled state. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerDataObjectEnabledList** (*powerRole*)

Gets all Power Data Object enables for a given power role. Equivalent of calling PowerDelivery.getPowerDataObjectEnabled() for all indexes.

**Parameters**

- powerRole** (*int*) –
- Source = 1 = powerdeliveryPowerRoleSource (1) Source
  - Sink = 2 = powerdeliveryPowerRoleSink (2) Sink

**Returns**

value (bool): Bit mapped representation of the enabled PDOs for a given power role Values align with a given rule index (bits 1-7, bit 0 is invalid). error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerDataObjectList** ()

Gets all Power Data Objects (PDO) for a given partner and power role. Equivalent to calling PowerDelivery.getPowerDataObject() on all partners, power roles, and index's.

**Returns**

value (tuple(*int*)): All Power Data Objects (PDOs) On success the length should be 28 (7 rules \* 2 partners \* 2 power roles) The order of which is:

- Rules 1-7 Local Source
- Rules 1-7 Local Sink
- Rules 1-7 Remote Source
- Rules 1-7 Remote Sink

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerRole** ()

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

**Returns**

**value (int): The current power role**

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1= powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerRolePreferred()**

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

**Returns****value (int): The current power role**

- Disabled = 0 = powerdeliveryPowerRoleDisabled
- Source = 1 = powerdeliveryPowerRoleSource
- Sink = 2 = powerdeliveryPowerRoleSink

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getRequestDataObject (partner)**

Gets the current Request Data Object (RDO) for a given partner.

**RDOs:**

- Are provided by the sinking device.
- Exist only after a successful PD negotiation (Otherwise zero).
- Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

**Parameters****partner (int) -**

- Local = 0 = powerdeliveryPartnerLocal
- Remote = 1 = powerdeliveryPartnerRemote

**Returns****value (int): Value of the request RDO. (Zero indicates the RDO is not active)**

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**request (request)**

Requests an action of the Remote partner. Actions are not guaranteed to occur.

**Parameters****request (int) -**

- pdRequestHardReset (0)
- pdRequestSoftReset (1)
- pdRequestDataReset (2)
- pdRequestPowerRoleSwap (3)
- pdRequestPowerFastRoleSwap (4)
- pdRequestDataRoleSwap (5)
- pdRequestVconnSwap (6)
- pdRequestSinkGoToMinimum (7)
- pdRequestRemoteSourcePowerDataObjects (8)
- pdRequestRemoteSinkPowerDataObjects (9)

**Returns (int):**

Non-zero BrainStem error code on failure.

**requestStatus ()**

Gets the status of the last request command sent.

**Returns**

value (int): the request status error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**resetEntityToFactoryDefaults ()**

Resets the PowerDelivery Entity to it factory default configuration.

**Returns (int):**

Non-zero BrainStem error code on failure.

**resetPowerDataObjectToDefault (powerRole, ruleIndex)**

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

**Parameters**

- **powerRole** (*int*) –
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** (*int*) – The index of the PDO in question. Valid index are 1-7.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setFastRoleSwapCurrent (current)**

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

**Parameters**

- **current** (*int*) – An enumerated value referring to value to be set. - 0A (0) - 900mA (1) - 1.5A (2) - 3A (3)

**Returns (int):**

Non-zero BrainStem error code on failure.

**setFlagMode (flag, mode)**

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters**

- **flag** (*int*) – Flag/Advertisement to be modified
- **mode** (*int*) –
  - Disabled (0)
  - Enable (1)
  - Auto (2) default

**Returns (int):**

Non-zero BrainStem error code on failure.

**setOverride (overrides)**

Sets the current overrides.

**Parameters**

- **overrides** (*int*) – Overrides to be set in a bit mapped representation.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPeakCurrentConfiguration (config)**

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

**Parameters**

**config (int)** – An enumerated value referring to the configuration to be set -  
Allowable values are 0 - 4

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerDataObject (powerRole, ruleIndex, pdo)**

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

**Parameters**

- **powerRole (int)** –
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex (int)** – The index of the PDO in question. Valid index are 1-7.
- **pdo (int)** – Power Data Object to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerDataObjectEnabled (powerRole, ruleIndex, enable)**

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

**Parameters**

- **powerRole (int)** –
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex (int)** – The index of the PDO in question. Valid index are 1-7.
- **enable (bool)** – The enabled state to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerRole (powerRole)**

Sets the power role to be advertised by the Local partner. (CC Strapping).

**Parameters**

- powerRole (int)** –
- Disabled = 0 = powerdeliveryPowerRoleDisabled
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
  - Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerRolePreferred (powerRole)**

Set the preferred power role to be advertised by the Local partner (CC Strapping).

**Parameters**

- powerRole (int)** –
- Disabled = 0 = powerdeliveryPowerRoleDisabled
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink

**Returns (int):**

Non-zero BrainStem error code on failure.

**setRequestDataObject (partner, rdo)**

Sets the current Request Data Object (RDO) for a given partner. (Only the local partner can be changed.)

**RDOs:**

- Are provided by the sinking device.
- Exist only after a successful PD negotiation (Otherwise zero).
- Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

**Parameters**

- **partner** (*int*) – Indicates which side of the PD connection is in question.  
Local = 0 = powerdeliveryPartnerLocal
- **rdo** (*int*) – RDO to be applied.

**Returns (int):**

Non-zero BrainStem error code on failure.

### 3.2.19 Rail

**class brainstem.entity.Rail (module, index)**

Provides power rail functionality on certain modules.

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

**Useful Constants:**

- KELVIN\_SENSING\_OFF (0)
- KELVIN\_SENSING\_ON (1)
- OPERATIONAL\_MODE\_AUTO (0)
- OPERATIONAL\_MODE\_LINEAR (1)
- OPERATIONAL\_MODE\_SWITCHER (2)
- OPERATIONAL\_MODE\_SWITCHER\_LINEAR (3)
- DEFAULT\_OPERATIONAL\_MODE (OPERATIONAL\_MODE\_AUTO)
- OPERATIONAL\_STATE\_INITIALIZING (0)
- OPERATIONAL\_STATE\_ENABLED (1)
- OPERATIONAL\_STATE\_FAULT (2)
- OPERATIONAL\_STATE\_HARDWARE\_CONFIG (8)
- OPERATIONAL\_STATE\_LINEAR (0)
- OPERATIONAL\_STATE\_SWITCHER (1)
- OPERATIONAL\_STATE\_LINEAR\_SWITCHER (2)
- OPERATIONAL\_STATE\_OVER\_VOLTAGE\_FAULT (16)
- OPERATIONAL\_STATE\_UNDER\_VOLTAGE\_FAULT (17)
- OPERATIONAL\_STATE\_OVER\_CURRENT\_FAULT (18)
- OPERATIONAL\_STATE\_OVER\_POWER\_FAULT (19)
- OPERATIONAL\_STATE\_REVERSE\_POLARITY\_FAULT (20)

- OPERATIONAL\_STATE\_OVER\_TEMPERATURE\_FAULT (21)
- OPERATIONAL\_STATE\_OPERATING\_MODE (24)
- OPERATIONAL\_STATE\_CONSTANT\_CURRENT (0)
- OPERATIONAL\_STATE\_CONSTANT\_VOLTAGE (1)
- OPERATIONAL\_STATE\_CONSTANT\_POWER (2)
- OPERATIONAL\_STATE\_CONSTANT\_RESISTANCE (3)

**clearFaults ()**

Clears the current fault state of the rail.

Refer to the module datasheet for definition of the rail faults.

**Returns**

Return result object with NO\_ERROR set or an Error.

**Return type**

*Result*

**getCurrent ()**

Get the rail current.

**Returns**

**Result object, containing NO\_ERROR and the current in microamps**  
or a non zero Error code.

**Return type**

*Result*

**getCurrentLimit ()**

Get the rail current limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**microamps (int)** – The current in micro-amps (1 == 1e-6A).

**Returns**

**Return result object with NO\_ERROR set and the current**  
limit setting in the Result.value or an Error condition.

**Return type**

*Result*

**getCurrentSetpoint ()**

Get the rail setpoint current.

Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

**Returns**

Return result object with NO\_ERROR set and the current rail current setting in the Result.value or an Error.

**Return type**

*Result*

**getEnable ()**

Get the state of the rail switch.

Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

**Returns**

Return result object with NO\_ERROR set and the current rail enable state in the Result.value or an Error condition.

**Return type***Result***getKelvinSensingEnable ()**

Determine whether kelvin sensing is enabled or disabled.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

**bEnable** (*bool*) – Kelvin sensing is enabled or disabled.

**Returns**

Return result object with NO\_ERROR set and the current rail kelvin sensing mode setting in the Result.value or an Error.

**Return type***Result***getKelvinSensingState ()**

Determine whether kelvin sensing has been disabled by the system.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Returns**

Return result object with NO\_ERROR set and the current rail kelvin sensing state setting in the Result.value or an Error.

**Return type***Result***getOperationalMode ()**

Determine the current operational mode of the system.

Refer to the module datasheet for definition of the rail operational mode capabilities.

**Returns**

Return result object with NO\_ERROR set and the current rail operational mode setting in the Result.value or an Error.

**Return type***Result***getOperationalState ()**

Determine the current operational state of the system.

Refer to the module datasheet for definition of the rail operational states.

**Returns**

Return result object with NO\_ERROR set and the current rail operational state in the Result.value or an Error.

**Return type***Result***getPower ()**

Get the rail power.

**Returns**

**Result object, containing NO\_ERROR and the power in milliwatts**  
or a non zero Error code.

**Return type***Result***getPowerLimit ()**

Get the rail power limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**milliwatts** (*int*) – The power in milli-watts (1 == 1e-3W).

**Returns**

**Return result object with NO\_ERROR set and the power limit setting in the Result.value or an Error condition.**

**Return type**

*Result*

**getPowerSetpoint ()**

Get the rail setpoint power.

Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Returns**

Return result object with NO\_ERROR set and the power rail power setting in the Result.value or an Error.

**Return type**

*Result*

**getResistance ()**

Get the rail resistance.

**Returns**

**Result object, containing NO\_ERROR and the resistance in milliohms or a non zero Error code.**

**Return type**

*Result*

**getResistanceSetpoint ()**

Get the rail setpoint resistance.

Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

**Returns**

Return result object with NO\_ERROR set and the resistance rail resistance setting in the Result.value or an Error.

**Return type**

*Result*

**getTemperature ()**

Get the rail temperature.

**Returns**

Return result object with NO\_ERROR set and the rail temperature in the Result.value or an Error condition.

**Return type**

*Result*

**getVoltage ()**

Get the rail supply voltage.

Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

On some modules this is a measured value so may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Return result object with NO\_ERROR set and the current rail voltage setting in the Result.value or an Error.

**Return type***Result***getVoltageMaxLimit ()**

Get the rail voltage maximum limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**microvolts** (*int*) – The voltage maximum in micro-volts (1 == 1e-6V).

**Returns**

**Return result object with NO\_ERROR set and the voltage minimum limit setting in the Result.value or an Error condition.**

**Return type***Result***getVoltageMinLimit ()**

Get the rail voltage minimum limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**microvolts** (*int*) – The voltage minimum in micro-volts (1 == 1e-6V).

**Returns**

**Return result object with NO\_ERROR set and the voltage minimum limit setting in the Result.value or an Error condition.**

**Return type***Result***getVoltageSetpoint ()**

Get the rail setpoint voltage.

Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Returns**

Return result object with NO\_ERROR set and the current rail voltage setting in the Result.value or an Error.

**Return type***Result***setCurrentLimit (*microamps*)**

Set the rail current limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**microamps** (*int*) – The current in micro-amps (1 == 1e-6A).

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type***Result.error***setCurrentSetpoint (*microamps*)**

Set the rail supply current.

Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

**Parameters**

**microamps** (*int*) – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setEnable (bEnable)**

Set the state of the rail switch.

Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

**Parameters**

**bEnable (bool)** – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setKelvinSensingEnable (bEnable)**

Enable or Disable kelvin sensing on the module.

Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

**bEnable (bool)** – enable or disable kelvin sensing.

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setOperationalMode (mode)**

Set the operational mode of the rail.

Refer to the module datasheet for definition of the rail operational capabilities.

**Parameters**

**mode (int)** – The operational mode to employ.

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setPowerLimit (milliwatts)**

Set the rail power limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**milliwatts (int)** – The power in milli-watts ( $1 == 1e-3W$ ).

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setPowerSetpoint** (*milliwatts*)

Set the rail supply power.

Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

**milliwatts** (*int*) – The power in milli-watts (1 == 1e-3W) to be supply by the rail.

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setResistanceSetpoint** (*millionohms*)

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**

**millionohms** (*int*) – The resistance in milli-ohms (1 == 1e-3Ohms) to be applied to the rail.

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setVoltageMaxLimit** (*microvolts*)

Set the rail voltage maximum limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**microvolts** (*int*) – The voltage maximum in micro-volts (1 == 1e-6V).

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setVoltageMinLimit** (*microvolts*)

Set the rail voltage minimum limit setting.

Check product datasheet to see if this feature is available.

**Parameters**

**microvolts** (*int*) – The voltage minimum in micro-volts (1 == 1e-6V).

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

Result.error

**setVoltageSetpoint** (*microvolts*)

Set the rail supply voltage.

Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

**microvolts** (*int*) – The voltage in micro-volts (1 == 1e-6V) to be supply by the rail.

**Returns**

**Return NO\_ERROR on success, or one of the common sets of return error codes on failure.**

**Return type**

*Result.error*

### 3.2.20 RCservo

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

Provides RCservo functionality on certain modules.

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

**Useful Constants:**

- SERVO\_DEFAULT\_POSITION (128)
- SERVO\_DEFAULT\_MIN (64)
- SERVO\_DEFAULT\_MAX (192)

**getEnable ()**

Get the enable status of the servo channel.

If the enable status is 0 the servo channel is disabled, if the status is 1 the channel is enabled.

**Returns**

**Result object, containing NO\_ERROR and servo enable status or a non zero Error code.**

**Return type**

*Result*

**getPosition ()**

Get the position of the servo channel.

Default configuration: 1ms = 64 and 2ms = 192

**Returns**

**Result object, containing NO\_ERROR and the servo position or a non zero Error code.**

**Return type**

*Result*

**getReverse ()**

Get the reverse status of the channel.

0 = not reversed, 1 = reversed

**Returns**

**Result object, containing NO\_ERROR and the reverse status or a non zero Error code.**

**Return type**

*Result*

**setEnable (enable)**

Enable the servo channel.

**Param:**

enable (bool): The state to be set. 0 is disabled, 1 is enabled

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setPosition (position)**

Set the position of the servo channel.

**Param:**

position (int): The position to be set. With the default configuration 64 = a 1ms pulse and 192 = a 2ms pulse.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setReverse (reverse)**

Set the output to be reverse on the servo channel.

**Param:**

reverse (bool): Reverse mode: 0 = not reversed, 1 = reversed. ie: setPosition of 64 would actually apply 192 to the servo output; however, getPosition will return 64.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

### 3.2.21 Relay

**class brainstem.entity.Relay (module, index)**

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

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

**Useful Constants:**

- VALUE\_LOW (0)
- VALUE\_HIGH (1)

**getEnable ()**

Get the relay enable state.

A return of 1 indicates the relay is enabled. A return of 0 indicates the relay is disabled.

**Returns**

**Result object, containing NO\_ERROR and digital state**  
or a non zero Error code.

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

Get the scaled micro volt value with reference to ground.

Get a 32 bit signed integer (in micro Volts) 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 NO\_ERROR and microVolts value**

or a non zero Error code.

**Return type***Result***setEnable (bEnable)**

Enables or disables the relay based on bEnable.

**Param:**

bEnable (int): Set 1 for enable, set 0 for disable.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type***Result.error*

## 3.2.22 Results

A module that provides a result class for returning results of UEI commands.

Results consist of an error attribute and a value attribute. If the error attribute is set to NO\_ERROR, then the result value is the response to the UEI command that was sent.

For more information about return values for commands and UEI's see the [Acroname BrainStem Reference](#)<sup>19</sup>

**class brainstem.result.Result (error, value)**

Result class for returning results of commands

Instances of Result represent the response to a command. The Result class also contains constants representing the possible errors that may be encountered during interaction with a BrainStem module.

**property error**

Return the error attribute

**property value**

Return the value attribute

---

<sup>19</sup> <https://acroname.com/reference>

### 3.2.23 Signal

See the [Signal Entity](#) for generic information.

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

The Signal Class is the interface to digital pins configured to produce square wave signals.

This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

**getEnable()**

Get the Enable/Disable of the signal.

**Returns**

Result object, containing NO\_ERROR and boolean value True for enabled  
False for disabled or a non zero Error code.

**getInvert()**

Get the invert status of the signal.

**Returns**

Result object, containing NO\_ERROR and boolean value True for inverted  
False for normal or a non zero Error code.

**getT2Time()**

Get the current wave active period (T2) in nanoseconds.

**Returns**

Result object, containing NO\_ERROR and an integer value not larger than the  
max unsigned 32bit value. or a non zero Error code.

**getT3Time()**

Get the current wave period (T3) in nanoseconds.

**Returns**

Result object, containing NO\_ERROR and an integer value not larger than the  
max unsigned 32bit value. or a non zero Error code.

**setEnable(enable)**

Enable/Disable the signal output.

**Parameters**

`enable` – True to enable, false to disable

**Returns**

Result.error Return NO\_ERROR on success, or one of the common sets of  
return error codes on failure.

**setInvert(invert)**

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and  
high at t2.

**Parameters**

`invert` – True to invert, false for normal mode.

**Returns**

Result.error Return NO\_ERROR on success, or one of the common sets of  
return error codes on failure.

**setT2Time(t2\_nsec)**

Set the signal active period or T2 in nanoseconds.

**Parameters**

**t2\_nsec** – Tninteger not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

**Returns**

Result.error Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**setT3Time (t3\_nsec)**

Set the signal period or T3 in nanoseconds.

**Parameters**

**t3\_nsec** – Tninteger not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

**Returns**

Result.error Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

### 3.2.24 System

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

Access system controls configuration and information.

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

**Useful Constants:**

- BOOT\_SLOT\_DISABLE (255)

**getBootSlot ()**

Get the store slot which is mapped when the module boots.

**Returns**

**Result object, containing NO\_ERROR and slot number**  
or a non zero Error code.

**Return type**

*Result*

**getErrors ()**

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

**Returns**

value: Bit mapped value representing the errors. See product datasheet for details.  
error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getHBInterval ()**

Get the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

**Returns**

**Result object, containing NO\_ERROR and the Heartbeat interval**  
or a non zero Error code.

**Return type**

*Result*

**getHardwareVersion()**

Get the module's hardware revision information.

The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

**Returns**

**Result object, containing NO\_ERROR and hardware revision**  
or a non zero Error code.

**Return type**

*Result*

**getInputCurrent()**

Get the module's input current.

**Returns**

**Result object, containing NO\_ERROR and input current**  
or a non zero Error code.

**Return type**

*Result*

**getInputPowerBehavior()****Gets the systems input power behavior.**

This behavior refers to where the device sources its power from and what happens if that power source goes away.

**Returns**

value (int): an enumerated value representing behavior. error: Non-zero Brain-Stem error code on failure.

**Return type**

*Result* (object)

**getInputPowerBehaviorConfig()****Gets the input power behavior configuration**

Certain behaviors use a list of ports to determine priority when budgeting power.

**Returns**

value (tuple(int)): A list of ports which indicate priority sequencing. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getInputPowerSource()**

Provides the source of the current power source in use.

**Returns**

value (int): An enumerated value representing the current input power source.  
error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getInputVoltage()**

Get the module's input voltage.

**Returns**

**Result object, containing NO\_ERROR and input voltage**  
or a non zero Error code.

**Return type**

*Result*

**getLED ()**

Get the system LED state.

Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

**Returns**

**Result object, containing NO\_ERROR and the LED State**  
or a non zero Error code.

**Return type**

*Result*

**getLinkInterface ()**

**Gets the link interface configuration.**

This refers to which interface is being used for control by the device.

**Returns**

**value (int): an enumerated value representing interface.**

- 0 = Auto
- 1 = Control Port
- 2 = Hub Upstream Port

error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getMaximumTemperature ()**

Get the module's maximum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

**Returns**

**Result object, containing NO\_ERROR and max temperature in micro-C**  
or a non zero Error code.

**Return type**

*Result*

**getMinimumTemperature ()**

Get the module's minimum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

**Returns**

**Result object, containing NO\_ERROR and max temperature in micro-C**  
or a non zero Error code.

**Return type**

*Result*

**getModel ()**

Get the module's model enumeration.

A subset of the possible model enumerations is defined in aProtocolDefs.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

**Returns**

**Result object, containing NO\_ERROR and model number**  
or a non zero Error code.

**Return type**

*Result*

**getModule ()**

Get the address the module uses on the BrainStem network.

**Returns**

**Result object, containing NO\_ERROR and the current module address**  
or a non zero Error code.

**Return type**

*Result*

**getModuleBaseAddress ()**

Get the base address the module.

The software and hardware addresses are added to the base address to produce the effective module address.

**Returns**

**Result object, containing NO\_ERROR and the current module address**  
or a non zero Error code.

**Return type**

*Result*

**getModuleHardwareOffset ()**

Get the module address hardware offset.

This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

**Returns**

**Result object, containing NO\_ERROR and module offset**  
or a non zero Error code.

**Return type**

*Result*

**getModuleSoftwareOffset ()**

Get the module address software offset.

The address offset that is added to the module base address, and potentially the hardware offset to produce the module effective address.

**Returns**

**Result object, containing NO\_ERROR and the current module address**  
or a non zero Error code.

**Return type**

*Result*

**getName ()**

**Gets a user defined name of the port.**

Helpful for identifying ports/devices in a static environment.

**Returns**

value: The current name of the port on success. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerLimit ()**

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

**Returns**

value (int): Power limit in milli-watts (mW) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerLimitMax()****Gets the user defined maximum power limit for the system.**

Provides mechanism for defining an unregulated power supplies capability.

**Returns**

value (int): Power limit in milli-watts (mW) error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getPowerLimitState()****Gets a bit mapped representation of the factors contributing to the power limit.**

Active limit can be found through PowerDeliveryClass::getPowerLimit().

**Returns**

value (int): The current power limit state. error: Non-zero BrainStem error code on failure.

**Return type***Result* (object)**getRouter()**

Get the router address the module uses to communicate with the host.

**Returns****Result object, containing NO\_ERROR and the current router address**  
or a non zero Error code.**Return type***Result***getRouterAddressSetting()**

Get the router address setting saved in the module.

This setting may be different from the effective router if the router has been set and saved but no reset has been made.

**Returns****Result object, containing NO\_ERROR and the current router address**  
or a non zero Error code.**Return type***Result***getSerialNumber()**

Get the module's serial number.

The serial number is a unique 32bit integer which is usually communicated in hexadecimal format.

**Returns****Result object, containing NO\_ERROR and serial number**  
or a non zero Error code.**Return type***Result***getTemperature()**

Get the module's current temperature in micro-C

**Returns****Result object, containing NO\_ERROR and max temperature in micro-C**  
or a non zero Error code.**Return type***Result*

**getUnregulatedCurrent ()**

Gets the current present at the unregulated port.

**Returns**

value (int): Unregulated current in micro-amps (mA) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getUnregulatedVoltage ()**

Gets the voltage present at the unregulated port.

**Returns**

value (int): Unregulated Voltage in micro-volts (mV) error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getUptime ()**

Get accumulated system uptime.

This is the total time the system has been powered up with the firmware running. The returned uptime is a count of minutes of uptime, or may be a module dependent counter.

**Returns**

**Result object, containing NO\_ERROR and uptime in minutes**  
or a non zero Error code.

**Return type**

*Result*

**getVersion ()**

Get the modules firmware version number.

The version number is packed into the return value. Utility functions in the Version module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

**Returns**

**Result object, containing NO\_ERROR packed version number**  
or a non zero Error code.

**Return type**

*Result*

**logEvents ()**

Save system log entries to slot defined by module.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

**reset ()**

Reset the system.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

**resetDeviceToFactoryDefaults ()**

Resets the device to it factory default configuration.

**Returns (int):**

Non-zero BrainStem error code on failure.

**resetEntityToFactoryDefaults ()**

Resets the SystemClass Entity to its factory default configuration.

**Returns (int):**

Non-zero BrainStem error code on failure.

**routeToMe (value)**

Enables/Disables the route to me function.

This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

**Parameters**

**value (int)** – Enable or disable of the route to me function 1 = enable.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**save ()**

Save the system operating parameters to the persistent module flash memory.

Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setBootSlot (value)**

Set a store slot to be mapped when the module boots.

The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

**Parameters**

**value (int)** – The slot number in aSTORE\_INTERNAL to be marked as a boot slot.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setHBInterval (value)**

Set the delay between heartbeat packets.

For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments. Valid values are 1-255; default is 10 (256 milliseconds).

**Parameters**

**value** (*int*) – Heartbeat interval settings.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setInputPowerBehavior** (*behavior*)

**Sets the systems input power behavior.**

This behavior refers to where the device sources its power from and what happens if that power source goes away.

**Parameters**

**behavior** (*int*) – An enumerated representation of behavior to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setInputPowerBehaviorConfig** (*config*)

**Sets the input power behavior configuration**

Certain behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

**config** (*tuple(int)*) – List of ports which indicate priority sequencing.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setLED** (*value*)

Set the system LED state.

Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

**Parameters**

**value** (*int*) – LED State setting.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setLinkInterface** (*linkInterface*)

**Sets the link interface configuration.**

This refers to which interface is being used for control by the device.

**Parameters**

**interface** (*int*) – An enumerated representation of interface to be set. \* 0 = Auto= systemLinkAuto \* 1 = Control Port = systemLinkUSBControl \* 2 = Hub Upstream Port = systemLinkUSBHub

**Returns (int):**

Non-zero BrainStem error code on failure.

**setModuleSoftwareOffset** (*value*)

Set the software address offset.

The module software offset is added to the base module address, and potentially a hardware offset to determine the final calculated address the module uses on

the BrainStem network. You must save and reset the module for this change to become effective.

**Warning:**

changing the module address may cause the module to “drop off” the BrainStem network if the module is also the router. Please review the BrainStem network fundamentals before modifying the module address.

**Parameters**

**value** (*int*) – The module address offset.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

**setName** (*name*)

**Sets a user defined name of the port.**

Helpful for identifying ports/devices in a static environment.

**Parameters**

**name** (*string*) – User defined name to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerLimitMax** (*limit*)

**Sets a user defined maximum power limit for the system.**

Provides mechanism for defining an unregulated power supplies capability.

**Parameters**

**limit** (*int*) – Limit in milli-watts (mW) to be set

**Returns (int):**

Non-zero BrainStem error code on failure.

**setRouter** (*value*)

Set the router address the module uses to communicate with the host.

**Warning:**

Changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

**Parameters**

**value** (*int*) – The module address of the router module on the network.

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

Result.error

### 3.2.25 Store

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

Access the store on a BrainStem Module.

The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure.

Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

#### Useful Constants:

- INTERNAL\_STORE (0)
- RAM\_STORE (1)
- SD\_STORE (2)
- EEPROM\_STORE (3)

`getSlotCapacity(slot)`

Get the slot capacity.

Returns the Capacity of the slot, i.e. The number of bytes it can hold.

#### return: Result:

Either the capacity of the slot in Result.value or an error.

`getSlotLocked(slot)`

#### Gets the current lock state of the slot

Allows for write protection on a slot.

##### Parameters

`slot (int)` – The slot number

##### Returns

`value`: The current locked state of the provided slot. `error`: Non-zero BrainStem error code on failure.

##### Return type

`Result` (object)

`getSlotSize(slot)`

Get the slot size.

Returns the size of the data currently filling the slot in bytes.

#### return: Result:

Either the size of the slot in Result.value or an error.

`getSlotState(slot)`

Get slot state.

Slots which contain reflexes may be “enabled,” i.e. the reflexes contained in the slot are active.

##### Parameters

`slot (int)` – The slot number.

**return: Result:**

Return result object with NO\_ERROR set and the current state of the slot in the Result.value or an Error.

**loadSlot (slot, data, \_=None)**

Load the slot.

**Parameters**

- **slot** (*int*) – The slot number.
- **data** (*str, bytes*) – The data.
- **\_** (*int*) – (length Deprecated) Unused parameter, and will be removed in next minor release.

**return: Result.error:**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**setSlotLocked (slot, lock)****Sets the locked state of the slot**

Allows for write protection on a slot.

**Parameters**

- **slot** (*int*) – The slot number
- **lock** (*bool*) – Locked state to set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**slotDisable (slot)**

Disable the slot

**slotEnable (slot)**

Enable the slot

**unloadSlot (slot)**

Unload the slot data.

**Parameters**

- **slot** (*int*) – The slot number.

**return: Result:**

Either Returns Result object with NO\_ERROR set and its value attribute set with an object of type bytes containing the unloaded data. Or a Result object with a non-zero error.

### 3.2.26 Temperature

**class brainstem.entity.Temperature (module, index)**

Provide interface to temperature sensor.

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

**getValue ()**

Get the current temperature in micro-C.

**Returns**

value: The temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getValueMax ()**

Get the module's maximum temperature in micro-C since the last power cycle.

**Returns**

value: The maximum temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**getValueMin ()**

Get the module's minimum temperature in micro-C since the last power cycle.

**Returns**

value: The minimum temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**reset ()**

Get the module's maximum temperature in micro-C since the last power cycle.

**Returns**

value: The maximum temperature in micro-C (uC) error: Non-zero BrainStem error code on failure.

**Return type**  
*Result* (object)

**resetEntityToFactoryDefaults ()**

Resets the TemperatureClass Entity to its factory default configuration.

**Returns (int):**

Non-zero BrainStem error code on failure.

### 3.2.27 Timer

**class** brainstem.entity.Timer (*module, index*)

Schedules events to occur at future times.

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

#### Useful Constants:

- SINGLE\_SHOT\_MODE (0)
- REPEAT\_MODE (1)
- DEFAULT\_MODE (SINGLE\_SHOT\_MODE)

**getExpiration ()**

Get the currently set expiration time in microseconds.

This is not a “live” timer. That is, it shows the expiration time originally set with setExpiration; it does not “tick down” to show the time remaining before expiration.

**Returns**

Return result object with NO\_ERROR set and the timer expiration in uSeconds in the Result.value or an Error.

**Return type***Result***getMode ()**

Get the mode of the timer.

Valid timer modes are single mode and repeat mode.

**Returns**

Return result object with NO\_ERROR set and the timer mode either single (0) or repeat (1) in the Result.value or an Error.

**Return type***Result***setExpiration (usecDuration)**

Set the expiration time for the timer entity.

When the timer expires, it will fire the associated timer[index]() reflex.

**Parameters**

**usecDuration (int)** – The duration before timer expiration in microseconds.

**Returns**

**Return NO\_ERROR on success, or one of the common**

sets of return error codes on failure.

**Return type***Result.error***setMode (mode)**

Set the mode of the timer.

**Parameters**

**mode (int)** – The mode of the timer. aTIMER\_MODE\_REPEAT or aTIMER\_MODE\_SINGLE.

**Returns**

**Return NO\_ERROR on success, or one of the common**

sets of return error codes on failure.

**Return type***Result.error*

### 3.2.28 UART

**class brainstem.entity.UART (module, index)**

Provides UART entity access on certain BrainStem modules.

A UART is a “Universal Asynchronous Reciever/Transmitter”. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines

**getBaudRate ()**

Get the Baud rate of the UART.

**Returns**

**Result object, containing NO\_ERROR and the UART baud rate**  
or a non zero Error code.

**Return type***Result***getEnable ()**

Get the enable status of the UART.

**Returns**

**Result object, containing NO\_ERROR and the UART state**  
or a non zero Error code.

**Return type**

*Result*

**getProtocol ()**

Get the protocol format of the UART.

**Returns**

**Result object, containing NO\_ERROR and the UART protocol**  
or a non zero Error code.

**Return type**

*Result*

**setBaudRate (rate)**

Set the Baud rate of the UART.

**Param:**

rate (int):

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

**setEnable (bEnable)**

Enable the UART.

**Param:**

bEnable (bool): True = Enable, False = Disable

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

**setProtocol (protocol)**

Set the protocol format of the UART.

**Param:**

protocol (int):

**Returns**

Return NO\_ERROR on success, or one of the common sets of return error codes on failure.

**Return type**

*Result.error*

### 3.2.29 USB

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

USBClass provides methods to interact with a USB hub and USB switches.

Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

#### Useful Constants:

- UPSTREAM\_MODE\_AUTO (2)
- UPSTREAM\_MODE\_PORT\_0 (0)
- UPSTREAM\_MODE\_PORT\_1 (1)
- UPSTREAM\_MODE\_NONE (255)
- DEFAULT\_UPSTREAM\_MODE (UPSTREAM\_MODE\_AUTO)
- UPSTREAM\_STATE\_PORT\_0 (0)
- UPSTREAM\_STATE\_PORT\_1 (1)
- BOOST\_0\_PERCENT (0)
- BOOST\_4\_PERCENT (1)
- BOOST\_8\_PERCENT (2)
- BOOST\_12\_PERCENT (3)
- PORT\_MODE\_SDP (0)
- PORT\_MODE\_CDP (1)
- PORT\_MODE\_CHARGING (2)
- PORT\_MODE\_PASSIVE (3)
- PORT\_MODE\_USB2\_A\_ENABLE (4)
- PORT\_MODE\_USB2\_B\_ENABLE (5)
- PORT\_MODE\_VBUS\_ENABLE (6)
- PORT\_MODE\_SUPER\_SPEED\_1\_ENABLE (7)
- PORT\_MODE\_SUPER\_SPEED\_2\_ENABLE (8)
- PORT\_MODE\_USB2\_BOOST\_ENABLE (9)
- PORT\_MODE\_USB3\_BOOST\_ENABLE (10)
- PORT\_MODE\_AUTO\_CONNECTION\_ENABLE (11)
- PORT\_MODE\_CC1\_ENABLE (12)
- PORT\_MODE\_CC2\_ENABLE (13)
- PORT\_MODE\_SBU\_ENABLE (14)
- PORT\_MODE\_CC\_FLIP\_ENABLE (15)
- PORT\_MODE\_SS\_FLIP\_ENABLE (16)
- PORT\_MODE\_SBU\_FLIP\_ENABLE (17)
- PORT\_MODE\_USB2\_FLIP\_ENABLE (18)

- PORT\_MODE\_CC1\_INJECT\_ENABLE (19)
- PORT\_MODE\_CC2\_INJECT\_ENABLE (20)
- PORT\_SPEED\_NA (0)
- PORT\_SPEED\_HISPEED (1)
- PORT\_SPEED\_SUPERSPEED (2)

**clearPortErrorStatus (channel)**

Clear the error status for the given channel.

**Parameters**

**channel (int)** – The USB port number

Returns: Result object

**getAltModeConfig (channel)**

**Gets alt mode configuration for defined USB channel.**

See the product datasheet for device specific details.

**Parameters**

**channel (int)** – The USB channel

Returns: Result object

**getCC1Current (channel)**

Get the current through the CC1 line for a port.

**Parameters**

**channel (int)** – The USB port number

Returns: Result object

**getCC1Voltage (channel)**

Get the voltage on the CC1 line for a port.

**Parameters**

**channel (int)** – The USB port number

Returns: Result object

**getCC2Current (channel)**

Get the current through the CC2 line for a port.

**Parameters**

**channel (int)** – The USB port number

Returns: Result object

**getCC2Voltage (channel)**

Get the voltage on the CC2 line for a port.

**Parameters**

**channel (int)** – The USB port number

Returns: Result object

**getCableFlip (channel)**

Get the status of cable orientation flip within the S85 switch.

**Parameters**

**channel (int)** – The USB port number

Returns: Result object

**getConnectMode (channel)**

Get The connection mode for the Switch.

**Parameters**

**channel (int)** – Upstream port to be applied.

**Returns**

value (int): The connect mode error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getDownstreamBoostMode ()**

Get the downstream boost mode.

**Returns: Result object**

Result value 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

**getDownstreamDataSpeed (*channel*)**

Get the downstream port data speed.

**Returns: Result object****Result value:**

- N/A: PORT\_SPEED\_NA = 0
- Hi Speed: PORT\_SPEED\_HISPEED = 1
- SuperSpeed: PORT\_SPEED\_SUPERSPEED = 2

**getEnumerationDelay ()**

Get the interport enumeration delay in milliseconds.

Returns: Result object

**getHubMode ()****Get a bit mapped representation of the hub mode.**

Usually represents the port data and power lines enable/disable state in one bit packed result. See the product datasheet for state mapping.

Returns: Result object

**getPortCurrent (*channel*)**

Get the current through the power line for a port.

**Parameters**

*channel* (*int*) – The USB port number

Returns: Result object

**getPortCurrentLimit (*channel*)**

Get the current limit for the port.

Returns: Result object

**getPortError (*channel*)**

Get the error for the Port.

Returns: Result object

**getPortMode (*channel*)**

Get the mode for the Port. The mode setting defaults to SDP or Standard Downstream port, and can be set to CDP (charging downstream port) for devices that require high port charge current above 500 millamps.

Returns: Result object

**getPortState (*channel*)**

Get the state for the Port.

Returns: Result object

**getPortVoltage (channel)**

Get the voltage on the power line for a port.

**Parameters**

- **channel (int)** – The USB port number

Returns: Result object

**getSBU1Voltage (channel)**

Get the voltage on the SBU1 line for a port.

**Parameters**

- **channel (int)** – The USB port number

Returns: Result object

**getSBU2Voltage (channel)**

Get the voltage on the SBU2 line for a port.

**Parameters**

- **channel (int)** – The USB port number

Returns: Result object

**getUpstreamBoostMode ()**

Get the upstream boost mode.

**Returns: Result object**

Result value 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

**getUpstreamMode ()**

Get the upstream switch mode for the USB upstream ports.

Returns: Result object

**getUpstreamState ()**

Get the upstream switch state for the USB upstream ports.

**Returns: Result object**

Result value 2 if no ports plugged in; 0 if port0 is active, 1 if port1 is active.

**setAltModeConfig (channel, configuration)****Sets alt mode configuration for defined USB channel.**

See the product datasheet for device specific details

**Parameters**

- **channel (int)** – The USB channel
- **configuration (uint)** – The configuration to set

**Returns (int):**

Non-zero BrainStem error code on failure.

**setCC1Enable (channel, enable)**

Enable CC1 lines for a Type C USB port

**Parameters**

- **channel (int)** – The USB port number
- **enable (int)** – enable (0 = disable, 1 = enable)

**Returns (int):**

Non-zero BrainStem error code on failure.

**setCC2Enable (channel, enable)**

Enable CC2 lines for a Type C USB port

**Parameters**

- **channel (int)** – The USB port number
- **enable (int)** – enable [0 = disable, 1 = enable]

**Returns (int):**

Non-zero BrainStem error code on failure.

**setCableFlip (channel, enable)**

Enables a cable orientation flip within the S85 switch.

**Parameters**

- **channel1** (*int*) - The USB port number
- **enable** (*int*) - enables cable flip. [0 = disable, 1 = enable]

**Returns (int):**

Non-zero BrainStem error code on failure.

**setConnectMode (channel, mode)**

Set The connection mode for the Switch.

**Parameters**

- **channel1** (*int*) - Upstream port to be applied.
- **mode** (*int*) - 0 = Manual mode, 1 = Auto mode.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataDisable (channel)**

Disable just the data lines for a USB port.

**Parameters**

- **channel1** (*int*) - The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataEnable (channel)**

Enable just the data lines for a USB port.

**Parameters**

- **channel1** (*int*) - The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDownstreamBoostMode (setting)****Set the downstream boost mode.**

Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

**Parameters**

- **setting** (*int*) - Downstream boost setting 0, 1, 2, or 3.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setEnumerationDelay (ms\_delay)****Set the import enumeration delay in milliseconds.**

This setting must be saved with a stem.system.save() call for it to be active. This setting is persistent across hub power down. Resetting the hub will return this setting to the default value of 0ms.

**Parameters**

`ms_delay (int)` – Interport delay in milliseconds

**Returns (int):**

Non-zero BrainStem error code on failure.

**setHiSpeedDataDisable (channel)**

Disable Hi-Speed (USB2.0) data transfer for a USB port.

**Parameters**

`channel (int)` – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setHiSpeedDataEnable (channel)**

Enable Hi-Speed (USB2.0) data transfer for a USB port.

**Parameters**

`channel (int)` – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setHubMode (mode)**

**Set a bit mapped representation of the hub mode.**

Usually represents the port data and power lines enable/disable. See the product datasheet for state mapping.

**Parameters**

`mode (int)` – The hub state

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPortCurrentLimit (channel, microAmps)**

**Set the current limit for the port. If the set limit is not achievable,**

devices will round down to the nearest available current limit setting. This setting can be saved with a `stem.system.save()` call to make it persistent.

**Parameters**

- `channel (int)` – Port index.
- `microAmps (int)` – The current limit setting in microAmps (1A=10e6)

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPortDisable (channel)**

Disable both power and data lines for a USB port.

**Parameters**

`channel (int)` – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPortEnable (channel)**

Enable both power and data lines for a USB port.

**Parameters**

`channel (int)` – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPortMode** (*channel, mode*)

**Set the mode for the Port.**

The mode setting defaults to SDP or Standard Downstream port, and can be set to CDP (charging downstream port) for devices that require high port charge current above 500 millamps.

**Parameters**

- **channel** (*int*) – Port Index.
- **mode** (*int*) – Mode The port mode setting (0 - SDP, 1 - CDP).

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerDisable** (*channel*)

Disable just the power line for a USB port.

**Parameters**

- **channel** (*int*) – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerEnable** (*channel*)

Enable just the power line for a USB port.

**Parameters**

- **channel** (*int*) – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setSBUEnable** (*channel, enable*)

Enable SBU1/SBU2 lines for a Type C USB port based on usbPortMode settings.

**Parameters**

- **channel** (*int*) – The USB port number
- **enable** (*int*) – enables SBU1/SBU2 [0 = disable, 1 = enable]

**Returns (int):**

Non-zero BrainStem error code on failure.

**setSuperSpeedDataDisable** (*channel*)

Disable SuperSpeed (USB3.0) data transfer for a USB port.

**Parameters**

- **channel** (*int*) – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setSuperSpeedDataEnable** (*channel*)

Enable SuperSpeed (USB3.0) data transfer for a USB port.

**Parameters**

- **channel** (*int*) – The USB port number

**Returns (int):**

Non-zero BrainStem error code on failure.

**setUpstreamBoostMode** (*setting*)

**Set the upstream boost mode.**

Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes

are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

**Parameters**

`setting (int)` – Upstream boost setting 0, 1, 2, or 3.

**Returns (int):**

Non-zero BrainStem error code on failure.

`setUpstreamMode (mode)`

Set the upstream switch mode for the USB upstream ports

**Parameters**

`mode (int)` –

- Auto: UPSTREAM\_MODE\_AUTO = 2
- Port 0: UPSTREAM\_STATE\_PORT\_0 = 0
- Port 1: UPSTREAM\_STATE\_PORT\_1 = 1

**Returns (int):**

Non-zero BrainStem error code on failure.

### 3.2.30 USB System

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

The USBSystem class provides high level control of the lower level Port Class.

`getDataRoleBehavior ()`

Gets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

**Returns**

`value (int)`: An enumerated representation of the current behavior. `error`: Non-zero BrainStem error code on failure.

**Return type**

`Result` (object)

`getDataRoleBehaviorConfig ()`

Gets the current data role behavior configuration Certain data role behaviors use a list of ports to determine priority host priority.

**Returns**

`value (tuple(int))`: A list of ports which indicate priority sequencing. `error`: Non-zero BrainStem error code on failure.

**Return type**

`Result` (object)

`getDataRoleList ()`

Gets the data role of all ports with a single call Equivalent to calling Port.getDataRole() on each individual port.

**Returns**

`value (int)`: Bit packed representation of the data role for all ports. `error`: Non-zero BrainStem error code on failure.

**Return type**

`Result` (object)

**getEnabledList()**

Gets the current enabled status of all ports with a single call. Equivalent to calling Port-Class::setEnabled() on each port.

**Returns**

value (int): Bit packed representation of the enabled status for all ports. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getEnumerationDelay()**

Gets the inter-port enumeration delay in milli-seconds (mS). Delay is applied upon hub enumeration.

**Returns**

value (int): Current inter-port delay in milli-seconds (mS). error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getModeList()**

Gets the current mode of all ports with a single call. Equivalent to calling Port-Class::getMode() on each port.

**Returns**

value (tuple(int)): List of modes of each port. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerBehavior()**

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system. i.e. What happens when requested power greater than available power.

**Returns**

value (int): An enumerated representation of the current behavior. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getPowerBehaviorConfig()**

Gets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

**Returns**

value (tuple(int)): A list of ports which indicate priority sequencing. error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getSelectorMode()**

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

**Returns**

value (int): The current mode error: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getStateList ()**

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port.

**Returns**

*value* (*tuple(int)*): List of states for each port. *error*: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**getUpstream ()**

Gets the current upstream port.

**Returns**

*value*: Index of upstream port. *error*: Non-zero BrainStem error code on failure.

**Return type**

*Result* (object)

**resetEntityToFactoryDefaults ()**

Resets the USBSystemClass Entity to it factory default configuration.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataRoleBehavior (*behavior*)**

Sets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

**Parameters**

*behavior* (*int*) – An enumerated representation of the behavior to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setDataRoleBehaviorConfig (*config*)**

Sets the current data role behavior configuration Certain data role behaviors use a list of ports to determine host priority.

**Parameters**

*config* (*tuple(int)*) – List of ports which indicate priority sequencing.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setEnabledList (*enabledList*)**

Sets the enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

**Parameters**

*enabledList* (*int*) – Bit packed representation of the enabled status for all ports to be applied.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setEnumerationDelay (*delay*)**

Set the inter-port enumeration delay in milliseconds. This setting should be saved with a stem.system.save() call. Delay is applied upon hub enumeration.

**Parameters**

*delay* (*int*) – Delay in milli-seconds (mS) to be applied between port enables.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setModeList** (*modeList*)

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port

**Parameters**

**modeList** (*tuple(int)*) – List of modes to be set for each port.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerBehavior** (*behavior*)

Sets the behavior of how available power is managed. i.e. What happens when requested power is greater than available power.

**Parameters**

**behavior** (*int*) – An enumerated representation of the behavior to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setPowerBehaviorConfig** (*config*)

Sets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

**config** (*tuple(int)*) – List of ports which indicate priority sequencing.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setSelectorMode** (*mode*)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

**Parameters**

**mode** (*mode*) – Mode to be set.

**Returns (int):**

Non-zero BrainStem error code on failure.

**setUpstream** (*port*)

Sets the upstream port.

**Parameters**

**port** (*int*) – Upstream port to be applied.

**Returns (int):**

Non-zero BrainStem error code on failure.

### 3.2.31 Version

Provides version access utilities.

**brainstem.version.get\_version\_string** (*packed\_version=None*)

Returns the library version as a string

**Parameters**

**packed\_version** (*int (Optional)*) – If version is provided, it is unpacked and presented as the version string. Most useful for printing the firmware version currently installed on a module.

`brainstem.version.unpack_version(packed_version)`

Returns the library version as a 3-tuple (major, minor, patch)

**Parameters**

`packed_version (int (Optional))` – The packed version number.

## 3.3 C++ API Reference

Welcome to the BrainStem C++ API reference documentation. This documentation covers the c++ Acroname BrainStem library. This reference assumes that you understand the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation.

- *BrainStem Overview*
- *BrainStem Terminology*
- *Getting Started with the BrainStem.*

The Getting started guide is particularly useful for learning how to use the application tools we provide to communicate with your hardware.

---

### 3.3.1 Errors

#### group aErrors

Unified list of Error codes for BrainStem module Iteration.

aError.h provides a unified list of error codes. These error codes apply accross all API's. Library functions will return one of these error codes when appropriate.

### 3.3.2 Classes

#### class AnalogClass : public Acroname::BrainStem::EntityClass

MARK: Analog Class.

*AnalogClass*. Interface to analog entities on BrainStem modules. Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

#### class AppClass : public Acroname::BrainStem::EntityClass

MARK: App Class.

*AppClass*. Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

#### class ClockClass : public Acroname::BrainStem::EntityClass

MARK: Clock Class.

*ClockClass*. Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

---

**Note:** Clock time must be reset if power to the BrainStem module is lost.

---

---

```
class DigitalClass : public Acroname::BrainStem::EntityClass
```

MARK: Digital Class.

*DigitalClass*. Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCservo Input, RCservo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

```
class EntityClass
```

Subclassed by *Acroname::BrainStem::AnalogClass*, *Acroname::BrainStem::AppClass*,  
*Acroname::BrainStem::ClockClass*, *Acroname::BrainStem::DigitalClass*, *Acroname::BrainStem::EqualizerClass*, *Acroname::BrainStem::I2CClass*, *Acroname::BrainStem::MuxClass*,  
*Acroname::BrainStem::PointerClass*, *Acroname::BrainStem::PortClass*, *Acroname::BrainStem::PowerDeliveryClass*, *Acroname::BrainStem::RailClass*,  
*Acroname::BrainStem::RCServoClass*, *Acroname::BrainStem::RelayClass*, *Acroname::BrainStem::SignalClass*, *Acroname::BrainStem::StoreClass*,  
*Acroname::BrainStem::SystemClass*, *Acroname::BrainStem::TemperatureClass*, *Acroname::BrainStem::TimerClass*, *Acroname::BrainStem::UARTClass*, *Acroname::BrainStem::USBClass*,  
*Acroname::BrainStem::USBSystemClass*

```
class EqualizerClass : public Acroname::BrainStem::EntityClass
```

MARK: Equalizer Class.

*EqualizerClass*. Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

```
class I2CClass : public Acroname::BrainStem::EntityClass
```

MARK: I2C Class.

*I2CClass*. Interface the I2C buses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entities bus.

```
class Link
```

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.

```
class Module
```

The *Module* class provides a generic interface to a BrainStem hardware module. The *Module* class is the parent class for all BrainStem modules. Each module inherits from *Module* and implements its hardware specific features.

Subclassed by *a40PinModule*, *aMTMDAQ1*, *aMTMDAQ2*, *aMTMIOSerial*, *aMTMLoad1*, *aMTMPM1*, *aMTMRelay*, *aMTMStemModule*, *aUSBCSwitch*, *aUSBHub2x4*, *aUSBHub3c*, *aUSBHub3p*

```
class MuxClass : public Acroname::BrainStem::EntityClass
```

MARK: Mux Class.

*MuxClass*. A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to on or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

class **PointerClass** : public Acroname::BrainStem::*EntityClass*

MARK: Pointer Class.

*PointerClass*. Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

class **PortClass** : public Acroname::BrainStem::*EntityClass*

MARK: Port Class.

Port Class The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

class **PowerDeliveryClass** : public Acroname::BrainStem::*EntityClass*

MARK: Power Delivery Class.

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.

class **RailClass** : public Acroname::BrainStem::*EntityClass*

MARK: Rail Class.

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

class **RCServoClass** : public Acroname::BrainStem::*EntityClass*

MARK: RCServo Class.

*RCServoClass*. Interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

class **RelayClass** : public Acroname::BrainStem::*EntityClass*

MARK: Relay Class.

*RelayClass*. Interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

class **SignalClass** : public Acroname::BrainStem::*EntityClass*

MARK: Signal Class.

*SignalClass*. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

class **StoreClass** : public Acroname::BrainStem::*EntityClass*

MARK: Store Class.

*StoreClass*. The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

class **SystemClass** : public Acroname::BrainStem::*EntityClass*

MARK: System Class.

*SystemClass*. The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc. The most common brainstem example uses the system entity to blink the User LED.

class **TemperatureClass** : public Acroname::BrainStem::*EntityClass*

MARK: Temperature Class.

*TemperatureClass*. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

class **TimerClass** : public Acroname::BrainStem::*EntityClass*

MARK: Timer Class.

*TimerClass*. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

class **UARTClass** : public Acroname::BrainStem::*EntityClass*

MARK: UART Class.

UART Class. A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines.

class **USBCClass** : public Acroname::BrainStem::*EntityClass*

MARK: USB Class.

*USBCClass*. The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

class **USBSystemClass** : public Acroname::BrainStem::*EntityClass*

MARK: USB System Class.

USBSystem Class The USBSystem class provides high level control of the lower level Port Class.

Subclassed by *aUSBHub3c::HubClass*

### 3.3.3 Acroname Modules

#### Hubs & Switches

*USBHub3c*

*USBHub3p*

*USBHub2x4*

*USBCSwitch*

#### MTM Modules

*MTM-DAQ-2*

*MTM-EtherStem*

*MTM-IO-Serial*

*MTM-Load-1*

*MTM-PM-1*

*MTM-Relay*

*MTM-USBStem*

#### Abstract Classes

*MTM-Stem Module*

#### USBHub3c

##### Class

class **aUSBHub3c** : public Acroname::BrainStem::*Module*

Concrete Module implementation of a USBHub3c Allows a user to connect to and control an attached hub.

## Public Types

```
enum PORT_ID
    Port ID
    Values:
        enumerator kPORT_ID_0
        enumerator kPORT_ID_1
        enumerator kPORT_ID_2
        enumerator kPORT_ID_3
        enumerator kPORT_ID_4
        enumerator kPORT_ID_5
        enumerator kPORT_ID_CONTROL
        enumerator kPORT_ID_POWER_C

typedef enum aUSBHub3c::PORT_ID PORT_ID_t
    Port ID
```

## Public Members

```
HubClass hub
    Hub Class

Acroname::BrainStem::AppClass app[aUSBHUB3C_NUM_APPS]
    App Class

Acroname::BrainStem::PointerClass pointer[aUSBHUB3C_NUM_POINTERS]
    Pointer Class

Acroname::BrainStem::PowerDeliveryClass pd[aUSBHUB3C_NUM_USB_PORTS]
    Power Delivery Class

Acroname::BrainStem::RailClass rail[aUSBHUB3C_NUM_RAILS]
    Rail Class
```

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB3C\_NUM\_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TemperatureClass* **temperature**[aUSBHUB3C\_NUM\_TEMPERATURES]

Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBHUB3C\_NUM\_TIMERS]

Timer Class

Acroname::BrainStem::*I2CClass* **i2c**[aUSBHUB3C\_NUM\_I2C]

I2C Class

Acroname::BrainStem::*USBClass* **usb**

USB Class

Acroname::BrainStem::*UARTClass* **uart**[aUSBHUB3C\_NUM\_UART]

UART Class

class **HubClass** : public Acroname::BrainStem::*USBSystemClass*

Hub class implementation for use with USBHub3c.

## Defines

**aUSBHUB3C\_MODULE** 6

USBHub3c module number

**aUSBHUB3C\_NUM\_APPS** 4

Number of App instances available

**aUSBHUB3C\_NUM\_POINTERS** 4

Number of Pointer instances available

**aUSBHUB3C\_NUM\_STORES** 2

Number of Store instances available

**aUSBHUB3C\_NUM\_INTERNAL\_SLOTS** 12

Store: Number of internal slots instances available

**aUSBHUB3C\_NUM\_RAM\_SLOTS** 1

Store: Number of RAM slot instances available

**aUSBHUB3C\_STORE\_INTERNAL\_INDEX 0**

Store: Array index for internal store

**aUSBHUB3C\_STORE\_RAM\_INDEX 1**

Store: Array index for RAM store

**aUSBHUB3C\_STORE\_EEPROM\_INDEX 2**

Store: Array index for EEPROM store

**aUSBHUB3C\_NUM\_TEMPERATURES 3**

Number of Temperature instances available

**aUSBHUB3C\_NUM\_TIMERS 8**

Number of Timer instances available

**aUSBHUB3C\_NUM\_USB 1**

Number of USB instances available

**aUSBHUB3C\_NUM\_USB\_PORTS 8**

Number of USB ports available

**aUSBHUB3C\_NUM\_PD\_PORTS 8**

Number of PD compatible ports available

**aUSBHUB3C\_NUM\_PD\_RULES\_PER\_PORT 7**

Number of PD Rules per port available

**aUSBHUB3C\_NUM\_RAILS 7**

Number of Rail instances available

**aUSBHUB3C\_NUM\_I2C 1**

Number of I2C instances available

**aUSBHUB3C\_NUM\_UART 1**

Number of UART instances available

## USBHub3p

### Class

class **aUSBHub3p** : public Acroname::BrainStem::*Module*

Concrete Module implementation of a *aUSBHub3p* Allows a user to connect to and control an attached hub.

## Public Members

Acroname::BrainStem::*AppClass* **app**[aUSBHUB3P\_NUM\_APPS]

App Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBHUB3P\_NUM\_POINTERS]

Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB3P\_NUM\_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TemperatureClass* **temperature**

Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBHUB3P\_NUM\_TIMERS]

Timer Class

Acroname::BrainStem::*USBClass* **usb**

USB Class

## Defines

**aUSBHUB3P\_MODULE** 6

USBHub3p module number

**aUSBHUB3P\_NUM\_APPS** 4

Number of App instances available

**aUSBHUB3P\_NUM\_POINTERS** 4

Number of Pointer instances available

**aUSBHUB3P\_NUM\_STORES** 2

Number of Store instances available

**aUSBHUB3P\_NUM\_INTERNAL\_SLOTS** 12

Store: Number of internal slots instances available

**aUSBHUB3P\_NUM\_RAM\_SLOTS** 1

Store: Number of RAM slot instances available

**aUSBHUB3P\_NUM\_TIMERS** 8

Number of Timer instances available

**aUSBHUB3P\_NUM\_USB** 1

Number of USB instances available

**aUSBHUB3P\_NUM\_USB\_PORTS** 8

Number of USB ports available

## Port State Defines

**aUSBHUB3P\_USB\_VBUS\_ENABLED** 0

USB VBUS current state

**aUSBHUB3P\_USB2\_DATA\_ENABLED** 1

USB2 data current state

**aUSBHUB3P\_USB3\_DATA\_ENABLED** 3

USB3 data current state

**aUSBHUB3P\_USB\_SPEED\_USB2** 11

USB2 speed current state

**aUSBHUB3P\_USB\_SPEED\_USB3** 12

USB3 speed current state

**aUSBHUB3P\_USB\_ERROR\_FLAG** 19

Error indicator for this port

(see ‘Port Errors’ below)

**aUSBHUB3P\_USB2\_BOOST\_ENABLED** 20

USB2 boost current state

**aUSBHUB3P\_DEVICE\_ATTACHED** 23

Device attached indicator for this port

## Port State Error Defines

**aUSBHUB3P\_ERROR\_VBUS\_OVERCURRENT** 0

VBUS overcurrent error

**aUSBHUB3P\_ERROR\_VBUS\_BACKDRIVE** 1

VBUS backdrive (backpower) error

**aUSBHUB3P\_ERROR\_HUB\_POWER** 2

Hub power error

**aUSBHUB3P\_ERROR\_OVER\_TEMPERATURE** 3

Over temperature error

## USBHub2x4

### Class

class **aUSBHub2x4** : public Acroname::BrainStem:*Module*

Concrete Module implementation of a USBHub2x4 Allows a user to connect to and control an attached hub.

### Public Members

Acroname::BrainStem::*AppClass* **app**[aUSBHUB2X4\_NUM\_APPS]

App Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBHUB2X4\_NUM\_POINTERS]

Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBHUB2X4\_NUM\_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TemperatureClass* **temperature**

Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBHUB2X4\_NUM\_TIMERS]

Timer Class

Acroname::BrainStem::*USBClass* **usb**

USB Class

## Defines

**aUSBHUB2X4\_MODULE** 6

USBHub2x4 module number

**aUSBHUB2X4\_NUM\_APPS** 4

Number of App instances available

**aUSBHUB2X4\_NUM\_POINTERS** 4

Number of Pointer instances available

**aUSBHUB2X4\_NUM\_STORES** 2

Number of Store instances available

**aUSBHUB2X4\_NUM\_INTERNAL\_SLOTS** 12

Store: Number of internal slots instances available

**aUSBHUB2X4\_NUM\_RAM\_SLOTS** 1

Store: Number of RAM slot instances available

**aUSBHUB2X4\_NUM\_TIMERS** 8

Number of Timer instances available

**aUSBHUB2X4\_NUM\_USB** 1

Number of USB instances available

**aUSBHUB2X4\_NUM\_USB\_PORTS** 4

Number of USB ports available

## Port State Defines

**aUSBHUB2X4\_USB\_VBUS\_ENABLED** 0

USB VBUS current state

**aUSBHUB2X4\_USB2\_DATA\_ENABLED** 1

USB2 data current state

**aUSBHUB2X4\_USB\_ERROR\_FLAG** 19

Error indicator for this port

(see ‘Port Errors’ below)

**aUSBHUB2X4\_USB2\_BOOST\_ENABLED** 20

USB2 boost current state

**aUSBHUB2X4\_DEVICE\_ATTACHED** 23

Device attached indicator for this port

**aUSBHUB2X4\_CONSTANT\_CURRENT** 24

Constant current mode indicator

## Port State Error Defines

**aUSBHUB2X4\_ERROR\_VBUS\_OVERCURRENT** 0

VBUS overcurrent error

**aUSBHUB2X4\_ERROR\_OVER\_TEMPERATURE** 3

Over temperature error

**aUSBHub2X4\_ERROR\_DISCHARGE** 4

Discharge error

## USBCSwitch

### Class

class **aUSBCSwitch** : public Acroname::BrainStem::*Module*

Concrete Module implementation of a USBCSwitch Allows a user to connect to and control an attached switch.

### Public Types

enum **EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS**

Equalizer 3P0 transmitter configs

*Values:*

enumerator **MUX\_1db\_COM\_0db\_900mV**

enumerator **MUX\_0db\_COM\_1db\_900mV**

enumerator **MUX\_1db\_COM\_1db\_900mV**

enumerator **MUX\_0db\_COM\_0db\_900mV**

enumerator **MUX\_0db\_COM\_0db\_1100mV**

```
enumerator MUX_1db_COM_0db_1100mV  
  
enumerator MUX_0db_COM_1db_1100mV  
  
enumerator MUX_2db_COM_2db_1100mV  
  
enumerator MUX_0db_COM_0db_1300mV  
  
enum EQUALIZER_3P0_RECEIVER_CONFIGS  
    Equalizer 3P0 receiver configs  
    Values:  
    enumerator LEVEL_1_3P0  
  
    enumerator LEVEL_2_3P0  
  
    enumerator LEVEL_3_3P0  
  
    enumerator LEVEL_4_3P0  
  
    enumerator LEVEL_5_3P0  
  
    enumerator LEVEL_6_3P0  
  
    enumerator LEVEL_7_3P0  
  
    enumerator LEVEL_8_3P0  
  
    enumerator LEVEL_9_3P0  
  
    enumerator LEVEL_10_3P0  
  
    enumerator LEVEL_11_3P0  
  
    enumerator LEVEL_12_3P0  
  
    enumerator LEVEL_13_3P0  
  
    enumerator LEVEL_14_3P0  
  
    enumerator LEVEL_15_3P0
```

```
enumerator LEVEL_16_3P0

enum EQUALIZER_2P0_TRANSMITTER_CONFIGS
    Equalizer 2P0 transmitter configs
    Values:
        enumerator TRANSMITTER_2P0_40mV
        enumerator TRANSMITTER_2P0_60mV
        enumerator TRANSMITTER_2P0_80mV
        enumerator TRANSMITTER_2P0_0mV

enum EQUALIZER_2P0_RECEIVER_CONFIGS
    Equalizer 3P0 receiver configs
    Values:
        enumerator LEVEL_1_2P0
        enumerator LEVEL_2_2P0

enum EQUALIZER_CHANNELS
    Equalizer channels
    Values:
        enumerator BOTH
        enumerator MUX
        enumerator COMMON

enum daughtercard_type
    Daughter Cards
    Values:
        enumerator NO_DAUGHTERCARD
        enumerator PASSIVE_DAUGHTERCARD
        enumerator REDRIVER_DAUGHTERCARD
        enumerator UNKNOWN_DAUGHTERCARD
```

## Public Members

Acroname::BrainStem::*AppClass* **app**[aUSBCSWITCH\_NUM\_APPS]  
App Class

Acroname::BrainStem::*MuxClass* **mux**  
Mux Class

Acroname::BrainStem::*PointerClass* **pointer**[aUSBCSWITCH\_NUM\_POINTERS]  
Pointer Class

Acroname::BrainStem::*StoreClass* **store**[aUSBCSWITCH\_NUM\_STORES]  
Store Class

Acroname::BrainStem::*SystemClass* **system**  
System Class

Acroname::BrainStem::*TimerClass* **timer**[aUSBCSWITCH\_NUM\_TIMERS]  
Timer Class

Acroname::BrainStem::*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 Members

Acroname::BrainStem::*AppClass* **app**[aMTMIOSERIAL\_NUM\_APPS]

App Class

Acroname::BrainStem::*DigitalClass* **digital**[aMTMIOSERIAL\_NUM\_DIGITALS]

Digital Class

Acroname::BrainStem::*I2CClass* **i2c**[aMTMIOSERIAL\_NUM\_I2C]

I2C Class

Acroname::BrainStem::*UARTClass* **uart**[aMTMIOSERIAL\_NUM\_UART]

UART Class

Acroname::BrainStem::*PointerClass* **pointer**[aMTMIOSERIAL\_NUM\_POINTERS]

Pointer Class

Acroname::BrainStem::*RailClass* **rail**[aMTMIOSERIAL\_NUM\_RAILS]

Rail Class

Acroname::BrainStem::*RCServoClass* **servo**[aMTM\_STEM\_NUM\_SERVOS]

RC Servo Class

Acroname::BrainStem::*SignalClass* **signal**[aMTMIOSERIAL\_NUM\_SIGNALS]

Signal Class

Acroname::BrainStem::*StoreClass* **store**[aMTMIOSERIAL\_NUM\_STORES]

Store Class

Acroname::BrainStem::*SystemClass* **system**

System Class

Acroname::BrainStem::*TemperatureClass* **temperature**

Temperature Class

Acroname::BrainStem::*TimerClass* **timer**[aMTMIOSERIAL\_NUM\_TIMERS]

Timer Class

Acroname::BrainStem::*USBClass* **usb**

USB Class

## 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\_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

### 3.3.4 Analog Class

**class AnalogClass** : public Acroname::BrainStem::*EntityClass*

MARK: Analog Class.

*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 [get-BulkCaptureState\(\)](#) will return either bulkCaptureFinished or bulkCaptureError.

**aErr** **getBulkCaptureState** (uint8\_t \*state)

Get the current bulk capture state for this analog.

#### Parameters

**state** – the state of bulk capture.

- Idle: bulkCaptureIdle = 0
- Pending: bulkCapturePending = 1
- Finished: bulkCaptureFinished = 2
- Error: bulkCaptureError = 3

#### Returns

Returns common entity return values

### 3.3.5 App Class

```
class AppClass : public Acroname::BrainStem::EntityClass
```

MARK: App Class.

[AppClass](#). Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

#### Public Functions

**AppClass** (void)

Constructor.

**~AppClass** (void)

Destructor.

**void init** ([Module](#) \*pModule, const uint8\_t index)

Initialize the class.

#### Parameters

- **pModule** – The module.
- **index** – The cmdAPP reflex index to be addressed.

**aErr execute** (const uint32\_t appParam)

Execute the app reflex on the module. Don't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

#### Parameters

**appParam** – The app parameter handed to the reflex.

#### Returns

[aErrNone](#) - success.

#### Returns

[aErrTimeout](#) - The request timed out waiting to start execution.

#### Returns

[aErrConnection](#) - No active link connection.

#### Returns

[aErrNotFound](#) - the app reflex was not found or not enabled on the module.

```
aErr execute (const uint32_t appParam, uint32_t *returnVal, const uint32_t msTimeout = 1000)
```

Execute the app reflex on the module. Wait for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

**Parameters**

- **appParam** – The app parameter handed to the reflex.
- **returnVal** – The return value filled in from the result of executing the reflex routine.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

**Returns**

*aErrNone* - success.

**Returns**

*aErrTimeout* - The request timed out waiting for a response.

**Returns**

*aErrConnection* - No active link connection.

**Returns**

*aErrNotFound* - the app reflex was not found or not enabled on the module.

### 3.3.6 Clock Class

```
class ClockClass : public Acroname::BrainStem::EntityClass
```

MARK: Clock Class.

*ClockClass*. Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

---

**Note:** Clock time must be reset if power to the BrainStem module is lost.

---

#### Public Functions

```
ClockClass (void)
```

Constructor.

```
virtual ~ClockClass (void)
```

Destructor.

```
void init (Module *pModule, const uint8_t index)
```

Initialize the class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – The index of the clock entity being initialized.

```
aErr getYear (uint16_t *year)
```

Get the four digit year value (0-4095).

**Parameters**

**year** – Get the year portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr setYear (const uint16_t year)`

Set the four digit year value (0-4095).

**Parameters**

`year` – Set the year portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr getMonth (uint8_t *month)`

Get the two digit month value (1-12).

**Parameters**

`month` – The two digit month portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr setMonth (const uint8_t month)`

Set the two digit month value (1-12).

**Parameters**

`month` – The two digit month portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr getDay (uint8_t *day)`

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

**Parameters**

`day` – The two digit day portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr setDay (const uint8_t day)`

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

**Parameters**

`day` – The two digit day portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr getHour (uint8_t *hour)`

Get the two digit hour value (0-23).

**Parameters**

`hour` – The two digit hour portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr setHour (const uint8_t hour)`

Set the two digit hour value (0-23).

**Parameters**

`hour` – The two digit hour portion of the real-time clock value.

**Returns**

Returns common entity return values

`aErr getMinute (uint8_t *min)`

Get the two digit minute value (0-59).

**Parameters**

`min` – The two digit minute portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **setMinute** (const uint8\_t min)

Set the two digit minute value (0-59).

**Parameters**

**min** – The two digit minute portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **getSecond** (uint8\_t \*sec)

Get the two digit second value (0-59).

**Parameters**

**sec** – The two digit second portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **setSecond** (const uint8\_t sec)

Set the two digit second value (0-59).

**Parameters**

**sec** – The two digit second portion of the real-time clock value.

**Returns**

Returns common entity return values

### 3.3.7 Digital Class

class **DigitalClass** : public Acroname::BrainStem::*EntityClass*

MARK: Digital Class.

*DigitalClass*. Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

#### Public Functions

**DigitalClass** (void)

Constructor.

virtual ~**DigitalClass** (void)

Destructor.

void **init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – The index of the digital entity being initialized.

*aErr* **setConfiguration** (const uint8\_t configuration)

Set the digital configuration to one of the available 5 states. Note: Some configurations are only supported on specific pins.

**Parameters**

**configuration** –

- Digital Input: digitalConfigurationInput = 0
- Digital Output: digitalConfigurationOutput = 1
- RCServo Input: digitalConfigurationRCServoInput = 2

- RCServo Output: digitalConfigurationRCServoOutput = 3
- High Z State: digitalConfigurationHiZ = 4
- Digital Input: digitalConfigurationInputPullUp = 0
- Digital Input: digitalConfigurationInputNoPull = 4
- Digital Input: digitalConfigurationInputPullDown = 5

**Returns**

Returns common entity return values

**Returns**

*aErrConfiguration* - Entity does not support this configuration.

***aErr getConfiguration* (uint8\_t \*configuration)**

Get the digital configuration.

**Parameters**

**configuration** -- Current configuration of the digital entity.

**Returns**

Returns common entity return values

***aErr setState* (const uint8\_t state)**

Set the logical state.

**Parameters**

**state** - The state to be set. 0 is logic low, 1 is logic high.

**Returns**

Returns common entity return values

***aErr getState* (uint8\_t \*state)**

Get the state.

**Parameters**

**state** - The current state of the digital entity. 0 is logic low, 1 is logic high.

Note: If in high Z state an error will be returned.

**Returns**

Returns common entity return values

***aErr setStateAll* (const uint32\_t state)**

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

**Parameters**

**state** - The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc.

**Returns**

Returns common entity return values

***aErr getStateAll* (uint32\_t \*state)**

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

**Parameters**

**state** - The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.

**Returns**

Returns common entity return values

### 3.3.8 Entity Class

```
class EntityClass
    Subclassed by Acroname::BrainStem::AnalogClass, Acroname::BrainStem::AppClass,
    Acroname::BrainStem::ClockClass, Acroname::BrainStem::DigitalClass, Acron-
    ame::BrainStem::EqualizerClass, Acroname::BrainStem::I2CClass, Acron-
    ame::BrainStem::MuxClass, Acroname::BrainStem::PointerClass, Acron-
    ame::BrainStem::PortClass, Acroname::BrainStem::PowerDeliveryClass, Acron-
    ame::BrainStem::RailClass, Acroname::BrainStem::RCServoClass, Acron-
    ame::BrainStem::RelayClass, Acroname::BrainStem::SignalClass, Acron-
    ame::BrainStem::StoreClass, Acroname::BrainStem::SystemClass, Acron-
    ame::BrainStem::TemperatureClass, Acroname::BrainStem::TimerClass, Acron-
    ame::BrainStem::UARTClass, Acroname::BrainStem::USBClass, Acron-
    ame::BrainStem::USBSystemClass
```

#### Public Functions

**EntityClass** (void)

Constructor.

virtual ~**EntityClass** (void)

Destructor.

void **init** (*Module* \*pModule, const uint8\_t command, const uint8\_t index)

init.

Initialize the entity class.

**Parameters**

- **pModule** – The BrainStem module object.
- **command** – The command of the UEI.
- **index** – The index of the UEI entity.

*aErr* **callUEI** (const uint8\_t option)

A callUEI is a setUEI that has no data length.

**Parameters**

- **option** – An option for the UEI.

**Returns**

Returns common entity return values

*aErr* **setUEI8** (const uint8\_t option, const uint8\_t byteValue)

Set a byte value.

**Parameters**

- **option** – The option for the UEI.
- **byteValue** – The value.

**Returns**

Returns common entity return values

*aErr* **setUEI8** (const uint8\_t option, const uint8\_t param, const uint8\_t byteValue)

Set a byte value with a subindex.

**Parameters**

- **option** – The option for the UEI.
- **param** – of the option.
- **byteValue** – The value.

**Returns**

Returns common entity return values

*aErr* **getUEI8** (const uint8\_t option, uint8\_t \*byteValue)

Get a byte value.

**Parameters**

- **option** – The option for the UEI.
- **byteValue** – The value.

**Returns**

Returns common entity return values

*aErr* **getUEI8** (const uint8\_t option, const uint8\_t param, uint8\_t \*byteValue)

Get a byte value with a parameter.

**Parameters**

- **option** – The option for the UEI.
- **param** – The parameter.
- **byteValue** – The value.

**Returns**

Returns common entity return values

*aErr* **setUEI16** (const uint8\_t option, const uint16\_t shortValue)

Set a 2-byte value.

**Parameters**

- **option** – The option for the UEI.
- **shortValue** – The value.

**Returns**

Returns common entity return values

*aErr* **getUEI16** (const uint8\_t option, uint16\_t \*shortValue)

Get a 2-byte value.

**Parameters**

- **option** – The option for the UEI.
- **shortValue** – The value.

**Returns**

Returns common entity return values

*aErr* **getUEI16** (const uint8\_t option, const uint8\_t param, uint16\_t \*shortValue)

Get a 2-byte value with a parameter.

**Parameters**

- **option** – The option for the UEI.
- **param** – The parameter.
- **shortValue** – The value.

**Returns**

Returns common entity return values

*aErr* **setUEI32** (const uint8\_t option, const uint32\_t intValue)

Set a 4-byte value.

**Parameters**

- **option** – The option for the UEI.
- **intValue** – The value.

**Returns**

Returns common entity return values

*aErr* **setUEI32** (const uint8\_t option, const uint8\_t subIndex, const uint32\_t intValue)

Set a 4-byte value, with a subindex parameter.

**Parameters**

- **option** – The option for the UEI.
- **subIndex** – The subindex to set.
- **intValue** – The value.

**Returns**

Returns common entity return values

*aErr* **getUEI32** (const uint8\_t option, uint32\_t \*intValue)

Get a 4-byte value.

**Parameters**

- **option** – The option for the UEI.
- **intValue** – The 4 byte value

**Returns**

Returns common entity return values

*aErr* **getUEI32** (const uint8\_t option, const uint8\_t param, uint32\_t \*intValue)

Get a 4-byte value with parameter.

**Parameters**

- **option** – The option for the UEI.
- **param** – The parameter.
- **intValue** – The 4 byte value

**Returns**

Returns common entity return values

*aErr* **getUEIBytes8** (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* **getUEIBytes32** (const uint8\_t option, uint32\_t \*buf, const size\_t bufLength, size\_t \*unloadedLength)

Unloads UEI Bytes data as 4-byte values

**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 4-byte values)

**Returns**

Returns common entity return values

uint8\_t **getIndex** (void) const

Get the UEI entity index.

**Returns**

The 1 byte index of the UEI entity.

*aErr* **drainUEI** (const uint8\_t option)

Drain all packets matching this UEI from the packet fifo.

This functionality is useful in rare cases where packet synchronization is lost and a valid return packet is not accessible.

`aErr setStreamEnabled (uint8_t enabled)`

Enables streaming for all possible option codes within the cmd and index the entity was created for.

**Parameters**

`enabled` – The state to be applied. 0 = Disabled; 1 = enabled

**Returns**

Returns common entity return values

`aErr registerOptionCallback (const uint8_t option, const bool enable, Link::streamCallback\_t cb, void *pRef)`

Registers a callback function based on a specific option code. Option code applies to the cmd and index of the called API.

**Parameters**

- `option` – option to filter by (supports Wildcards)
- `enable` – True - installs/updates callback and ref; False - uninstalls callback
- `cb` – Callback to be executed when a new packet matching the criteria is received.
- `pRef` – Pointer to user reference for use inside the callback function.

**Returns**

`aErrNotFound` - Item not found (uninstalling only)

**Returns**

`aErrNone` - success

`aErr getStreamStatus (std::map<uint64_t, uint32_t> *status)`

Gets all available stream values associated with the cmd and index of the called API. Keys can be decoded with [Link::getStreamKeyElement](#).

**Parameters**

`status` – map of option value pairs to be filled. Option codes are based on the cmd and index of the calling API.

**Returns**

`aErrParam` if status is null

**Returns**

`aErrResource` - if the link is not valid

**Returns**

`aErrNone` - success

## Public Static Functions

`static uint8_t getUEIBytesSequence (const uint8_t sequence)`

**Parameters**

`sequence` -- Sequence byte to be checked.

**Returns**

The sequence number of the byte.

`static bool getUEIBytesContinue (const uint8_t sequence)`

**Parameters**

`sequence` -- Sequence byte to be checked.

**Returns**

True - Continue bit is set (more packets to come); False - Continue bit is not set (first or last packet).

`static uint8_t sUEIBytesFilter (const aPacket *packet, const void *ref)`

Filter function for UEI Bytes calls. Exposed for unit-testing purposes only.

**Parameters**

- **packet** – UEI packet to be checked/filtered.
- **ref** – Opaque reference handle

### 3.3.9 Equalizer Class

```
class EqualizerClass : public Acroname::BrainStem::EntityClass
```

MARK: Equalizer Class.

*EqualizerClass*. Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

#### Public Functions

**EqualizerClass** (void)

Constructor.

**~EqualizerClass** (void)

Destructor.

**void init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

##### Parameters

- **pModule** – The module.
- **index** – The index.

*aErr setReceiverConfig* (const uint8\_t channel, const uint8\_t config)

Sets the receiver configuration for a given channel.

##### Parameters

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

##### Returns

Returns common entity return values.

*aErr getReceiverConfig* (const uint8\_t channel, uint8\_t \*config)

Gets the receiver configuration for a given channel.

##### Parameters

- **channel** – The equalizer receiver channel.
- **config** – Configuration of the receiver.

##### Returns

Returns common entity return values.

*aErr setTransmitterConfig* (const uint8\_t config)

Sets the transmitter configuration

##### Parameters

- **config** – Configuration to be applied to the transmitter.

##### Returns

Returns common entity return values.

*aErr getTransmitterConfig* (uint8\_t \*config)

Gets the transmitter configuration

##### Parameters

- **config** – Configuration of the Transmitter.

**Returns**

Returns common entity return values.

### 3.3.10 I2C Class

```
class I2CCClass : public Acroname::BrainStem::EntityClass
```

MARK: I2C Class.

*I2CCClass*. 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

**I2CCClass** (void)

Constructor.

**virtual ~I2CCClass** (void)

Destructor.

**void init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – The index of the digital entity being initialized.

*aErr* **read** (const uint8\_t address, const uint8\_t length, uint8\_t \*result)

Read from a device on this I2C bus.

**Parameters**

- **address** -- The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** -- The length of the data to read in bytes.
- **result** -- The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAIN-STEM\_MAXPACKETBYTES - 5

**Returns**

Returns common entity return values

*aErr* **write** (const uint8\_t address, const uint8\_t length, const uint8\_t \*data)

Write to a device on this I2C bus.

**Parameters**

- **address** -- The I2C address (7bit <XXXX-XXX0>) of the device to write.
- **length** -- The length of the data to write in bytes.
- **data** -- The data to send to the device, This array should be no larger than aBRAINSTEM\_MAXPACKETBYTES - 5

**Returns**

Returns common entity return values

*aErr* **setPullup** (const bool bEnable)

Set bus pull-up state. This call only works with stems that have software controlled pull-ups. Check the datasheet for more information. This parameter is saved when system.save is called.

**Parameters**

- **bEnable** -- true enables pull-ups false disables them.

**Returns**

Returns common entity return values

*aErr* **setSpeed** (const uint8\_t speed)

Set I2C bus speed.

This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz

**Parameters**

**speed** -- The speed setting value.

**Returns**

Returns common entity return values

*aErr* **getSpeed** (uint8\_t \*speed)

Get I2C bus speed.

This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz

**Parameters**

**speed** -- The speed setting value.

**Returns**

Returns common entity return values

### 3.3.11 Link Class

class **Link**

The **Link** class provides an interface to a BrainStem link. The link is used to create interfaces to modules on a BrainStem network. The link represents a connection to the BrainStem network from a host computer. The link is always associated with a transport (e.g.: USB, Ethernet, etc.) and a link module, but there are several ways to make this association.

- a. The link can be fully specified with a transport and module serial number
- b. The link can be created by searching a transport and connecting to the first module found.

Calling connect on a link will start a connection to the module based on The link specification. Calling disconnect will disconnect the link from the the current connection.

#### Public Types

enum **STREAM\_PACKET**

Enumeration of stream packet types.

*Values:*

enumerator **kSTREAM\_PACKET\_UNKNOWN**

enumerator **kSTREAM\_PACKET\_U8**

enumerator **kSTREAM\_PACKET\_U16**

```

enumerator kSTREAM_PACKET_U32

enumerator kSTREAM_PACKET_BYTES

enumerator kSTREAM_PACKET_SUBINDEX_U8

enumerator kSTREAM_PACKET_SUBINDEX_U16

enumerator kSTREAM_PACKET_SUBINDEX_U32

enumerator kSTREAM_PACKET_LAST

```

**enum `STREAM_KEY`**

Enumeration for element types within a stream key.

*Values:*

```

enumerator STREAM_KEY_MODULE_ADDRESS

enumerator STREAM_KEY_CMD

enumerator STREAM_KEY_OPTION

enumerator STREAM_KEY_INDEX

enumerator STREAM_KEY_SUBINDEX

```

**typedef enum Acroname::BrainStem::*Link*::`STREAM_PACKET` `STREAM_PACKET_t`**

Enumeration of stream packet types.

**typedef std::function<*aErr*(const *aPacket* \*packet, void \*pRef)> `streamCallback_t`**

Function signature for streaming callbacks.

**Param *packet***

reference to streaming packet

**Param *pRef***

User provided reference

**Return**

*aErrNone* - Success. Return value is not currently used.

**typedef enum Acroname::BrainStem::*Link*::`STREAM_KEY` `STREAM_KEY_t`**

Enumeration for element types within a stream key.

## Public Functions

**Link** (const *linkSpec* linkSpecifier, const char \*name = "Link")

*Link* Constructor. Takes a fully specified *linkSpec* pointer and creates a link instance with this specifier information.

### Parameters

- **linkSpecifier** – The connection details for a specific module.
- **name** – A name for the link to be created. This name can be used to reference the link during later interactions.

**Link** (const char \*name = "Link")

*Link* constructor without a specifier will most likely use the discoverAndConnect call to create a connection to a link module.

### Parameters

- **name** – A name for the link to be created.

**~Link** (void)

Destructor.

**aErr discoverAndConnect** (const *linkType* type, const uint32\_t serialNumber = 0, const uint8\_t model = 0)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will be made.

### Parameters

- **type** – Transport on which to search for available BrainStem link modules. See the *transport* enum for supported transports.
- **serialNumber** – Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.
- **model** – Acroname model number for the device.

### Returns

*aErrBusy* - if the module is already in use.

### Returns

*aErrParam* - if the transport type is undefined.

### Returns

*aErrNotFound* - if the module cannot be found or if no modules found.

### Returns

*aErrNone* - If the connect was successful.

**aErr connect** (void)

Connect to a link with a fully defined specifier.

### Returns

*aErrBusy* - if the module is running, starting or stopping. Try again in a bit.

### Returns

*aErrDuplicate* - If the module is already connected and running.

### Returns

*aErrConnection* - If there was an error with the connection. User needs to disconnect, then reconnect.

### Returns

*aErrConfiguration* - If the link has an invalid *linkSpec*.

### Returns

*aErrNotFound* - if the module cannot be found.

**Returns**

*aErrNone* If the connect was successful.

**bool isConnected (void)**

Check to see if a module is connected. isConnected looks for a connection to an active module.

**Returns**

true: connected, false: not connected.

**linkStatus getStatus (void)**

Check the status of the module connection.

**Returns**

linkStatus (see aLink.h for status values)

**aErr disconnect (void)**

Disconnect from the BrainStem module.

**Returns**

*aErrResource* - If the there is no valid connection.

**Returns**

*aErrConnection* - If the disconnect failed, due to a communication issue.

**Returns**

*aErrNone* If the disconnect was successful.

**aErr reset (void)**

Reset The underlying link stream.

**Returns**

*aErrResource* - If the there is no valid connection.

**Returns**

*aErrConnection* - If the reset failed, due to a communication issue.

**Returns**

*aErrNone* If the reset was successful.

**const char \*getName (void)**

Accessor for link Name. Returns a pointer to the string representing the link. This string is part of the link, and will be destroyed with it. If you need access to the link name beyond the life of the link, then copy the char\* returned.

**Returns**

Pointer to character array containing the name of the link.

**aErr getLinkSpecifier (*linkSpec* \*spec)**

Accessor for current link specificaiton.

**Parameters**

*spec* -- an allocated empty link spec reference.

**Returns**

*aErrNotFound* - If no *linkSpec* set for current link.

**aErr setLinkSpecifier (const *linkSpec* linkSpecifier)**

Accessor Set current link specification.

**Parameters**

*linkSpecifier* -- The specifier that will replace the current spec.

**Returns**

*aErrBusy* - If link is currently connected.

**aErr getModuleAddress (uint8\_t \*address)**

Gets the module address of the module the link is connected too. A zero is returned if no module can not be determined or if the link is not connected.

*aErr* **sendUEI** (const *uei* packet)

Sends a BrainStem protocol UEI packet on the link. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

**Parameters**

*packet* – The command UEI packet to send.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrParam* - data too long or short.

**Returns**

*aErrPacket* - invalid module address.

**Returns**

*aErrNone* - success.

*aErr* **sendUEI** (const *uei* packet, const uint8\_t subindex)

Sends a BrainStem protocol UEI packet on the link where the packet contains a subindex. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

**Parameters**

- *packet* – The command UEI packet to send.
- *subindex* – The subindex of the command option.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrParam* - data too long or short.

**Returns**

*aErrPacket* - invalid module address.

**Returns**

*aErrNone* - success.

*aErr* **receiveUEI** (const uint8\_t module, const uint8\_t command, const uint8\_t option, const uint8\_t index, *uei* \**packet*)

Awaits receipt of the first available matching UEI packet from the link. The first four arguments describe the packet to wait for. When successful, the supplied uei ref is filled with the received UEI. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

**Parameters**

- *module* – The module address.
- *command* – The command.
- *option* – The uei option.
- *index* – The index of the uei entity.
- *packet* – The uei packet reference to be filled on success.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrPacket* - invalid module address.

**Returns**

*aErrTimeout* - no packet available.

**Returns**

*aErrNone* - success.

*aErr* **receiveUEI** (const uint8\_t module, const uint8\_t command, const uint8\_t option, const uint8\_t index, *uei* \**packet*, aPacketMatchPacketProc proc)

Awaits receipt of the first available matching UEI packet from the link. The first four arguments and proc describe the packet to wait for. When successful, the supplied uei ref is

filled with the received UEI. This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

**Parameters**

- **module** – The module address.
- **command** – The command.
- **option** – The uei option.
- **index** – The index of the uei entity.
- **packet** – The uei packet reference to be filled on success.
- **proc** – The callback used for determining a matching packet.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrPacket* - invalid module address.

**Returns**

*aErrTimeout* - no packet available.

**Returns**

*aErrNone* - success.

***aErr dropMatchingUEIPackets*** (const uint8\_t module, const uint8\_t command, const uint8\_t option, const uint8\_t index)

Drops all existing queued packets that match. from the link. The arguments decribe the packets to be matched This is an advanced interface, please see the relevant section of the reference manual for more information about UEIs.

**Parameters**

- **module** – The module address.
- **command** – The command.
- **option** – The uei option.
- **index** – The index of the uei entity.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrPacket* - invalid module address.

**Returns**

*aErrNone* - success.

***aErr sendPacket*** (const uint8\_t module, const uint8\_t command, const uint8\_t length, const uint8\_t \*data)

Sends a raw BrainStem protocol packet on the link. where the length does not include the module or the command. address byte and can be 0 to aBRAIN-STEM\_MAXPACKETBYTES - 1. This is an advanced interface, please see the relevant section of the reference manual for more information about BrainStem Packet protocol.

**Parameters**

- **module** – The address of the destination module.
- **command** – The length of the data being sent.
- **length** – The length of the data being sent.
- **data** – The data to send.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrParam* - data too long or short.

**Returns**

*aErrPacket* - invalid module address.

**Returns**

*aErrNone* - success.

`aErr receivePacket (const uint8_t module, const uint8_t *match, uint8_t *length, uint8_t *data)`

Awaits receipt of the first available matching raw BrainStem protocol packet from the link where the length does not include the module or command bytes and can be zero. The provided module and match array are compared to packets available and the first match is returned. The supplied data pointer must point to at least aBRAIN-STEM\_MAXPACKETBYTES - 1 bytes. When successful, the data is filled in with the packet data not including the module and command and the length pointer is updated with the length of the returned data.

This is an advanced interface, please see the relevant section of the reference manual for more information about BrainStem Packet protocol.

**Parameters**

- `module` – The module address.
- `match` – A byte array of the values to match for received packets.
- `length` – The length of the match data on entry and length of the returned data filled on success.
- `data` – The data filled on success.

**Returns**

`aErrConnection` - link not connected.

**Returns**

`aErrPacket` - invalid module address.

**Returns**

`aErrTimeout` - no packet available.

**Returns**

`aErrNone` - success.

`aErr loadStoreSlot (const uint8_t module, const uint8_t store, const uint8_t slot, const uint8_t *pData, const size_t length)`

Loads data into a BrainStem Slot. See the relevant section of the BrainStem reference for information about BrainStem Slots and Stores.

**Parameters**

- `module` -- *Module* address.
- `store` -- BrainStem store to access, possibilities include Internal, RAM, and SD.
- `slot` -- The Slot within the Brainstem store to place the data.
- `pData` -- Pointer to a buffer containing the data to load.
- `length` -- The length in bytes of the data buffer to write.

**Returns**

`aErrConnection` - link not connected.

**Returns**

`aErrParam` - invalid module address.

**Returns**

`aErrCancel` - The write process is closing and this call was unable to successfully complete.

**Returns**

`aErrNone` - success.

`aErr unloadStoreSlot (const uint8_t module, const uint8_t store, const uint8_t slot, uint8_t *pData, const size_t dataLength, size_t *pNRead)`

Unloads data from a BrainStem Slot. If there are no read.

reference for information about BrainStem Slots and Stores.

**Parameters**

- `module` -- *Module* address.

- **store** -- BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** -- The Slot within the Brainstem store to place the data.
- **pData** -- Pointer to a buffer with dataLength space in bytes that will be filled by the call.
- **dataLength** -- Expected length of the data, and at most the size of the pData buffer.
- **pNRead** -- The number of bytes actually read.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrParam* - invalid module address.

**Returns**

*aErrCancel* - The write process is closing and this call was unable to successfully complete.

**Returns**

*aErrOverrun* - The read would overrun the buffer, i.e there is more data in the slot than the buffer can handle.

**Returns**

*aErrNone* - success.

***aErr storeSlotSize*** (const uint8\_t module, const uint8\_t store, const uint8\_t slot, size\_t \*size)

Returns the current size of the data loaded in the slot specified.

**Parameters**

- **module** -- *Module* address.
- **store** -- BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** -- The Slot within the Brainstem store to place the data.
- **size** -- size in bytes of the data stored in the slot.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrParam* - invalid module address.

**Returns**

*aErrCancel* - The write process is closing and this request was unable to successfully complete.

**Returns**

*aErrNone* - success.

***aErr storeSlotCapacity*** (const uint8\_t module, const uint8\_t store, const uint8\_t slot, size\_t \*capacity)

Returns the maximum data capacity of the slot specified.

**Parameters**

- **module** -- *Module* address.
- **store** -- BrainStem store to access, possibilities include Internal, RAM, and SD.
- **slot** -- The Slot within the Brainstem store to place the data.
- **capacity** -- size in bytes of the data stored in the slot.

**Returns**

*aErrConnection* - link not connected.

**Returns**

*aErrParam* - invalid module address.

**Returns**

*aErrCancel* - The write process is closing and this request was unable to successfully complete.

**Returns**

*aErrNone* - success.

***aErr enableStream*** (const uint8\_t moduleAddress, const uint8\_t cmd, const uint8\_t option, const uint8\_t index, const bool enable)

Enables streaming for the supplied criteria.

**Parameters**

- **moduleAddress** – Address to filter on.
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - Enables streaming; False - disables streaming

***aErr isLinkStreaming*** (const uint8\_t moduleAddress, uint8\_t \*enabled)

Determines if the module is actively streaming. Does not indicate what is streaming, only if streaming is currently active.

**Parameters**

- **moduleAddress** – The devices module address.
- **enabled** – Variable to be populated.

**Returns**

Returns common entity return values

***aErr registerStreamCallback*** (const uint8\_t moduleAddress, const uint8\_t cmd, const uint8\_t option, const uint8\_t index, const bool enable, *streamCallback\_t* cb, void \*pRef)

Registers a callback function based on a specific module, cmd, option, and index.

**Parameters**

- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **enable** – True - installs/updates callback and ref; False - uninstalls callback
- **cb** – Callback to be executed when a new packet matching the criteria is received.
- **pRef** – Pointer to user reference for use inside the callback function.

**Returns**

*aErrNotFound* - Item not found (uninstalling only)

**Returns**

*aErrNone* - success

***aErr getStreamValue*** (const uint8\_t moduleAddress, const uint8\_t cmd, const uint8\_t option, const uint8\_t index, const uint8\_t subindex, uint32\_t \*value)

Gets stream value based on the search criteria

**Parameters**

- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)

**Returns**

*aErrStreamStale* if the value has not been updated since the last read.

**Returns**

*aErrNotFound* if no such stream element exists.

**Returns**

aErrNone - success

*aErr* **getStreamStatus** (const uint8\_t moduleAddress, const uint8\_t cmd, const uint8\_t option, const uint8\_t index, const uint8\_t subindex, std::map<uint64\_t, uint32\_t> \*status)

Gets all available stream values based on the search criteria.

**Parameters**

- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **status** – map of key value pairs to be filled based on the parameters.

*Link::getStreamKeyElement* should be used to decode the keys

**Returns**

aErrParam if status is null

**Returns**

aErrNone - success

std::vector<uint64\_t> **filterActiveStreamKeys** (const uint8\_t moduleAddress, const uint8\_t cmd, const uint8\_t option, const uint8\_t index, const uint8\_t subindex, const bool acquireLock)

Provides a list of active stream keys based on the supplied criteria. Exposed for unit-testing purposes only.

**Parameters**

- **moduleAddress** – Address to filter on (supports Wildcards)
- **cmd** – cmd to filter by (supports Wildcards)
- **option** – option to filter by (supports Wildcards)
- **index** – index to filter by (supports Wildcards)
- **acquireLock** – Option to acquire mutex before getting list elements.

**Returns**

List of keys meeting the search criteria

*aErr* **enablePacketLog** (const char \*logname)

Enable Packet logging.

Enable packet logging for this link. Enables the packet logging buffer, and writes packet traffic out to the file specified by logname.

**Parameters**

- **logname** – the path and filename indicating where to write the packet log.

**Returns**

aErr returns appropriate errors if it fails to enable the packet log.

*aErr* **disablePacketLog** (void)

Disable Packet logging.

disable packet logging for this link. Disables the packet log.

**Returns**

aErr returns appropriate errors if it fails to disable the debug log.

*aErr* **getFactoryData** (const uint8\_t module, const uint8\_t command, uint8\_t \*pData, const size\_t dataLength, size\_t \*unloadedLength)

For Internal use only!

*aErr* **setFactoryData** (const uint8\_t module, const uint8\_t command, const uint8\_t \*pData, const size\_t dataLength)

For Internal use only!

## Public Static Functions

```
static inline aErr sDiscover (const linkType type, aDiscoveryModuleFoundProc  
                           cbLinkFound, void *vpCBRef)
```

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.

### Returns

[aErrNotFound](#) if no devices were found.

### Returns

[aErrNone](#) on success.

```
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.

### Returns

true To continue processing, or false to stop processing.

```
static inline aErr sDiscover (const linkType type, list<linkSpec> *devices)
```

Discover is called with a specified transport to search for link modules on that transport. The devices list is filled with device specifiers. sDiscover returns aErrNone if the discovery process is successful, regardless of whether any links are found. An error is only returned if the link discovery process fails. Discovery can take some time.

### Parameters

- **type** – Transport to search for available BrainStem link modules on. See the [transport](#) enum for supported transports.
- **devices** – an empty list of specifiers that will be filled in.

### Returns

[aErrNotFound](#) if no devices were found.

### Returns

[aErrNone](#) on success.

```
static bool getStreamPacketType (const aPacket *packet, STREAM_PACKET_t *type)
```

Decodes the streaming packet type from a provided packet.

### Parameters

- **packet** – The packet to be interrogated.
- **type** – variable to be populated. Filled with kSTREAM\_PACKET\_UNKNOWN on failure.

**Returns**

true on success; false on failure.

**static bool `isSubindexType` (`STREAM_PACKET_t` type)**

Helper function for indicating whether the packet is a subindex type. The subindex can be queried through `Link::getStreamKeyElement`

**Parameters**

`type` -- The element to evaluate.

**Returns**

true if the type contains a subindex; false if it does not.

**static uint8\_t `getStreamKeyElement` (const uint64\_t key, `STREAM_KEY_t` element)**

Convenience function to unpack a stream key. Note: This function will assert if an out of range `STREAM_KEY_t` is used.

**Parameters**

- `key` – The key to be unpacked
- `element` – The element to unpack from the key.

**Returns**

The requested element from the key.

**static bool `isStreamPacket` (const `aPacket` \*packet)**

Convenience function to determine whether the value is a stream packet. Stream "Packets" encompass all `STREAM_PACKET_t` valid elements.

**Parameters**

`packet` – UEI stream packet to be checked.

**Returns**

Whether the packet is a stream sample or not

**static bool `isStreamSample` (const `aPacket` \*packet)**

Convenience function to determine whether the value is a stream sample. Stream "Samples" encompasses all `STREAM_KEY_t` except for `kSTREAM_PACKET_BYTES` which have a varied structure and depend on the cmd/option/index. Calling `isStreamPacket` prior is not required as this function will verify the packet type

**Parameters**

`packet` – UEI stream packet to be checked.

**Returns**

Whether the packet is a stream sample or not

**static `aErr getStreamSample` (const `aPacket` \*packet, uint64\_t \*timestamp = NULL, uint32\_t \*value = NULL, uint8\_t \*subindex = NULL)**

Convenience function to unpack the stream samples timestamp and value. Calling `isStreamSample` prior is not required as this function will verify the packet type.

**Parameters**

- `packet` – UEI stream packet to be unpacked.
- `timestamp` – Variable to be filled with stream sample timestamp. (optional)
- `value` – Variable to be filled with the stream sample (optional). May require casting to signed value depending on the cmd/option code.

**Returns**

`aErrPacket` - Not a stream packet

**Returns**

`aErrUnknown` - Unknown decoding issue.

**Returns**

`aErrNone` - success.

**static void `getTimestampParts` (const uint64\_t timestamp, uint32\_t \*seconds, uint32\_t \*uSeconds)**

Helper function for extracting the parts of a timestamp.

**Parameters**

- **timestamp** -- Value acquired from [Link::getStreamSample](#)
- **seconds** -- Seconds element from timestamp. Refers to the seconds since firmware boot.
- **uSeconds** -- Micro second element from the timestamp. Refers to the micro seconds from firmware boot. Micro seconds rolls over to seconds. Value range: 0-99999

static bool **linkStreamFilter** (const [aPacket](#) \*packet, void \*ref)

Filter function for Streaming packets. This is used internally whenever streaming is enabled. Exposed for unit-testing purposes only.

**Parameters**

- **packet** – UEI stream packet to be checked/filtered.
- **ref** – Opaque reference handle

### 3.3.12 Module Class

class **Module**

The [Module](#) class provides a generic interface to a BrainStem hardware module. The [Module](#) class is the parent class for all BrainStem modules. Each module inherits from [Module](#) and implements its hardware specific features.

Subclassed by [a40PinModule](#), [aMTMDAQ1](#), [aMTMDAQ2](#), [aMTMIOSerial](#), [aMTMLoad1](#), [aMTMPM1](#), [aMTMRelay](#), [aMTMStemModule](#), [aUSBCSwitch](#), [aUSBHub2x4](#), [aUSBHub3c](#), [aUSBHub3p](#)

#### Public Functions

**Module** (const uint8\_t address, const uint8\_t model = 0)

Constructor. Implicitly creates a link object with no specifier. Most often objects created with this constructor will use [linkDiscoverAndConnect](#) to find and connect to a module.

**Parameters**

- **address** – The BrainStem network address of the module. The default address (or base address for modules that support address offsets) is defined in each module's "Defs.h" header.
- **model** – Acroname model number.

virtual ~**Module** (void)

Destructor.

[aErr](#) **connect** (const [linkType](#) type, const uint32\_t serialNum)

Connect using the current link specifier.

**Parameters**

- **type** -- Transport on which to search for available BrainStem link modules. See the [transport](#) enum for supported transports.
- **serialNum** -- Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

**Returns**

[aErrBusy](#) - if the module is already in use.

**Returns**

*aErrParam* - if the type is incorrect or serialNum is not specified

**Returns**

*aErrNotFound* - if the module cannot be found.

**Returns**

*aErrNone* If the connect was successful.

***aErr connectFromSpec*** (const *linkSpec* linkSpecifier)

Connect to a link with a fully defined specifier.

**Parameters**

*linkSpecifier* -- Connect to module with specifier.

**Returns**

*aErrInitialization* - If there is currently no link object.

**Returns**

*aErrBusy* - If the link is currently connected.

**Returns**

*aErrParam* - if the specifier is incorrect.

**Returns**

*aErrNotFound* - if the module cannot be found.

**Returns**

*aErrNone* If the connect was successful.

***aErr discoverAndConnect*** (*linkType* type, const uint32\_t serialNum = 0)

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will

**Parameters**

- *type* -- Transport on which to search for available BrainStem link modules.

See the *transport* enum for supported transports.

- *serialNum* -- Specify a serial number to connect to a specific link module.

Use 0 to connect to the first link module found.

**Returns**

*aErrBusy* - if the module is already in use.

**Returns**

*aErrParam* - if the transport type is undefined.

**Returns**

*aErrNotFound* - if the module cannot be found.

**Returns**

*aErrNone* If the connect was successful.

***aErr connectThroughLinkModule*** (*Module* \*pModule)

Connect using link from another *Module*. This member function will connect to the same BrainStem used by given *Module*. If a link module is found on the specified transport, a connection will

**Parameters**

*pModule* -- Pointer to a valid *Module* class object.

**Returns**

*aErrParam* - if the module is undefined.

**Returns**

*aErrNone* - if the connect was successful.

**bool isConnected** (void)

Is the link connected to the BrainStem *Module*.

*linkStatus* **getStatus** (void)

Check the status of the BrainStem module connection.

**Returns**

*linkStatus* (see aLink.h for status values)

*aErr* **disconnect** (void)

Disconnect from the BrainStem module.

**Returns**

*aErrResource* - If there is no valid connection.

**Returns**

*aErrConnection* - If the disconnect failed, due to a communication issue.

**Returns**

*aErrNone* If the disconnect was successful.

*aErr* **reconnect** ()

Reconnect using the current link specifier.

**Returns**

*aErrBusy* - if the module is already in use.

**Returns**

*aErrParam* - if the specifier is incorrect.

**Returns**

*aErrNotFound* - if the module cannot be found.

**Returns**

*aErrNone* If the connect was successful.

*Link* \***getLink** (void) const

Get the current link object.

**Returns**

The link associated with the module.

*uint8\_t* **getModuleAddress** (void) const

Accessor to get the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

**Returns**

The module address.

**void** **setModuleAddress** (const *uint8\_t* address)

Accessor to set the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

**Parameters**

*address* - The module address.

*aErr* **getLinkSpecifier** (*linkSpec* \*spec)

Get linkSpecifier

**Parameters**

*spec* - - allocated linkspec struct will be filled with spec.

**Returns**

*aErrNone* - If the module does not have a spec.

*aErr* **hasUEI** (const *uint8\_t* command, const *uint8\_t* option, const *uint8\_t* index, const *uint8\_t* flags)

Queries the module to determine if it implements a UEI. Each UEI has a command, option or variant, index and flag. The hasUEI method queries for a fully specified UEI. Returns

`aErrNone` if the variation is supported and an appropriate error if not. This call is blocking for up to the nMSTimeout period.

**Parameters**

- `command` – One of the UEI commands (cmdXXX).
- `option` – The option or variant of the command.
- `index` – The entity index.
- `flags` – The flags (ueiOPTION\_SET or ueiOPTION\_GET).

**Returns**

`aErrNone` - The module supports this command and access flags.

**Returns**

`aErrMode` - The module supports the command but not the access flag.

**Returns**

`aErrNotFound` - The module does not support the command, option, or index.

**Returns**

`aErrTimeout` - The request timed out without a response.

**Returns**

`aErrConnection` - There is no active link

`aErr classQuantity` (const uint8\_t command, uint8\_t \*count)

Queries the module to determine how many entities of the specified class are implemented by the module. Zero is a valid return value. For example, calling `classQuantity` with the command parameter of cmdANALOG would return the number of analog entities implemented by the module.

**Parameters**

- `command` – One of the UEI commands (cmdXXX).
- `count` – When the request is successful count is updated with the number of entities found.

**Returns**

`aErrNone` - Success.

**Returns**

`aErrTimeout` - The request timed out without a response.

**Returns**

`aErrConnection` - There is no active link.

`aErr subClassQuantity` (const uint8\_t command, const uint8\_t index, uint8\_t \*count)

Queries the module to determine how many subclass entities of the specified class are implemented by the module for a given entity index. This is used for entities which may be 2-dimensional. E.g. cmdMUX subclasses are the number of channels supported by a particular mux type (index); as a specific example, a module may support 4 UART channels, so `subClassQuantity(cmdMUX, aMUX_UART...)` could return 4. Zero is a valid return value.

**Parameters**

- `command` – One of the UEI commands (cmdXXX).
- `index` – The entity index.
- `count` – The number of subclasses found.

**Returns**

`aErrNone` - Success.

**Returns**

`aErrTimeout` - The request timed out waiting for response.

**Returns**

`aErrConnection` - There is no active link.

`aErr entityGroup` (const uint8\_t command, const uint8\_t index, uint8\_t \*group)

Queries the module the group assigned to an entity and index. Entities groups are used to specify when certain hardware features are fundamentally related. E.g. certain hardware

modules may have some digital pins associated with an adjustable voltage rail; these digitals would be in the same group as the rail. Zero is the default group.

**Parameters**

- **command** – One of the UEI commands (cmdXXX).
- **index** – The entity index.
- **group** – Upon success, group is filled with the entities group value.

**Returns**

*aErrNone* - Success.

**Returns**

*aErrTimeout* - The request timed out without response.

**Returns**

*aErrConnection* - There is no active link.

**aErr debug** (const uint8\_t \*pData, const uint8\_t length)

Sends a debug packet to the module containing the provided data. Modules receiving debug packets simply echo the packet back to the sender. If the round-trip is successful, the reply data will match the data sent. This method returns aErrNone when successful, if not successful, an appropriate error is returned.

**Parameters**

- **pData** – A pointer to an array of data to be sent in the debug packet.
- **length** – The length of the data array.

**Returns**

*aErrNone* - Success.

**Returns**

*aErrTimeout* - Timeout occurred without response.

**Returns**

*aErrConnection* - No active link exists.

**void setNetworkingMode** (const bool mode)

Sets the networking mode of the module object. By default the module object is configured to automatically adjust its address based on the device's current module address. So that, if the device has a software or hardware offset it will still be able to communicate with the device. If advanced networking is required the auto networking mode can be turned off.

**Parameters**

- **mode** – True/1 for Auto Networking, False/0 for manual networking

### 3.3.13 Mux Class

class **MuxClass** : public Acroname::BrainStem::*EntityClass*

MARK: Mux Class.

**MuxClass**. A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

## Public Functions

**MuxClass** (void)

Constructor.

**~MuxClass** (void)

Destructor.

**void init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity, i.e. aMUX\_UART or aMUX\_USB.

**aErr getEnable** (uint8\_t \*bEnabled)

Get the mux enable/disable status

**Parameters**

**bEnabled** – true: mux is enabled, false: the mux is disabled.

**Returns**

Returns common entity return values

**aErr setEnable** (const uint8\_t bEnable)

Enable the mux.

**Parameters**

**bEnable** – true: enables the mux for the selected channel.

**Returns**

Returns common entity return values

**aErr getChannel** (uint8\_t \*channel)

Get the current selected mux channel.

**Parameters**

**channel** – Indicates which channel is selected.

**Returns**

Returns common entity return values

**aErr setChannel** (const uint8\_t channel)

Set the current mux channel.

**Parameters**

**channel** – mux channel to select.

**Returns**

Returns common entity return values

**aErr getChannelVoltage** (const uint8\_t channel, int32\_t \*microvolts)

Get the voltage of the indicated mux channel.

---

**Note:** Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

**Parameters**

- **channel** – The channel in which voltage was requested.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

**Returns**

Returns common entity return values

```
aErr getConfiguration (int32_t *config)
Get the configuration of the mux.
Parameters
    config - integer representing the mux configuration either default, or split-mode.
Returns
    Returns common entity return values

aErr setConfiguration (const int32_t config)
Set the configuration of the mux.
Parameters
    config - integer representing the mux configuration either muxConfig_default,
        or muxConfig_splitMode.
Returns
    Returns common entity return values

aErr getSplitMode (int32_t *splitMode)
Get the current split mode mux configuration.
Parameters
    splitMode - integer representing the channel selection for each sub-channel
        within the mux. See the data-sheet for the device for specific information.
Returns
    Returns common entity return values

aErr setSplitMode (const int32_t splitMode)
Sets the mux's split mode configuration.
Parameters
    splitMode - integer representing the channel selection for each sub-channel
        within the mux. See the data-sheet for the device for specific information.
Returns
    Returns common entity return values
```

### 3.3.14 Pointer Class

```
class PointerClass : public Acroname::BrainStem::EntityClass
```

MARK: Pointer Class.

*PointerClass*. Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

## Public Functions

**PointerClass** (void)

Constructor.

**~PointerClass** (void)

Destructor.

**void init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – The index the pointer element index.

**aErr getOffset** (uint16\_t \*offset)

Get the offset of the pointer

**Parameters**

- **offset** – The value of the offset.

**Returns**

All possible standard UEI return values.

**aErr setOffset** (uint16\_t offset)

Set the offset of the pointer

**Parameters**

- **offset** – The value of the offset.

**Returns**

All possible standard UEI return values.

**aErr getMode** (uint8\_t \*mode)

Get the mode of the pointer

**Parameters**

- **mode** – The mode: aPOINTER\_MODE\_STATIC or aPOINTER\_MODE\_AUTO\_INCREMENT.

**Returns**

All possible standard UEI return values.

**aErr setMode** (uint8\_t mode)

Set the mode of the pointer

**Parameters**

- **mode** – The mode: aPOINTER\_MODE\_STATIC or aPOINTER\_MODE\_AUTO\_INCREMENT.

**Returns**

All possible standard UEI return values.

**aErr getTransferStore** (uint8\_t \*handle)

Get the handle to the store.

**Parameters**

- **handle** – The handle of the store.

**Returns**

All possible standard UEI return handles.

**aErr setTransferStore** (uint8\_t handle)

Set the handle to the store.

**Parameters**

- **handle** – The handle of the store.

**Returns**

All possible standard UEI return handles.

*aErr* **initiateTransferToStore** (uint8\_t length)

Transfer data to the store.

**Parameters**

length – The length of the data transfer.

**Returns**

All possible standard UEI return values.

*aErr* **initiateTransferFromStore** (uint8\_t length)

Transfer data from the store.

**Parameters**

length – The length of the data transfer.

**Returns**

All possible standard UEI return values.

*aErr* **getChar** (uint8\_t \*value)

Get a char (1 byte) value from the pointer at this object's index, where elements are 1 byte long.

**Parameters**

value – The value of a single character (1 byte) stored in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **setChar** (const uint8\_t value)

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

**Parameters**

value – The single char (1 byte) value to be stored in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **getShort** (uint16\_t \*value)

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long

**Parameters**

value – The value of a single short (2 byte) stored in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **setShort** (const uint16\_t value)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

**Parameters**

value – The single short (2 byte) value to be set in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **getInt** (uint32\_t \*value)

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

**Parameters**

value – The value of a single int (4 byte) stored in the pointer.

**Returns**

All possible standard UEI return values.

`aErr setInt (const uint32_t value)`

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

**Parameters**

`value` - The single int (4 byte) value to be stored in the pointer.

**Returns**

All possible standard UEI return values.

### 3.3.15 Port Class

`class PortClass : public Acroname::BrainStem::EntityClass`

MARK: Port Class.

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 set-DataHSEnabled.

**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 set-DataHSEnabled.

**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 `setDataEnabled`.

**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 `setEnabled`.

**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* **getVconnEnabled**(uint8\_t \*enable)

Gets the current enable value of the Vconn lines. Sub-component of getVconnEnabled.

**Parameters**

**enable** – 1 = Vconn enabled; 0 = Vconn 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* **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* **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 product datasheet for details.

**Returns**

Returns common entity return values

*aErr* **getMode** (uint32\_t \*mode)

Gets current mode of the port

**Parameters**

`mode` – Bit mapped value representing the ports mode. See product datasheet for details.

**Returns**

Returns common entity return values

`aErr setMode (const uint32_t mode)`

Sets the mode of the port

**Parameters**

`mode` – Port mode to be set. See product datasheet for details.

**Returns**

Returns common entity return values

`aErr getErrors (uint32_t *errors)`

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

**Parameters**

`errors` – Bit mapped field representing the current errors of the ports

**Returns**

Returns common entity return values

`aErr getCurrentLimit (uint32_t *limit)`

Gets the current limit of the port.

**Parameters**

`limit` – Variable to be filled with the limit in microAmps (uA).

**Returns**

Returns common entity return values

`aErr setCurrentLimit (const uint32_t limit)`

Sets the current limit of the port.

**Parameters**

`limit` – Current limit to be applied in microAmps (uA).

**Returns**

Returns common entity return values

`aErr getCurrentLimitMode (uint8_t *mode)`

Gets the current limit mode. The mode determines how the port will react to an over current condition.

**Parameters**

`mode` – Variable to be filled with an enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

`aErr setCurrentLimitMode (const uint8_t mode)`

Sets the current limit mode. The mode determines how the port will react to an over current condition.

**Parameters**

`mode` – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

`aErr getAvailablePower (uint32_t *power)`

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

**Parameters**

`power` – Variable to be filled with the available power in milli-watts (mW).

**Returns**

Returns common entity return values

`aErr getAllocatedPower (int32_t *power)`

Gets the currently allocated power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

**Parameters**

`power` – Variable to be filled with the allocated power in milli-watts (mW).

**Returns**

Returns common entity return values

`aErr getPowerLimit (uint32_t *limit)`

Gets the user defined power limit for the port.

**Parameters**

`limit` – Variable to be filled with the power limit in milli-watts (mW).

**Returns**

Returns common entity return values

`aErr setPowerLimit (const uint32_t limit)`

Sets a user defined power limit for the port.

**Parameters**

`limit` – Power limit to be applied in milli-watts (mW).

**Returns**

Returns common entity return values

`aErr getPowerLimitMode (uint8_t *mode)`

Gets the power limit mode. The mode determines how the port will react to an over power condition.

**Parameters**

`mode` – Variable to be filled with an enumerated representation of the power limit mode. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

`aErr setPowerLimitMode (const uint8_t mode)`

Sets the power limit mode. The mode determines how the port will react to an over power condition.

**Parameters**

`mode` – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

`aErr getName (uint8_t *buffer, const size_t bufLength, size_t *unloadedLength)`

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

**Parameters**

- `buffer` – pointer to the start of a c style buffer to be filled
- `bufLength` – Length of the buffer to be filed
- `unloadedLength` – Length that was actually received and filled.

**Returns**

Returns common entity return values

*aErr* **setName** (uint8\_t \*buffer, const size\_t bufLength)

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

**Parameters**

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

*aErr* **getDataHSRoutingBehavior** (uint8\_t \*mode)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **setDataHSRoutingBehavior** (const uint8\_t mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **getDataSSRoutingBehavior** (uint8\_t \*mode)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **setDataSSRoutingBehavior** (const uint8\_t mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **getVbusAccumulatedPower** (int32\_t \*milliwatthours)

Gets the Vbus Accumulated Power

**Parameters**

**milliwatthours** – The accumulated power on Vbus in milliwatt-hours.

**Returns**

Returns common entity return values

**aErr** **resetVbusAccumulatedPower** (void)  
Resets the Vbus Accumulated Power to zero.

**Returns**  
Returns common entity return values

**aErr** **getVconnAccumulatedPower** (int32\_t \*milliwatthours)  
Gets the Vconn Accumulated Power

**Parameters**  
**milliwatthours** – The accumulated power on Vconn in milliwatt-hours.

**Returns**  
Returns common entity return values

**aErr** **resetVconnAccumulatedPower** (void)  
Resets the Vconn Accumulated Power to zero.

**Returns**  
Returns common entity return values

**aErr** **setHSBoost** (const uint8\_t boost)  
Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Parameters**  
**boost** – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

**Returns**  
Returns common entity return values

**aErr** **getHSBoost** (uint8\_t \*boost)  
Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Parameters**  
**boost** – An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.

**Returns**  
Returns common entity return values

**aErr** **resetEntityToFactoryDefaults** (void)  
Resets the *PortClass* Entity to it factory default configuration.

**Returns**  
Returns common entity return values

### 3.3.16 Power Delivery Class

```
class PowerDeliveryClass : public Acroname::BrainStem::EntityClass
```

MARK: Power Delivery Class.

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** (void)

Constructor.

**~PowerDeliveryClass** (void)

Destructor.

**aErr getConnectionState** (uint8\_t \*state)

Gets the current state of the connection in the form of an enumeration.

### Parameters

state - Pointer to be filled with the current connection state.

### Returns

Returns common entity return values

**aErr getNumberOfPowerDataObjects** (const uint8\_t partner, const uint8\_t powerRole, uint8\_t \*numRules)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

### Parameters

- **partner** - Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** - Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **numRules** - Variable to be filled with the number of PDOs.

### Returns

Returns common entity return values

**aErr getPowerDataObject** (const uint8\_t partner, const uint8\_t powerRole, const uint8\_t ruleIndex, uint32\_t \*pdo)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

### Parameters

- **partner** - Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** - Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** - The index of the PDO in question. Valid index are 1-7.
- **pdo** - Variable to be filled with the requested power rule.

### Returns

Returns common entity return values

**aErr setPowerDataObject** (const uint8\_t powerRole, const uint8\_t ruleIndex, const uint32\_t pdo)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

### Parameters

- **powerRole** - Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** - The index of the PDO in question. Valid index are 1-7.
- **pdo** - Power Data Object to be set.

### Returns

Returns common entity return values

`aErr resetPowerDataObjectToDefault (const uint8_t powerRole, const uint8_t ruleIndex)`

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

#### Parameters

- **powerRole** - Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** - The index of the PDO in question. Valid index are 1-7.

#### Returns

Returns common entity return values

`aErr getPowerDataObjectList (uint32_t *buffer, const size_t bufLength, size_t *unloadedLength)`

Gets all Power Data Objects (PDOs). Equivalent to calling `PowerDeliveryClass::getPowerDataObject()` on all partners, power roles, and index's.

#### Parameters

- **buffer** - pointer to the start of a c style buffer to be filled The order of which is:
  - Rules 1-7 Local Source
  - Rules 1-7 Local Sink
  - Rules 1-7 Partner Source
  - Rules 1-7 Partner Sink.
- **bufLength** - Length of the buffer to be 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 uint8\_t partner, const uint32\_t rdo)

Sets the current Request Data Object (RDO) for a given partner. (Only the local partner can be changed.) RDOs: Are provided by the sinking device. Exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

**Parameters**

- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
- **rdo** – Request Data Object to be set.

**Returns**

Returns common entity return values

*aErr* **getPowerRole** (uint8\_t \*powerRole)

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

**Parameters**

- powerRole** – Variable to be 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 (0)

**Returns**

Returns common entity return values

`aErr request (const uint8_t request)`

Requests an action of the Remote partner. Actions are not guaranteed to occur.

**Parameters**

`request` – Request to be issued to the remote partner

- pdRequestHardReset (0)
- pdRequestSoftReset (1)
- pdRequestDataReset (2)
- pdRequestPowerRoleSwap (3)
- pdRequestPowerFastRoleSwap (4)
- pdRequestDataRoleSwap (5)
- pdRequestVconnSwap (6)
- pdRequestSinkGoToMinimum (7)
- pdRequestRemoteSourcePowerDataObjects (8)
- pdRequestRemoteSinkPowerDataObjects (9)

**Returns**

The returned error represents the success of the request being sent to the partner only. The success of the request being serviced by the remote partner can be obtained through `PowerDeliveryClass::requestStatus()` Returns common entity return values

`aErr requestStatus (uint32_t *status)`

Gets the status of the last request command sent.

**Parameters**

`status` – Variable to be filled with the status

**Returns**

Returns common entity return values

`aErr getOverride (uint32_t *overrides)`

Gets the current enabled overrides

**Parameters**

`overrides` – Bit mapped representation of the current override configuration.

**Returns**

Returns common entity return values

`aErr setOverride (const uint32_t overrides)`

Sets the current enabled overrides

**Parameters**

`overrides` – Overrides to be set in a bit mapped representation.

**Returns**

Returns common entity return values

`aErr resetEntityToFactoryDefaults (void)`

Resets the *PowerDeliveryClass* Entity to it factory default configuration.

`aErr getFlagMode (const uint8_t flag, uint8_t *mode)`

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters**

- `flag` – Flag/Advertisement to be modified
- `mode` – Variable to be filled with the current mode.
  - Disabled (0)
  - Enabled (1)
  - Auto (2) default

**Returns**

Returns common entity return values

`aErr setFlagMode (const uint8_t flag, const uint8_t mode)`

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters**

- `flag` – Flag/Advertisement to be modified
- `mode` – Value to be applied.
  - Disabled (0)
  - Enabled (1)
  - Auto (2) default

**Returns**

Returns common entity return values

`aErr getPeakCurrentConfiguration (uint8_t *configuration)`

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

**Parameters**

**configuration** – An enumerated value referring to the current configuration.  
 • Allowable values are 0 - 4

**Returns**

Returns common entity return values

*aErr* **setPeakCurrentConfiguration** (const uint8\_t configuration)

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

**Parameters**

**configuration** – An enumerated value referring to the configuration to be set  
 • Allowable values are 0 - 4

**Returns**

Returns common entity return values

*aErr* **getFastRoleSwapCurrent** (uint8\_t \*swapCurrent)

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

**Parameters**

**swapCurrent** – An enumerated value referring to current swap value.  
 • 0A (0)  
 • 900mA (1)  
 • 1.5A (2)  
 • 3A (3)

**Returns**

Returns common entity return values

*aErr* **setFastRoleSwapCurrent** (const uint8\_t swapCurrent)

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

**Parameters**

**swapCurrent** – An enumerated value referring to value to be set.  
 • 0A (0)  
 • 900mA (1)  
 • 1.5A (2)  
 • 3A (3)

**Returns**

Returns common entity return values

**Public Static Functions**

static *aErr* **packDataObjectAttributes** (uint8\_t \*attributes, const uint8\_t partner, const uint8\_t powerRole, const uint8\_t ruleIndex)

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

**Parameters**

- **attributes** – variable to be filled with packed values.
- **partner** – Indicates which side of the PD connection.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
  - Source = 1 = powerdeliveryPowerRoleSource

- Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

**Returns**

aErrNone on success; aErrParam with bad input.

```
static aErr unpackDataObjectAttributes (const uint8_t attributes, uint8_t *partner,  
                                     uint8_t *powerRole, uint8_t *ruleIndex)
```

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

**Parameters**

- **attributes** – variable to be filled with packed values.
- **partner** – Indicates which side of the PD connection.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

**Returns**

aErrNone on success; aErrParam with bad input.

### 3.3.17 Rail Class

```
class RailClass : public Acroname::BrainStem::EntityClass
```

MARK: Rail Class.

*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 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 determine if this is a measured or stored value.

**Returns**

Returns common entity return values

*aErr* **setVoltageSetpoint** (const int32\_t microvolts)

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

**microvolts** – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

**Returns**

Returns common entity return values

*aErr* **getVoltageSetpoint** (int32\_t \*microvolts)

Get the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

**microvolts** – The voltage in micro-volts (1 == 1e-6V) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

*aErr* **setVoltageMinLimit** (const int32\_t microvolts)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

**microvolts** – The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

*aErr* **getVoltageMinLimit** (int32\_t \*microvolts)

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

**microvolts** – The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

`aErr setVoltageMaxLimit (const int32_t microvolts)`

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

`microvolts` - The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

`aErr getVoltageMaxLimit (int32_t *microvolts)`

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

`microvolts` - The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

`aErr getPower (int32_t *milliwatts)`

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

`milliwatts` - The power in milli-watts (1 == 1e-3W) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

`aErr setPowerSetpoint (const int32_t milliwatts)`

Set the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

`milliwatts` - The power in milli-watts (1 == 1e-3W) to be supplied by the rail.

**Returns**

Returns common entity return values

`aErr getPowerSetpoint (int32_t *milliwatts)`

Get the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

`milliwatts` - The power in milli-watts (1 == 1e-3W) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

`aErr setPowerLimit (const int32_t milliwatts)`

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

`milliwatts` - The power in milli-watts (mW).

**Returns**

Returns common entity return values

`aErr getPowerLimit (int32_t *milliwatts)`

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

`milliwatts` – The power in milli-watts (mW).

**Returns**

Returns common entity return values

`aErr getResistance (int32_t *millionohms)`

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**

`millionohms` – The resistance in milli-ohms (1 == 1e-3Ohms) currently drawn by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the `setResistance` interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

`aErr setResistanceSetpoint (const int32_t millionohms)`

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**

`millionohms` – The resistance in milli-ohms (1 == 1e-3Ohms) to be drawn by the rail.

**Returns**

Returns common entity return values

`aErr getResistanceSetpoint (int32_t *millionohms)`

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**

`millionohms` – The resistance in milli-ohms (1 == 1e-3Ohms) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the `setResistance` interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

`aErr setKelvinSensingEnable (const uint8_t bEnable)`

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

`bEnable` – enable or disable kelvin sensing.

**Returns**

Returns common entity return values

`aErr getKelvinSensingEnable (uint8_t *bEnable)`

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

`bEnable` – Kelvin sensing is enabled or disabled.

**Returns**

Returns common entity return values

`aErr getKelvinSensingState (uint8_t *state)`

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

`state` – Kelvin sensing is enabled or disabled.

**Returns**

Returns common entity return values

`aErr setOperationalMode (const uint8_t mode)`

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

**Parameters**

`mode` – The operational mode to employ.

**Returns**

Returns common entity return values

`aErr getOperationalMode (uint8_t *mode)`

Determine the current operational mode of the system. Refer to the module datasheet for definition of the rail operational mode capabilities.

**Parameters**

`mode` – The current operational mode setting.

**Returns**

Returns common entity return values

`aErr getOperationalState (uint32_t *state)`

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

**Parameters**

`state` – The current operational state, hardware configuration, faults, and operating mode.

**Returns**

Returns common entity return values

`aErr clearFaults (void)`

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

**Returns**

Returns common entity return values

### 3.3.18 RCservo Class

class `RCservoClass` : public Acroname::BrainStem::*EntityClass*

MARK: RCservo Class.

*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

### 3.3.19 Relay Class

```
class RelayClass : public Acroname::BrainStem::EntityClass
```

MARK: Relay Class.

*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

### 3.3.20 Signal Class

See the *Signal Entity* for generic information.

```
class SignalClass : public Acroname::BrainStem::EntityClass
```

MARK: Signal Class.

*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

### 3.3.21 Store Class

```
class StoreClass : public Acroname::BrainStem::EntityClass
```

MARK: Store Class.

*StoreClass*. The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for

details on which specific stores are implemented, if any, and the capacities of implemented stores.

## Public Functions

**StoreClass** (void)

Constructor.

**~StoreClass** (void)

Destructor.

**void init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

### Parameters

- **pModule** – The module.
- **index** – The index.

*aErr* **getSlotState** (const uint8\_t slot, uint8\_t \*state)

Get slot state.

### Parameters

- **slot** – The slot number.
- **state** – true: enabled, false: disabled.

### Returns

Returns common entity return values

*aErr* **loadSlot** (const uint8\_t slot, const uint8\_t \*pData, const uint16\_t length)

Load the slot.

### Parameters

- **slot** – The slot number.
- **pData** – The data.
- **length** – The data length.

### Returns

Returns common entity return values

*aErr* **unloadSlot** (const uint8\_t slot, const size\_t dataLength, uint8\_t \*pData, size\_t \*unloadedLength)

Unload the slot data.

### Parameters

- **pData** – Byte array that the unloaded data will be placed into.
- **dataLength** – The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than dataLength.
- **slot** – The slot number.

### Returns

Returns common entity return values

*aErr* **slotEnable** (const uint8\_t slot)

Enable slot.

### Parameters

- **slot** – The slot number.

### Returns

Returns common entity return values

`aErr slotDisable (const uint8_t slot)`

Disable slot.

**Parameters**

- `slot` – The slot number.

**Returns**

Returns common entity return values

`aErr getSlotCapacity (const uint8_t slot, size_t *capacity)`

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold.

**Parameters**

- `slot` – The slot number.
- `capacity` – The slot capacity.

**Returns**

Returns common entity return values

`aErr getSlotSize (const uint8_t slot, size_t *size)`

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes.

**Parameters**

- `slot` – The slot number.
- `size` – The slot size.

**Returns**

Returns common entity return values

`aErr getSlotLocked (const uint8_t slot, uint8_t *lock)`

Gets the current lock state of the slot Allows for write protection on a slot.

**Parameters**

- `slot` – The slot number
- `lock` – Variable to be 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

### 3.3.22 System Class

```
class SystemClass : public Acroname::BrainStem::EntityClass
```

MARK: System Class.

`SystemClass`. The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc. The most common brainstem example uses the system entity to blink the User LED.

## Public Functions

**SystemClass** (void)

Constructor.

virtual ~**SystemClass** (void)

Destructor.

void **init** (*Module* \*pModule, const uint8\_t index)

Initialize the aSystem class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – Not used; always 0.

*aErr* **getModule** (uint8\_t \*address)

Get the current address the module uses on the BrainStem network.

**Parameters**

- **address** – The address the module is using on the BrainStem network.

**Returns**

Returns common entity return values

*aErr* **getModuleBaseAddress** (uint8\_t \*address)

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

**Parameters**

- **address** – The address the module is using on the BrainStem network.

**Returns**

Returns common entity return values

*aErr* **setRouter** (const uint8\_t address)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

**Parameters**

- **address** – The router address to be used.

**Returns**

Returns common entity return values

*aErr* **getRouter** (uint8\_t \*address)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

**Parameters**

- **address** – The address.

**Returns**

Returns common entity return values

*aErr* **setHBInterval** (const uint8\_t interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these heartbeats 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 heartbeats are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

**Parameters**

**interval** – The current heartbeat delay.

**Returns**

Returns common entity return values

*aErr* **setLED** (const uint8\_t bOn)

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

**Parameters**

**bOn** – true: turn the LED on, false: turn LED off.

**Returns**

Returns common entity return values

*aErr* **getLED** (uint8\_t \*bOn)

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

**Parameters**

**bOn** – true: LED on, false: LED off.

**Returns**

Returns common entity return values

*aErr* **setBootSlot** (const uint8\_t slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

**Parameters**

**slot** – The slot number in aSTORE\_INTERNAL to be marked as a boot slot.

**Returns**

Returns common entity return values

*aErr* **getBootSlot** (uint8\_t \*slot)

Get the store slot which is mapped when the module boots.

**Parameters**

**slot** – The slot number in aSTORE\_INTERNAL that is mapped after the module boots.

**Returns**

Returns common entity return values

*aErr* **getVersion** (uint32\_t \*build)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

**Parameters**

**build** – The build version date code.

*aErr* **getModel** (uint8\_t \*model)

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem model codes". Other codes are used by Acroname for proprietary module types.

**Parameters**

`model` – The module's model enumeration.

**Returns**

Returns common entity return values

`aErr getHardwareVersion (uint32_t *hardwareVersion)`

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

**Parameters**

`hardwareVersion` – The module's hardware version information.

**Returns**

Returns common entity return values

`aErr getSerialNumber (uint32_t *serialNumber)`

Get the module's serial number. The serial number is a unique 32bit integer which is usually communicated in hexadecimal format.

**Parameters**

`serialNumber` – The module's serial number.

**Returns**

Returns common entity return values

`aErr save (void)`

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address

**Returns**

Returns common entity return values

`aErr reset (void)`

Reset the system.

**Returns**

Returns common entity return values

`aErr logEvents (void)`

Saves system log events to a slot defined by the module (usually ram slot 0).

**Returns**

Returns common entity return values

`aErr getUptime (uint32_t *uptimeCounter)`

Get the module's accumulated uptime in minutes

**Parameters**

`uptimeCounter` – The module's accumulated uptime in minutes.

**Returns**

Returns common entity return values

`aErr getTemperature (int32_t *temperature)`

Get the module's current temperature in micro-C

**Parameters**

`temperature` – The module's system temperature in micro-C

**Returns**

Returns common entity return values

`aErr getMinimumTemperature (int32_t *minTemperature)`

Get the module's minimum temperature ever recorded in micro-C (uC) This value will 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 (uint8_t *buffer, const size_t bufLength, size_t *unloadedLength)`

Gets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

- `buffer` – pointer to the start of a c style buffer to be filled
- `bufLength` – Length of the buffer to be filed
- `unloadedLength` – Length that was actually received and filled.

**Returns**

Returns common entity return values

`aErr setInputPowerBehaviorConfig (uint32_t *buffer, const size_t bufLength)`

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

- `buffer` – Pointer to the start of a c style buffer to be transferred.
- `bufLength` – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

`aErr getName (uint8_t *buffer, const size_t bufLength, size_t *unloadedLength)`

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

**Parameters**

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – 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 bufLength)

Sets a user defined name for the device. Helpful for identification when multiple devices of the same type are present in a system.

**Parameters**

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

*aErr* **resetEntityToFactoryDefaults** (void)

Resets the *SystemClass* Entity to it factory default configuration.

**Returns**

Returns common entity return values

*aErr* **resetDeviceToFactoryDefaults** (void)

Resets the device to it factory default configuration.

**Returns**

Returns common entity return values

*aErr* **getLinkInterface** (uint8\_t \*linkInterface)

Gets the link interface configuration. This refers to which interface is being used for control by the device.

**Parameters**

- linkInterface** – Variable to be filled with an enumerated value representing interface.
- 0 = Auto= systemLinkAuto
  - 1 = Control Port = systemLinkUSBControl
  - 2 = Hub Upstream Port = systemLinkUSBHub

**Returns**

Returns common entity return values

*aErr* **setLinkInterface** (const uint8\_t linkInterface)

Sets the link interface configuration. This refers to which interface is being used for control by the device.

**Parameters**

- linkInterface** – An enumerated representation of interface to be set.
- 0 = Auto= systemLinkAuto
  - 1 = Control Port = systemLinkUSBControl
  - 2 = Hub Upstream Port = systemLinkUSBHub

**Returns**

Returns common entity return values

*aErr* **getErrors** (uint32\_t \*errors)

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

**Parameters**

- errors** – Bit mapped field representing the devices errors

**Returns**

Returns common entity return values

### 3.3.23 Temperature Class

`class TemperatureClass : public Acroname::BrainStem::EntityClass`

MARK: Temperature Class.

*TemperatureClass*. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

#### Public Functions

`TemperatureClass (void)`

Constructor.

`~TemperatureClass (void)`

Destructor.

`void init (Module *pModule, const uint8_t index)`

Initialize the class.

**Parameters**

- `pModule` – The module to which this entity belongs.
- `index` – The index of the entity.

`aErr getValue (int32_t *temp)`

Get the modules temperature in micro-C

**Parameters**

- `temp` – The temperature in micro-Celsius (1 == 1e-6C).

**Returns**

Returns common entity return values

`aErr getValueMin (int32_t *minTemp)`

Get the module's minimum temperature in micro-C since the last power cycle.

**Parameters**

- `minTemp` – The module's minimum temperature in micro-C

**Returns**

Returns common entity return values

`aErr getValueMax (int32_t *maxTemp)`

Get the module's maximum temperature in micro-C since the last power cycle.

**Parameters**

- `maxTemp` – The module's maximum temperature in micro-C

**Returns**

Returns common entity return values

`aErr resetEntityToFactoryDefaults (void)`

Resets the *TemperatureClass* Entity to it factory default configuration.

### 3.3.24 Timer Class

```
class TimerClass : public Acroname::BrainStem::EntityClass
```

MARK: Timer Class.

*TimerClass*. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

#### Public Functions

```
TimerClass (void)
```

Constructor.

```
~TimerClass (void)
```

Destructor.

```
void init (Module *pModule, const uint8_t index)
```

Initialize the class.

##### Parameters

- **pModule** – The module to which this entity belongs.
- **index** – The index of the timer entity.

```
aErr getExpiration (uint32_t *usecDuration)
```

Get the currently set expiration time in microseconds. This is not a “live” timer. That is, it shows the expiration time originally set with setExpiration; it does not “tick down” to show the time remaining before expiration.

##### Parameters

**usecDuration** – The timer expiration duration in microseconds.

##### Returns

Returns common entity return values

```
aErr setExpiration (const uint32_t usecDuration)
```

Set the expiration time for the timer entity. When the timer expires, it will fire the associated timer[index]() reflex.

##### Parameters

**usecDuration** – The duration before timer expiration in microseconds.

##### Returns

Returns common entity return values

```
aErr getMode (uint8_t *mode)
```

Get the mode of the timer which is either single or repeat mode.

##### Parameters

**mode** – The mode of the timer. aTIMER\_MODE\_REPEAT or aTIMER\_MODE\_SINGLE.

##### Returns

Returns common entity return values

```
aErr setMode (const uint8_t mode)
```

Set the mode of the timer which is either single or repeat mode.

##### Parameters

**mode** – The mode of the timer. aTIMER\_MODE\_REPEAT or aTIMER\_MODE\_SINGLE.

**Returns**

Returns common entity return values

**Returns**

*aErrNone* - Action completed successfully.

### 3.3.25 UART Class

```
class UARTClass : public Acroname::BrainStem::EntityClass
```

MARK: UART Class.

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

### 3.3.26 USB Class

class **USBClass** : public Acroname::BrainStem::*EntityClass*

MARK: USB Class.

*USBClass*. The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

#### Public Functions

**USBClass** (void)

Constructor.

**~USBClass** (void)

Destructor.

**void init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity, i.e. the port

*aErr setPortEnable* (const uint8\_t channel)

Enable both power and data lines for a port.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr setPortDisable* (const uint8\_t channel)

Disable both power and data lines for a port.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setDataEnable (const uint8_t channel)`

Enable the only the data lines for a port without changing the state of the power line.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setDataDisable (const uint8_t channel)`

Disable only the data lines for a port without changing the state of the power line.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setHiSpeedDataEnable (const uint8_t channel)`

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setHiSpeedDataDisable (const uint8_t channel)`

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setSuperSpeedDataEnable (const uint8_t channel)`

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setSuperSpeedDataDisable (const uint8_t channel)`

Disable only the data lines for a port without changing the state of the power line, Super-Speed (3.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setPowerEnable (const uint8_t channel)`

Enable only the power line for a port without changing the state of the data lines.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

`aErr setPowerDisable (const uint8_t channel)`

Disable only the power line for a port without changing the state of the data lines.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **getPortCurrent** (const uint8\_t channel, int32\_t \*microamps)

Get the current through the power line for a port.

**Parameters**

- channel – The USB sub channel.

- microamps – The USB channel current in micro-amps (1 == 1e-6A).

**Returns**

Returns common entity return values

*aErr* **getPortVoltage** (const uint8\_t channel, int32\_t \*microvolts)

Get the voltage on the power line for a port.

**Parameters**

- channel – The USB sub channel.

- microvolts – The USB channel voltage in microvolts (1 == 1e-6V).

**Returns**

Returns common entity return values

*aErr* **getHubMode** (uint32\_t \*mode)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

**Parameters**

mode – The USB hub mode.

**Returns**

Returns common entity return values

*aErr* **setHubMode** (const uint32\_t mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

**Parameters**

mode – The USB hub mode.

**Returns**

Returns common entity return values

*aErr* **clearPortErrorStatus** (const uint8\_t channel)

Clear the error status for the given port.

**Parameters**

channel – The port to clear error status for.

**Returns**

Returns common entity return values

*aErr* **setUpstreamMode** (uint8\_t \*mode)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

**Parameters**

mode – The Upstream port mode.

**Returns**

Returns common entity return values

*aErr* **setUpstreamMode** (const uint8\_t mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0 and usbUpstreamModePort1

**Parameters**

mode – The Upstream port mode.

**Returns**

Returns common entity return values

*aErr* **getUpstreamState** (uint8\_t \*state)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

**Parameters**

**state** – The Upstream port state.

**Returns**

Returns common entity return values

*aErr* **setEnumerationDelay** (const uint32\_t ms\_delay)

Set the inter-port enumeration delay in milliseconds.

**Parameters**

**ms\_delay** – Millisecond delay in 100mS increments (100, 200, 300 etc.)

**Returns**

Returns common entity return values

*aErr* **getEnumerationDelay** (uint32\_t \*ms\_delay)

Get the inter-port enumeration delay in milliseconds.

**Parameters**

**ms\_delay** – Millisecond delay in 100mS increments (100, 200, 300 etc.)

**Returns**

Returns common entity return values

*aErr* **setPortCurrentLimit** (const uint8\_t channel, const uint32\_t microamps)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a `stem.system.save()` call.

**Parameters**

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

**Returns**

Returns common entity return values

*aErr* **getPortCurrentLimit** (const uint8\_t channel, uint32\_t \*microamps)

Get the current limit for the port.

**Parameters**

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

**Returns**

Returns common entity return values

*aErr* **setPortMode** (const uint8\_t channel, const uint32\_t mode)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device datasheet for a complete list of capabilities. Some devices user 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 packet bit mask.

**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 datasheet for a complete list of capabilities. Some devices implement a common bit mapping for port mode at `usbPortMode`

**Parameters**

- `channel` – USB downstream channel.
- `mode` – The port mode setting. Mode will be filled with the current setting.  
Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

`aErr getPortState (const uint8_t channel, uint32_t *state)`

Get the current State for the Port.

**Parameters**

- `channel` – USB downstream channel.
- `state` – The port mode setting. Mode will be filled with the current setting.  
Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

`aErr getPortError (const uint8_t channel, uint32_t *error)`

Get the current error for the Port.

**Parameters**

- `channel` – USB downstream channel.
- `error` – The port mode setting. Mode will be filled with the current setting.  
Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

`aErr setUpstreamBoostMode (const uint8_t setting)`

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

**Parameters**

- `setting` – Upstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

`aErr setDownstreamBoostMode (const uint8_t setting)`

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

**Parameters**

- `setting` – Downstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

`aErr getUpstreamBoostMode (uint8_t *setting)`

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

**Parameters**

`setting` – The current Upstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

`aErr getDownstreamBoostMode (uint8_t *setting)`

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

**Parameters**

`setting` – The current Downstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

`aErr getDownstreamDataSpeed (const uint8_t channel, uint8_t *speed)`

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

**Parameters**

- `channel` – USB downstream channel to check.
- `speed` – Filled with the current port data speed
  - N/A: `usbDownstreamDataSpeed_na` = 0
  - Hi Speed: `usbDownstreamDataSpeed_hs` = 1
  - SuperSpeed: `usbDownstreamDataSpeed_ss` = 2

**Returns**

Returns common entity return values

`aErr setConnectMode (const uint8_t channel, const uint8_t mode)`

Sets the connect mode of the switch.

**Parameters**

- `channel` – The USB sub channel.
- `mode` – The connect mode
  - `usbManualConnect` = 0
  - `usbAutoConnect` = 1

**Returns**

Returns common entity return values

`aErr getConnectMode (const uint8_t channel, uint8_t *mode)`

Gets the connect mode of the switch.

**Parameters**

- `channel` – The USB sub channel.
- `mode` – The current connect mode

**Returns**

Returns common entity return values

`aErr setCC1Enable (const uint8_t channel, const uint8_t bEnable)`

Set Enable/Disable on the CC1 line.

**Parameters**

- `channel` – USB channel.
- `bEnable` –
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

*aErr* **getCC1Enable** (const uint8\_t channel, uint8\_t \*pEnable)

Get Enable/Disable on the CC1 line.

**Parameters**

- **channel** -- USB channel.
- **pEnable** -
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

*aErr* **setCC2Enable** (const uint8\_t channel, const uint8\_t bEnable)

Set Enable/Disable on the CC2 line.

**Parameters**

- **channel** -- USB channel.
- **bEnable** -
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

*aErr* **getCC2Enable** (const uint8\_t channel, uint8\_t \*pEnable)

Get Enable/Disable on the CC1 line.

**Parameters**

- **channel** -- USB channel.
- **pEnable** -
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

*aErr* **getCC1Current** (const uint8\_t channel, int32\_t \*microamps)

Get the current through the CC1 for a port.

**Parameters**

- **channel** - The USB sub channel.
- **microamps** - The USB channel current in micro-amps (1 == 1e-6A).

**Returns**

Returns common entity return values

*aErr* **getCC2Current** (const uint8\_t channel, int32\_t \*microamps)

Get the current through the CC2 for a port.

**Parameters**

- **channel** - The USB sub channel.
- **microamps** - The USB channel current in micro-amps (1 == 1e-6A).

**Returns**

Returns common entity return values

*aErr* **getCC1Voltage** (const uint8\_t channel, int32\_t \*microvolts)

Get the voltage of CC1 for a port.

**Parameters**

- **channel** - The USB sub channel.
- **microvolts** - The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

*aErr* **getCC2Voltage** (const uint8\_t channel, int32\_t \*microvolts)

Get the voltage of CC2 for a port.

**Parameters**

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

*aErr* **setSBUEnable** (const uint8\_t channel, const uint8\_t bEnable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

**Parameters**

- **channel** – The USB sub channel.
- **bEnable** –
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

*aErr* **getSBUEnable** (const uint8\_t channel, uint8\_t \*pEnable)

Get the Enable/Disable status of the SBU

**Parameters**

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of the SBU

**Returns**

Returns common entity return values

*aErr* **setCableFlip** (const uint8\_t channel, const uint8\_t bEnable)

Set Cable flip. This will flip SBU, CC and SS data lines.

**Parameters**

- **channel** – The USB sub channel.
- **bEnable** –
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

*aErr* **getCableFlip** (const uint8\_t channel, uint8\_t \*pEnable)

Get Cable flip setting.

**Parameters**

- **channel** – The USB sub channel.
- **pEnable** – The enable/disable status of cable flip.

**Returns**

Returns common entity return values

*aErr* **setAltModeConfig** (const uint8\_t channel, const uint32\_t configuration)

Set USB Alt Mode Configuration.

**Parameters**

- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

**Returns**

Returns common entity return values

*aErr* **getAltModeConfig** (const uint8\_t channel, uint32\_t \*configuration)

Get USB Alt Mode Configuration.

**Parameters**

- **channel1** – 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**

- **channel1** – 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**

- **channel1** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

### 3.3.27 USBSYSTEM Class

class **USBSYSTEMClass** : public Acroname::BrainStem::*EntityClass*

MARK: USB System Class.

USBSYSTEM Class The USBSYSTEM class provides high level control of the lower level Port Class.

Subclassed by *aUSBHub3c::HubClass*

#### Public Functions

**USBSYSTEMClass** (void)

Constructor.

**~USBSYSTEMClass** (void)

Destructor.

**void init** (*Module* \*pModule, const uint8\_t index)

Initialize the class.

**Parameters**

- **pModule** – The module to which this entity belongs.
- **index** – The index of the entity, i.e. the port

*aErr* **setUpstream** (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 `Port-Class::getDataRole()` on each individual port.

**Parameters**

`roleList` – A bit packed representation of the data role for all ports.

**Returns**

Returns common entity return values

`aErr getEnabledList (uint32_t *enabledList)`

Gets the current enabled status of all ports with a single call. Equivalent to calling `Port-Class::setEnabled()` on each port.

**Parameters**

`enabledList` – Bit packed representation of the enabled status for all ports.

**Returns**

Returns common entity return values

`aErr setEnabledList (const uint32_t enabledList)`

Sets the enabled status of all ports with a single call. Equivalent to calling `Port-Class::setEnabled()` on each port.

**Parameters**

`enabledList` – Bit packed representation of the enabled status for all ports to be applied.

**Returns**

Returns common entity return values

`aErr getModeList (uint32_t *buffer, const size_t bufLength, size_t *unloadedLength)`

Gets the current mode of all ports with a single call. Equivalent to calling `Port-Class::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** (uint32\_t \*buffer, const size\_t bufLength)

Sets the mode of all ports with a single call. Equivalent to calling *PortClass::setMode()* on each port

**Parameters**

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

*aErr* **getStateList** (uint32\_t \*buffer, const size\_t bufLength, size\_t \*unloadedLength)

Gets the state for all ports with a single call. Equivalent to calling *PortClass::getState()* on each port.

**Parameters**

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be 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 bufLength, size\_t \*unloadedLength)

Gets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

*aErr* **setPowerBehaviorConfig** (uint32\_t \*buffer, const size\_t bufLength)

Sets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

*aErr* **getDataRoleBehavior** (uint8\_t \*behavior)

Gets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

**Parameters**

- behavior** – Variable to be filled with an enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **setDataRoleBehavior** (const uint8\_t behavior)

Sets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

**Parameters**

- behavior** – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **getDataRoleBehaviorConfig** (uint32\_t \*buffer, const size\_t bufLength, size\_t \*unloadedLength)

Gets the current data role behavior configuration Certain data role behaviors use a list of ports to determine priority host priority.

**Parameters**

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filled
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

*aErr* **setDataRoleBehaviorConfig** (uint32\_t \*buffer, const size\_t bufLength)

Sets the current data role behavior configuration Certain data role behaviors use a list of ports to determine host priority.

**Parameters**

- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

*aErr* **getSelectorMode** (uint8\_t \*mode)

Gets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

**Parameters**

- mode** – Variable to be filled with the selector mode

**Returns**

Returns common entity return values

*aErr* **setSelectorMode** (const uint8\_t mode)

Sets the current mode of the selector input. This mode determines what happens and in what order when the external selector input is used.

**Parameters**

- mode** – Mode to be set.

**Returns**

Returns common entity return values

*aErr* **resetEntityToFactoryDefaults** (void)

Resets the *USBSystemClass* Entity to it factory default configuration.

## 3.4 C API Reference

This section of the [Acroname BrainStem Reference](#) covers the lower level C programming interface. This reference assumes that you understand the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation.

- [BrainStem Overview](#)
- [BrainStem Terminology](#)
- [Getting Started with the BrainStem](#).

Most users will want to begin with the C++ API or Python libraries, but for some specialized applications, or applications that must be ANSI C only, the C API provides access a little closer to the metal than the other APIs. To use the C api effectivley you will need to understand a little more about the BrainStem protocol, and particularly about UEI's and their significance.

- [Appendix: Brainstem Universal Entity Interface](#)
  - [Appendix: The BrainStem Communication Protocol](#)
- 

### 3.4.1 Modules

#### group **aDefs**

Acroname Specific Universal Defines and includes.

The C-Interface requires some specific defines for cross platform compatibility. The [aDefs.h](#) file contains those defines and includes that are necessary at a global level across platforms.

Things like a cross platform way to specify line endings, safe c-string copy and concatenation operations, and boolean typedefs when they are not defined by default.

We rely on the following std headers:

- assert.h
- stddef.h
- stdint.h
- stdbool.h
- string.h
- stdio.h
- stdlib.h

#### group **aDiscovery**

Link Discovery Interface.

[aDiscovery.h](#) provides an interface for locating BrainStem modules accross multiple transports. It provides a way to find all modules for a give transport as well as specific modules by serial number, or first found.

#### group **aErrors**

Unified list of Error codes for BrainStem module Interation.

aError.h provides a unified list of error codes. These error codes apply accross all API's. Library functions will return one of these error codes when appropriate.

**group aFile**

Platform Independent File Access Interface.

*aFile.h* provides a platform independent interface for opening, reading, and writing files.

**group aLink**

BrainStem Link Interface.

*aLink.h* provides the interface for creating and maintaining the link to a BrainStem module, and the BrainStem network. It includes facilities for starting and stopping links, as well as sending and receiving BrainStem protocol packets.

**group aMutex**

Platform Independent Synchronization Primitive.

*aMutex.h* Provides a platform independent synchronization mechanism. The link interface and the packer fifos both use this interface for synchronization between threads. Includes facilities for creating, locking and unlocking mutex primitives.

**group aPacket**

BrainStem Packet.

*aPacket.h* Provides an interface for creating and destroying BrainStem Protocol packets.

**group aProtocoldefs**

BrainStem Protocol Definitions.

*aProtocoldefs.h* Provides protocol and BrainStem specific defines for entities, communication, and protocol specifics.

**group aStream**

Platform Independent Stream Abstraction.

*aStream.h* provides a platform independent stream abstraction for common I/O streams. Provides facilities for creating and destroying as well as writing and reading from streams.

**group aTime**

Basic Time procedures Sleep and Get process tics.

*aTime.h* provides a platform independent interface for millisecond sleep, and for getting process tics.

**group aVersion**

Library version interface.

*aVersion.h* Provides version information for the BrainStem2 library.

**group aUEI**

UEI Utilities.

*aUEI.h* Provides structs and utilities for working with UEIs.

### 3.4.2 aDefs.h

#### group aDefs

Acroname Specific Universal Defines and includes.

The C-Interface requires some specific defines for cross platform compatibility. The *aDefs.h* file contains those defines and includes that are necessary at a global level across platforms.

Things like a cross platform way to specify line endings, safe c-string copy and concatenation operations, and boolean typedefs when they are not defined by default.

We rely on the following std headers:

- assert.h
- stddef.h
- stdint.h
- stdbool.h
- string.h
- stdio.h
- stdlib.h

**aSHOWERR** (msg, error) (printf("Error in File; %s on line; %d, %s: %d\n", \_\_FILE\_\_, \_\_LINE\_\_, msg, error))

A macro that will emit an error on stdout including file and line number when compiled without NDEBUG flag. When NDEBUG is defined it emits nothing.

**os\_NEW\_LN** "\n"

A macro containing the appropriate line ending characters for a given platform \r\n on windows and \n elsewhere.

**aStringCopySafe** (d, l, s) strncpy((d), (s), (l))

A macro that maps to platform specific safe string copy. Parameters are;

- d: destination
- l: length
- s: source

**aStringCatSafe** (d, l, s) strncat((d), (s), (l))

A macro that maps to platform specific safe string concatenation. Parameters are;

- d: destination
- l: length
- s: source

**aSNPRINTF** snprintf

A macro that maps to the platform specific safe printf output.

```
aLIBEXPORT __attribute__((visibility ("default")))
```

A macro that expands for dynamic library linking on a given platform.

```
aMemPtr void *
```

An acronym specific semantic define for a pointer to a chunk of memory.

```
const char *aDefs_GetModelName (const int modelNum)
```

Returns a printable model string.

### 3.4.3 aDiscovery.h

```
group aDiscovery
```

Link Discovery Interface.

*aDiscovery.h* provides an interface for locating BrainStem modules accross multiple transports. It provides a way to find all modules for a give transport as well as specific modules by serial number, or first found.

```
enum linkType
```

Enum *linkType*.

The linkType enum specifies the connection transport type.

*Values*:

```
enumerator INVALID
```

- Undefined link type.

```
enumerator USB
```

- USB link type.

```
enumerator TCP_IP
```

- TCP/IP link type.

```
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.

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)
```

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.

#### Returns

Returns the number of modules found.

```
linkSpec *aDiscovery_FindModule (const linkType type, uint32_t serialNum)
```

Function `aDiscovery_FindModule`.

Finds the module with the given serial number on the given transport type.

#### Parameters

- **type** – The trasport type on which search for devices. Valid `linkType` “linktypes” are accepted
- **serialNum** – The serial number of the Module to find.

#### Returns

A pointer to the `linkSpec` for the requested module if found or NULL otherwise. This call Allocates memory that must be freed by a call to `aLinkSpec_Destroy`.

```
linkSpec *aDiscovery_FindFirstModule (const linkType type)
```

Function `aDiscovery_FindFirstModule`.

Finds the first module found on the given transport.

#### Parameters

`type` – The transport type on which search for devices. Valid `linkType` “linktypes” are accepted

#### Returns

A pointer to the `linkSpec` for the requested module if found or NULL otherwise. This call Allocates memory that must be freed by a call to `aLinkSpec_Destroy`.

## LinkSpec Functions

`linkSpec *aLinkSpec_Create (const linkType type)`

Function `aLinkSpec_Create`.

Creates a `linkSpec` object with transport set to the given type.

### Parameters

`type` – The transport type on which search for devices. Valid `linkType` “linktypes” are accepted

### Returns

A pointer to the `linkSpec` for the requested module or NULL if there was an error allocating memory. This call Allocates memory that must be freed by a call to `aLinkSpec_Destroy`.

`aErr aLinkSpec_Destroy (linkSpec **spec)`

Function `aLinkSpec_Destroy`.

Destroys and clears the referenced `linkSpec`.

### Parameters

`spec` – A pointer to the `linkSpec` pointer previously allocated.

### Returns

`aErrNone` on success or an error if there was an error encountered deallocating the `linkSpec`.

## 3.4.4 Error Codes

### enum aErr

The `aErr` enum lists the possible error codes for library calls. BrainStem commands generally return a set of unified Error codes. The API tries to be consistent and return these errors from every interaction with the stem.

*Values:*

#### enumerator aErrNone

0 - Success, no error.

#### enumerator aErrMemory

1 - Memory allocation.

#### enumerator aErrParam

2 - Invalid parameter.

#### enumerator aErrNotFound

3 - Not found.

#### enumerator aErrFileNameLength

4 - File name too long.

enumerator **aErrBusy**

5 - Resource busy.

enumerator **aErrIO**

6 - Input/Output error.

enumerator **aErrMode**

7 - Invalid Mode.

enumerator **aErrWrite**

8 - Write error.

enumerator **aErrRead**

9 - Read error.

enumerator **aErrEOF**

10 - End of file.

enumerator **aErrNotReady**

11 - Not ready, no bytes available.

enumerator **aErrPermission**

12 - Insufficient permissions.

enumerator **aErrRange**

13 - Value out of range.

enumerator **aErrSize**

14 - Invalid Size.

enumerator **aErrOverrun**

15 - Buffer/queue overrun.

enumerator **aErrParse**

16 - Parse error.

enumerator **aErrConfiguration**

17 - Configuration error.

enumerator **aErrTimeout**

18 - Timeout occurred.

enumerator **aErrInitialization**

19 - Initialization error.

enumerator **aErrVersion**

20 - Invalid version.

enumerator **aErrUnimplemented**

21 - Functionality unimplemented.

enumerator **aErrDuplicate**

22 - Duplicate request.

enumerator **aErrCancel**

23 - 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 too short.

enumerator **aErrInvalidEntity**

28 - Invalid entity error.

enumerator **aErrInvalidOption**

29 - Invalid option code.

enumerator **aErrResource**

30 - Resource unavailable.

enumerator **aErrMedia**

31 - Media error.

enumerator **aErrAsyncReturn**

32 - Asynchronous return.

enumerator **aErrStreamStale**

33 - Stream value is stale.

enumerator **aErrUnknown**

34 - Unknown error.

const char \***aError\_GetErrorText** (*aErr* err)

Returns a printable error string.

### 3.4.5 aFile.h

#### group aFile

Platform Independent File Access Interface.

*aFile.h* provides a platform independent interface for opening, reading, and writing files.

**typedef void \*aFileRef**

Typedef *aFileRef* Opaque reference to a file handle.

#### enum a FileMode

Enum *a FileMode*.

Represents whether the file is to be opened in read or write mode.

*Values:*

enumerator **a FileModeReadOnly**

File read mode.

enumerator **a FileModeWriteOnly**

File write mode.

enumerator **a FileModeAppend**

File write mode from end of current file.

enumerator **a FileModeUnknown**

File in unknown mode.

#### enum a FileSeekMode

Enum *a FileSeekMode*.

Represents the seek start location.

*Values:*

enumerator **a SeekStart**

Perform a seek from the beginning of the file.

enumerator **a SeekCurrent**

Perform a seek from the current location.

enumerator **a SeekEnd**

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 *a FileMode* eMode)

Open a File.

Opens the file Given in pFilename with the given fileMode eMode.

**Parameters**

- **pFilename** – path to file.
- **eMode** – Open the file for Reading or Writing.

**Returns**

*aFileRef* on success or NULL on failure.

*aErr* **aFile\_Close** (*aFileRef* \*fileRef)

Close an open file.

Close an open file. The fileRef is set to NULL on success.

**Parameters**

**fileRef** – Pointer to the handle to the open file.

**Return values**

- **aErrNone** – Success.
- **aErrParam** – invalid file reference.

**Returns**

Function returns aErr values.

*aErr* **aFile\_Read** (*aFileRef* fileRef, uint8\_t \*pBuffer, const size\_t nLength, size\_t \*pActuallyRead)

Read from an open file.

Read from an open file.

**Parameters**

- **fileRef** – The handle to the open file.
- **pBuffer** – The data buffer to read into.
- **nLength** – The length of the read buffer.
- **pActuallyRead** – The Number of bytes actually read from the file.

**Return values**

- **aErrNone** – Success.
- **aErrMode** – The file is not readable.
- **aErrIO** – An error occurred reading from the file.
- **aErrEOF** – Read reached the end of the file.

**Returns**

Function returns aErr values.

*aErr* **aFile\_Write** (*aFileRef* fileRef, const uint8\_t \*pBuffer, const size\_t nLength, size\_t \*pActuallyWritten)

Write to an open file.

Write to an open file.

**Parameters**

- **fileRef** – The handle to the open file.
- **pBuffer** – The data to write.
- **nLength** – The length of the data to write.
- **pActuallyWritten** – The Number of bytes actually written to the file.

**Return values**

- **aErrNone** – Success.
- **aErrMode** – The file is not writable.
- **aErrIO** – An error occurred writing to the file.

**Returns**

Function returns aErr values.

*aErr aFile\_Seek (aFileRef fileRef, const long nOffset, aFileSeekMode seekFrom)*

Seek within an open file.

Seek within an open file.

**Parameters**

- **fileRef** – The handle to the open file.
- **nOffset** – The number of bytes to move within the file.
- **seekFrom** – The location to begin the seek from.

**Return values**

- **aErrNone** – Success.
- **aErrEOF** – Seek would run off the end of the file.
- **aErrRange** – Seek would run off the beginning of the file.
- **aErrIO** – An error occurred moving the file pointer.

**Returns**

Function returns aErr values.

*aErr aFile\_GetSize (aFileRef fileRef, size\_t \*pulSize)*

Get the size of an open file.

Get the size of an open file.

**Parameters**

- **fileRef** – The handle to the open file.
- **pulSize** – Out param filled with the size of the open file.

**Return values**

- **aErrNone** – Success.
- **aErrParam** – the fileRef is invalid.
- **aErrIO** – an error occurred calculating the size.

**Returns**

Function returns aErr values.

`aErr aFile_Delete (const char *pFilename)`

Delete a File.

Deletes the given file pFilename.

#### Parameters

`pFilename` – Path to file.

#### Return values

- `aErrNone` – Success.
- `aErrPermission` – user has insufficient priviledges.
- `aErrNotFound` – if the file cannot be located.

#### Returns

Function returns aErr values.

### 3.4.6 aLink.h

*group aLink*

BrainStem Link Interface.

`aLink.h` provides the interface for creating and maintaining the link to a BrainStem module, and the BrainStem network. It includes facilities for starting and stopping links, as well as sending and receiving BrainStem protocol packets.

`typedef uint32_t aLinkRef`

Typedef `aLinkRef` Opaque reference to a BrainStem link.

Typedef for aLinkRef for an opaque reference to BrainStem Link.

`enum linkStatus`

Represents the current state of the BrainStem link.

*Values:*

`enumerator STOPPED`

Link currently stopped.

`enumerator INITIALIZING`

Starting communication with module.

`enumerator RUNNING`

Link running.

`enumerator STOPPING`

Link is in the process of stopping.

`enumerator SYNCING`

Packet framing lost re-syncing.

enumerator **INVALID\_LINK\_STREAM**

Link stream provided is not valid.

enumerator **IO\_ERROR**

Communication error occurred on link, could not resync.

enumerator **RESETTING**

Resetting the link connection

enumerator **UNKNOWN\_ERROR**

Something really bad happened, but we couldn't determine what.

*aLinkRef* **aLink\_CreateUSB** (const uint32\_t serialNumber)

Create a BrainStem link reference.

Creates a reference to a BrainStem link. The linkStream is now maintained by the BrainStem link. If the link already exists, the use count for that link will be incremented and the linkRef for that entry will be returned.

Links created with this procedure must use aLink\_Destroy to properly dispose of the link reference and associated connections.

**Parameters**

**serialNumber** – the TCPIP address

**Returns**

aLinkRef identifier if successful or 0 otherwise.

*aLinkRef* **aLink\_CreateTCPIP** (const uint32\_t address, const uint16\_t port)

Creates a reference to a BrainStem link. The linkStream is now maintained by the BrainStem link. If the link already exists, the use count for that link will be incremented and the linkRef for that entry will be returned.

Links created with this procedure must use aLink\_Destroy to properly dispose of the link reference and associated connections.

**Parameters**

- **address** – the TCPIP address
- **port** – the TCPIP port

**Returns**

aLinkRef identifier if successful or 0 otherwise.

*aErr* **aLink\_Destroy** (*aLinkRef* \*linkRef)

Destroy a BrainStem link reference.

Destroys a Link reference. deallocating associated resources cleanly.

Links created with aLink\_Create must use aLink\_Destroy to clean up resources used by the link Ref.

**Parameters**

**linkRef** – a Pointer to a valid LinkRef. The linkRef will be set to NULL on successful completion of the Destroy call.

**Returns**

aStreamRef Return value will always be NULL. The return value has been left for backwards compatibility.

`aErr aLink_Reset (const aLinkRef linkRef)`

Reset a connection to a BrainStem module.

Stop the active connection to the BrainStem if the Link contains a valid stream Reference, and clear out the communication buffers, and restart the link.

#### Parameters

`linkRef` – A valid LinkRef.

#### Return values

- `aErrNone` – the call completed successfully, a subsequent call to `aLink_GetStatus` should return the current state of the link.
- `aErrParam` – No valid LinkRef provided.

#### Returns

Function returns `aErr` values.

`linkStatus aLink_GetStatus (const aLinkRef linkRef)`

Return the current status of the BrainStem link.

Return the current status of the BrainStem link.

#### Parameters

`linkRef` – A valid LinkRef.

#### Returns

`linkStatus` See the possible `linkStatus` values.

`aPacket *aLink_GetPacket (const aLinkRef linkRef)`

Return the first packet in the Link 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 outgoing link FIFO.

Put a packet into the outgoing link FIFO.

**Parameters**

- **linkRef** – A valid LinkRef.
- **packet** – A BrainStem packet.

**Return values**

- **aErrNone** – Call successfully added the packet.
- **aErrParam** – Invalid LinkRef or packet.
- **aErrResource** – Unable to create memory for packet in FIFO.

**Returns**

Function returns aErr values.

### 3.4.7 aMutex.h

#### group aMutex

Platform Independent Synchronization Primitive.

[aMutex.h](#) Provides a platform independent synchronization mechanism. The link interface and the 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 occurred.

#### Return values

- **aErrNone** -- Successfully aquired the lock.
- **aErrParam** -- If the MutexRef was invalid.

#### Returns

Function returns aErr values.

**Returns**

aErrDuplicate - If a specific error occurred locking the mutex.

**aErr aMutex\_TryLock (aMutexRef mutex)**

Mutex TryLock.

Non Blocking attempt to Lock the mutex. The call will return immediately with aErrBusy if another process or thread owns the lock.

**Return values**

- **aErrNone** -- Successfully acquired the lock.
- **aErrParam** -- If the MutexRef was invalid.
- **aErrBusy** -- If the lock was already in use.

**Returns**

Function returns aErr values.

**aErr aMutex\_Unlock (aMutexRef mutex)**

Mutex Unlock.

Relinquish the lock on the mutex.

**Return values**

- **aErrNone** -- Successfully unlocked mutex.
- **aErrParam** -- If the MutexRef was invalid.
- **aErrPermission** -- If the lock is owned by another thread.

**Returns**

Function returns aErr values.

### 3.4.8 aPacket.h

**group aPacket**

BrainStem Packet.

*aPacket.h* Provides an interface for creating and destroying BrainStem Protocol packets.

const uint16\_t **VALIDPACKET**

Const value used to check packet validity.

**struct aPacket**

Struct for BrainStem packets.

The check member is for checking the validity of the packet structure in memory. Current size is used during link stream processing. Address, dataSize and data fulfill the requirements of the BrainStem protocol.

bool **aVALIDPACKET** (const *aPacket* \*packet)

Check packet pointer for validity.

Checks to make sure a packet was allocated using aPacket\_Create.

**Parameters**

*packet* -- valid packet pointer.

**Returns**

bool - True for valid false otherwise.

**aPacket Functions**

*aPacket* \***aPacket\_Create** (void)

Create a BrainStem packet.

Create a BrainStem packet.

**Returns**

*aPacket* - Pointer or NULL on error.

*aPacket* \***aPacket\_CreateWithData** (const uint8\_t address, const uint8\_t dataLength, const uint8\_t \*data)

Create a BrainStem packet, containing the given data.

Create a BrainStem packet with data.

**Parameters**

- **address** -- Module address of the BrainStem module.
- **dataLength** -- The length of the data array.
- **data** -- Pointer to the beginning of the packet data.

**Returns**

*aPacket* - Pointer or NULL on error.

*aErr* **aPacket\_Reset** (*aPacket* \*packet)

Reset an existing packet.

Zero out any data the packet contains.

**Return values**

- **aErrNone** -- If the reset was successful.
- **aErrParam** -- If the packet is not valid.

**Returns**

Function returns *aErr* values.

*aErr* **aPacket\_AddByte** (*aPacket* \*packet, const uint8\_t byte)

Accumulate a Byte into a packet.

A packet can be constructed byte by byte. the first byte added will be the BrainStem module address, the second byte the data length, and subsequent bytes will be data payload. This call will fail if more than dataLength bytes are added, or if address is an invalid module address (i.e. an odd number).

**Return values**

- **aErrNone** -- Adding the byte was successful.
- **aErrParam** -- The packet was invalid.
- **aErrPacket** -- The byte added violates the BrainStem protocol.

**Returns**

Function returns *aErr* values.

`bool aPacket_IsComplete (const aPacket *packet)`

Determine whether a packet is complete.

A packet can be constructed byte by byte. This call determines whether such a packet has been completed. It checks that dataSize is equal to the currentSize minus the Address and dataSize bytes.

**Returns**

`bool` - True if complete false if not complete.

`aErr aPacket_Destroy (aPacket **packet)`

Destroy a BrainStem packet.

Safely destroy a brainstem packet and deallocate the associated resources.

**Parameters**

`packet` -- A pointer to a pointer of a valid packet. The packet pointer will be set to NULL on successful destruction of the packet.

**Return values**

- `aErrNone` -- The packet was successfully destroyed.
- `aErrParam` -- The packetRef is invalid.

**Returns**

Function returns aErr values.

### 3.4.9 aProtocoldefs.h

`group aProtocoldefs`

BrainStem Protocol Definitions.

`aProtocoldefs.h` Provides protocol and BrainStem specific defines for entities, communication, and protocol specifics.

`aBRAINSTEM_MAXPACKETBYTES 28`

**8 Bytes** - Packet protocol payload maximum.

`group UEI_Defines`

UEI and Command support for C/C++ and Reflex languages.

**Defines**

`ueiSPECIFIER_INDEX_MASK 0x1F`

`0x1F` - Mask bits for Index on index byte.

`ueiSPECIFIER_RETURN_MASK 0xE0`

`0xE0` - Mask bits for Return value on index byte.

`ueiSPECIFIER_RETURN_HOST 0x20`

`1 << 5` - Specifier Bit for UEI response to host.

```
ueiSPECIFIER_RETURN_I2C 0x40
  2 << 5 - Specifier Bit for UEI response to Module over I2C.

ueiSPECIFIER_RETURN_VM 0x60
  3 << 5 - Specifier Bit for UEI response to VM on module.

ueiREPLY_ERROR 0x80
  1 << 7 - Error flag on response in index byte.

ueiREPLY_STREAM 0x40
  1 << 6 - Stream flag on response in index byte.

ueiOPTION_GET 0x40
  0x40 - Option byte code for UEI Get request.

ueiOPTION_VAL 0x00
  0x00 - Option byte code for UEI Val response.

ueiOPTION_SET 0x80
  0x80 - Option byte code for UEI Set request.

ueiOPTION_ACK 0xC0
  0xC0 - Option byte code for UEI Ack response.

ueiOPTION_MASK 0x3F
  0x3F - Mask for getting command option from option byte.

ueiOPTION_OP_MASK 0xC0
  0xC0 - Mask for getting Operation Get/Set/Val/Ack

ueiBYTES_CONTINUE 0x80

ueiBYTES_CONTINUE_MASK 0x7F
```

## System Entity

### group cmdSYSTEM\_Defines

System entity defines.

## Defines

**cmdSYSTEM 3**

3 - System entity command code.

*group cmdSYSTEM\_Command\_Options*

## Defines

**systemModule 1**

1 - Module address option code.

**systemRouter 2**

2 - Router address option code.

**systemHBInterval 3**

3 - Heartbeat interval option code.

**systemLED 4**

4 - User LED option code.

**systemSleep 5**

5 - Sleep option code.

**systemBootSlot 6**

6 - Boot Slot option code.

**aSystemBootSlotNone 255**

255 - Disable boot slot value for Boot Slot option.

**systemVersion 7**

7 - Firmware Version option code.

**systemModel 8**

8 - Model option code.

**systemSerialNumber 9**

9 - Serial Number option code.

**systemSave 10**

10 - System save option code.

**systemReset 11**

11 - System reset option code.

```
systemInputVoltage 12
    12 - Input voltage option code.

systemModuleHardwareOffset 13
    13 - Module Offset option code.

systemModuleBaseAddress 14
    14 - Module Base address option code.

systemModuleSoftwareOffset 15
    15 - Module Software offset option code.

systemRouterAddressSetting 16
    16 - Router address setting option code.

systemIPConfiguration 17
    17 - IP configuration setting option code

systemIPModeDHCP 0

systemIPModeStatic 1

systemIPModeDefault 0

systemIPAddress 18
    18 - IP address setting option code

systemIPStaticAddressSetting 19
    19 - Static IP address setting option code

systemRouteToMe 20
    20 - Route to me setting option code

systemInputCurrent 21
    21 - Input current option code.

systemUptime 22
    22 - System uptime option code.

systemMaxTemperature 23
    23 - System max temperature option code.

systemLogEvents 24
    24 - System log events option code.
```

**systemUnregulatedVoltage** 25  
25 - Unregulated System Voltage option code.

**systemUnregulatedCurrent** 26  
26 - Unregulated System Current option code.

**systemTemperature** 27  
27 - System temperature option code

**systemMinTemperature** 28  
28 - System min temperature option code

**systemInputPowerSource** 29  
29 - System input power source option code

**systemInputPowerBehavior** 31  
30 - System input power behavior option code

**systemInputPowerBehaviorConfig** 31  
31 - System input power behavior config option code

**systemName** 32  
32 - System name option code

**systemPowerLimit** 33  
33 - System power limit option code

**systemPowerLimitMax** 34  
34 - System power limit max option code

**systemPowerLimitState** 35  
35 - System power limit state option code

**systemResetEntityToFactoryDefaults** 36  
36 -

**systemResetDeviceToFactoryDefaults** 37  
37 -

**systemLinkInterface** 38  
38 - Setting the link interface for control

**systemLinkAuto** 0  
0 System Link is automatically defined

```
systemLinkUSBControl 1
  1 System Link through control port

systemLinkUSBHub 2
  2 System Link through the Hub (upstream connection)

systemReserved 39
  39 - Reserved Option Code for Acroname Internal Use Only

systemHardwareVersion 40
  40 - Hardware Version option code

systemErrors 41
  41 - System Error option code

systemErrors_ThermalProtection_Bit 0
  0 - Thermal Protection bit for operational Errors option code.

systemErrors_OutputPowerProtection_Bit 1
  1 - Output Power Protection bit for operational Errors option code.

systemNumberOfOptions 45
  45 - Number of Options for System, always last entry
```

## Slot Entity

```
group cmdSLOT_Defines
  System entity defines.

  Defines

    cmdSLOT 4
      4 - Slot Command Code.

  group cmdSLOT_Command_Options
```

## Defines

```
slotCapacity 1
    1 - Slot Capacity option code.

slotSize 2
    2 - Slot size option code

slotOpenRead 3
    3 - Slot Open Read option code.

slotOpenWrite 4
    4 - Slot Open Write option code.

slotSeek 5
    5 - Slot Seek option code.

slotRead 6
    6 - Slot Read option code.

slotWrite 7
    7 - Slot Write option code.

slotClose 8
    8 - Slot Close option code.

bitSlotError 0x80
    0x80 - Bit Slot error code.
```

## App Entity

```
group cmdAPP_Defines
```

App Entity defines.

## Defines

```
cmdAPP 5
    5 - App command code.
```

```
group cmdAPP_Command_Options
```

## Defines

**appExecute** 1  
1 - Execute option code.

**appReturn** 2  
2 - Return option code.

## Mux Entity

*group cmdMUX\_Defines*  
Mux Entity defines.

## Defines

**cmdMUX** 6  
6 - Mux command code.

*group cmdMUX\_Command\_Options*

## Defines

**muxEnable** 1  
1 - Channel enable option code.

**muxChannel** 2  
2 - Select the active channel on the mux.

**muxVoltage** 3  
3 - Get voltage measurement for the channel.

**muxConfig** 4  
4 - Get voltage measurement for the channel.

**muxConfig\_default** 0

**muxConfig\_splitMode** 1

**muxConfig\_channelPriority** 2

**muxSplit** 5  
5 - Get voltage measurement for the channel.

## Pointer Entity

*group cmdPOINTER\_Defines*

Pointer entity defines.

### Defines

**cmdPOINTER** 7

7 - Pointer command code.

*group cmdPOINTER\_Command\_Options*

### Defines

**pointerOffset** 1

1 - Pointer offset option code.

**pointerMode** 2

2 - Pointer mode option code.

**pointerModeStatic** 0

0 - Static pointer mode for pointer mode option code.

**pointerModeIncrement** 1

1 - Increment pointer mode for pointer mode option code.

**DefaultPointerMode** 0

**pointerModeStatic** - Default pointer mode for pointer mode option code.

**pointerTransferStore** 3

3 - Set Transfer store option code.

**pointerChar** 4

4 - Char pointer option code.

**pointerShort** 5

5 - Short pointer option code.

**pointerInt** 6

6 - Int pointer option code.

**pointerTransferToStore** 7

7 - Transfer to Store option code.

```
pointerTransferFromStore 8
  8 - Transfer From store option code.
```

## Debug command

```
cmdDEBUG 23
  Debug command.
```

## Analog Entity

```
group cmdANALOG_Defines
  Analog Entity defines.
```

### Defines

```
cmdANALOG 30
  30 - Analog command code.
```

```
group cmdANALOG_Command_Options
```

### Defines

```
analogConfiguration 1
  1 - Analog configuration option code.
```

```
analogConfigurationInput 0
  0 - Input configuration for configuration option code.
```

```
analogConfigurationOutput 1
  1 - Output configuration for configuration option code.
```

```
analogConfigurationHiZ 2
  2 - HiZ configuration for configuration option code.
```

```
analogValue 2
  2 - Analog Value option code.
```

```
analogVoltage 3
  3 - Analog Voltage option code.
```

```
analogBulkCaptureSampleRate 4
  4 - Analog Bulk Capture Sample Rate option code.
```

**analog\_Hz\_Minimum** 7000

**7000** - minimum hertz sample rate for Bulk capture Sample Rate option code.

**analog\_Hz\_Maximum** 200000

**200000** - maximum hertz sample rate for Bulk capture Sample Rate option code.

**analogBulkCaptureNumberOfSamples** 5

**5** - Bulk Capture number of samples option code.

**analogBulkCapture** 6

**6** - Bulk Capture option code.

**analogBulkCaptureState** 7

**7** - Bulk Capture State option code.

**bulkCaptureIdle** 0

**0** - Idle state for Bulk Capture state option code.

**bulkCapturePending** 1

**1** - Pending state for Bulk Capture state option code.

**bulkCaptureFinished** 2

**2** - Finished state for Bulk Capture state option code.

**bulkCaptureError** 3

**3** - Error state for Bulk Capture state option code.

**analogRange** 8

**8** - Analog Range option code.

**analogRange\_P0V064N0V064** 0

**0** - +/- 64mV range for Analog Range option code.

**analogRange\_P0V64N0V64** 1

**1** - +/- 640mV range for Analog Range option code.

**analogRange\_P0V128N0V128** 2

**2** - +/- 128mV range for Analog Range option code.

**analogRange\_P1V28N1V28** 3

**3** - +/- 1.28V range for Analog Range option code.

**analogRange\_P1V28N0V0** 4

**4** - 0-1.28V range for Analog Range option code.

**analogRange\_P0V256N0V256 5**

5 - +/- 256mV range for Analog Range option code.

**analogRange\_P2V56N2V56 6**

6 - +/- 2.56V range for Analog Range option code.

**analogRange\_P2V56N0V0 7**

7 - 0-2.56V range for Analog Range option code.

**analogRange\_P0V512N0V512 8**

8 - +/- 512mV range for Analog Range option code.

**analogRange\_P5V12N5V12 9**

9 - +/- 5.12V range for Analog Range option code.

**analogRange\_P5V12N0V0 10**

10 - 0-5.12V range for Analog Range option code.

**analogRange\_P1V024N1V024 11**

11 - +/- 1.024V range for Analog Range option code.

**analogRange\_P10V24N10V24 12**

12 - +/- 10.24V range for Analog Range option code.

**analogRange\_P10V24N0V0 13**

13 - 0-10.24V range for Analog Range option code.

**analogRange\_P2V048N0V0 14**

14 - 0-2.048V range for Analog Range option code.

**analogRange\_P4V096N0V0 15**

15 - 0-4.096V range for Analog Range option code.

**analogEnable 9**

9 - Analog Enable option code.

## Digital Entity

**group cmdDIGITAL\_Defines**

Digital entity defines.

**Defines****cmdDIGITAL** 31

31 - Digital command code.

*group cmdDIGITAL\_Command\_Options***Defines****digitalConfiguration** 1

1 - Digital configuration option code.

**digitalConfigurationInput** 0x00

0 - Input Digital configuration for configuration option code.

**digitalConfigurationOutput** 0x01

1 - Output Digital configuration for configuration option code.

**digitalConfigurationRCServoInput** 0x02

2 - RC Servo Input Digital configuration for configuration option code.

**digitalConfigurationRCServoOutput** 0x03

3 - RC Servo Output Digital configuration for configuration option code.

**digitalConfigurationHiZ** 0x04

4 - Hi Z the digital pin.

**digitalConfigurationInputPullUp** 0x00

0 - Input digital configuration with pull-up.

**digitalConfigurationInputNoPull** 0x04

4 - Input digital configuration with no pull-up/pull-down.

**digitalConfigurationInputPullDown** 0x05

5 - Input digital configuration with pull-down.

**digitalConfigurationSignalOutput** 0x06

6 - Signal output configuration

**digitalConfigurationSignalInput** 0x07

7 - Signal input configuration

**digitalConfigurationSignalCounterInput** 0x08

8 - Signal input conter configuration

```
digitalState 2
 9 - State option code.
```

```
digitalStateAll 3
```

## Rail Entity

```
group cmdRAIL_Defines
  Rail entity defines.
```

### Defines

```
cmdRAIL 32
  32 - Rail command code.
```

```
group cmdRAIL_Command_Options
```

### Defines

```
railVoltage 1
  1 - Rail Voltage option code.
```

```
railCurrent 2
  2 - Rail Current option code.
```

```
railCurrentLimit 3
  3 - Rail Current limit option code.
```

```
railTemperature 4
  4 - Rail Temperature option code.
```

```
railEnable 5
  5 - Rail Enable option code.
```

```
railValue 6
  6 - Rail Value option code.
```

```
railKelvinSensingEnable 7
  7 - Rail Kelvin sensing Mode option code.
```

```
kelvinSensingOff_Value 0
  0 - Kelvin Sensing off mode for Kelvin Sensing mode option code.
```

**kelvinSensingOn\_Value** 1

1 - Kelvin Sensing on mode for Kelvin Sensing mode option code.

**railKelvinSensingState** 8

8 - Kelving Sensing state option code.

**railOperationalMode** 9

9 - Operational mode option code. railOperationalMode is a bit masked field with two multi bit fields.

**railOperationalMode\_HardwareConfiguration\_Offset** 0

0-3 - Operational Mode hardware configuration offset region (bits[0:3]).

**railOperationalModeAuto\_Value** 0

0 - Auto operational mode for operational mode option code.

**railOperationalModeLinear\_Value** 1

1 - Linear mode for operational mode option code.

**railOperationalModeSwitcher\_Value** 2

2 - Switcher mode for operational mode option code.

**railOperationalModeSwitcherLinear\_Value** 3

3 - Switcher Linear mode for operational mode option code.

**railOperationalMode\_Mode\_Offset** 4

4-7 - Operational Mode offset region (bits[4:7]).

**railOperationalModeConstantCurrent\_Value** 0

0 - Constant Current mode for operational mode option code.

**railOperationalModeConstantVoltage\_Value** 1

1 - Constant Voltage mode for operational mode option code.

**railOperationalModeConstantPower\_Value** 2

2 - Constant Power mode for operational mode option code.

**railOperationalModeConstantResistance\_Value** 3

3 - Constant Resistance mode for operational mode option code.

**railOperationalModeFactoryReserved\_Value** 0xF

15 - Factory Reserved Operating Mode.

**DefaultOperationalRailMode\_Value** 0

0 - Default operational mode for operational mode option code.

**railOperationalState 10**

**10** - Operational state option code. The railOperationalState is a bit masked field that has single bit and multi-bit entries.

**railOperationalState\_Initializing\_Bit 0**

**0** - Initializing bit for operational state option code.

**railOperationalState\_Enabled\_Bit 1**

**1** - Enabled bit for operational state option code.

**railOperationalState\_Fault\_Bit 2**

**2** - Fault bit for operational state option code.

**railOperationalState\_HardwareConfiguration\_Offset 8**

**3-7** These bits are unused **8** - Hardware Configuration region (bits[8-15]) for operational state.

**railOperationalStateLinear\_Value 0**

**0** - Linear state for operational state option mode.

**railOperationalStateSwitcher\_Value 1**

**1** - Switcher state for operational state option mode.

**railOperationalStateSwitcherLinear\_Value 2**

**2** - Switcher Linear state for operational state option mode.

**railOperationalStateOverVoltageFault\_Bit 16**

**16** - Over Voltage Fault bit for operational state option mode.

**railOperationalStateUnderVoltageFault\_Bit 17**

**17** - Under Voltage Fault bit for operational state option mode.

**railOperationalStateOverCurrentFault\_Bit 18**

**18** - Over Current Fault bit for operational state option mode.

**railOperationalStateOverPowerFault\_Bit 19**

**19** - Over Power Fault bit for operational state option mode.

**railOperationalStateReversePolarityFault\_Bit 20**

**20** - Reverse Polarity Fault bit for operational state option mode.

**railOperationalStateOverTemperatureFault\_Bit 21**

**21** - Over Temperature Fault bit for operational state option mode.

**railOperationalStateReverseCurrentFault\_Bit 22**

**22** - Reverse Current Fault bit for operational state option mode.

**railOperationalStateOperatingMode\_Offset** 24

23 - This bit is Unused **24-31** - Operating Mode region (bits[24:31]) for operational state.

**railOperationalStateConstantCurrent\_Value** 0

0 - Constant Current mode for operational state option code.

**railOperationalStateConstantVoltage\_Value** 1

1 - Constant Voltage mode for operational state option code.

**railOperationalStateConstantPower\_Value** 2

2 - Constant Power mode for operational state option codes.

**railOperationalStateConstantResistance\_Value** 3

3 - Constant Resistance mode for operational state option code.

**railVoltageSetpoint** 11

11 - Rail Setpoint Voltage option code

**railCurrentSetpoint** 12

12 - Rail Setpoint Current option code.

**railVoltageMinLimit** 13

13 - Rail Voltage min limit option code.

**railVoltageMaxLimit** 14

14 - Rail Voltage max limit option code.

**railPower** 15

15 - Rail power option code.

**railPowerSetpoint** 16

16 - Rail Setpoint power option code.

**railPowerLimit** 17

17 - Rail power limit option code.

**railResistance** 18

18 - Rail resistance option code.

**railResistanceSetpoint** 19

19 - Rail Setpoint resistance option code.

**railClearFaults** 20

20 - Rail Clear Fault Codes.

```
railFactoryReserved 62
```

63 - Factory Reserved Code.

```
railFactoryReserved2 63
```

63 - Factory Reserved Code.

## Temperature Entity

```
group cmdTEMPERATURE_Defines
```

Temperature entity defines.

### Defines

```
cmdTEMPERATURE 33
```

33 - Temperature command code.

```
group cmdTEMPERATURE_Command_Options
```

### Defines

```
temperatureMicroCelsius 1
```

1 - Temperature option code.

```
temperatureMinimumMicroCelsius 2
```

2 - Min temperature option code.

```
temperatureMaximumMicroCelsius 3
```

3 - Max temperature option code.

```
temperatureResetEntityToFactoryDefaults 4
```

4 Reset temperature entity option code

```
temperatureNumberOfOptions 5
```

2 - Number of Options for temperature, always last entry

## Capacity Command

*group cmdCAPACITY\_Defines*

Capacity command.

### Defines

**cmdCAPACITY 73**

73 - Capacity command code.

*group cmdCAPACITY\_Command\_Options*

### Defines

**capacityUEI 1**

1 - UEI command option.

**capacitySubClassSize 3**

3 - SubClass size command option.

**capacityClassQuantity 4**

4 - Class Quantity command option.

**capacitySubClassQuantity 5**

5 - SubClass Quantity command option.

**capacityEntityGroup 6**

6 - Entity Group command option.

**capacityBuild 255**

7 - Build command option.

## Store Entity

*group cmdSTORE\_Defines*

Store entity defines.

## Defines

**cmdSTORE** 77

77 - Store command code.

*group cmdSTORE\_Command\_Options*

## Defines

**storeSlotEnable** 1

1 - Slot Enable option code.

**storeSlotDisable** 2

2 - Slot Disable option code.

**storeSlotState** 3

3 - Slot State option code.

**storeWriteSlot** 4

4 - Write Slot option code.

**storeReadSlot** 5

5 - Read Slot option code.

**storeCloseSlot** 6

6 - Close Slot option code.

**storeLock** 7

7 - Lock Slot option code.

**storeNumberOfOptions** 8

8 - Number of Options for cmdStore, always last entry

## Timer Entity

*group cmdTIMER\_Defines*

Timer Entity Defines.

## Defines

**cmdTIMER** 79

79 - Timer command code.

*group cmdTIMER\_Command\_Options*

## Defines

**timerExpiration** 1

1 - Timer expiration option code.

**timerMode** 2

2 - Timer Mode option code.

**timerModeSingle** 0

0 - Single mode for timer mode option code.

**timerModeRepeat** 1

1 - Repeat mode for timer mode option code.

**DefaultTimerMode** 0

timerModeSingle - Default mode for timer mode option code.

## Clock Entity

*group cmdCLOCK\_Defines*

Clock entity defines.

## Defines

**cmdCLOCK** 83

83 - Clock command code.

*group cmdCLOCK\_Command\_Options*

## Defines

```
clockYear 1
    1 - Year option code.

clockMonth 2
    2 - Month option code.

clockDay 3
    3 - Day option code.

clockHour 4
    4 - Hour option code.

clockMinute 5
    5 - Minute option code.

clockSecond 6
    6 - Second option code.
```

## USB Entity

```
group cmdUSB_Defines
    USB entity defines.
```

## Defines

```
cmdUSB 18
    18 - USB command code.
```

```
group cmdUSB_Command_Options
```

## Defines

```
usbPortEnable 1
    1 - Port Enable option code.

usbPortDisable 2
    2 - Port Disable option code.

usbDataEnable 3
    3 - Data Enable option code.
```

**usbDataDisable** 4

4 - Data Disable option code.

**usbPowerEnable** 5

5 - Power Enable option code.

**usbPowerDisable** 6

6 - Power Disable option code.

**usbPortCurrent** 7

7 - Port Current option code.

**usbPortVoltage** 8

8 - Port Voltage option code.

**usbHubMode** 9

9 - Hub Mode option code.

**usbPortClearErrorStatus** 12

12 - Hub Clear Error Status option code.

**usbUpstreamMode** 14

13 - SystemTemperature option code.

**usbUpstreamModeAuto** 2

2 - UpstreamMode Auto for upstream mode option code.

**usbUpstreamModePort0** 0

0 - UpstreamMode Port 0 for upstream mode option code.

**usbUpstreamModePort1** 1

1 - UpstreamMode Port 1 for upstream mode option code.

**usbUpstreamModeNone** 255

255 - UpstreamMode None to turn off all upstream connections.

**usbUpstreamModeDefault** 2

1 - UpstreamMode default for upstream mode option code.

**usbUpstreamState** 15

15 - UpstreamState option code.

**usbUpstreamStateNone** 2

2 - UpstreamMode Auto for upstream mode option code.

**usbUpstreamStatePort 0**

0 - UpstreamMode Port 0 for upstream mode option code.

**usbUpstreamStatePort 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 passthrough of VBUS

```
usbPortMode_USB2AEnable 4
  4 - USB2 dataline A side enabled

usbPortMode_USB2BEnable 5
  4 - USB2 dataline B side enabled

usbPortMode_VBusEnable 6
  5 - USB VBUS enabled

usbPortMode_SuperSpeed1Enable 7
  6 - USB SS Speed dataline side A enabled

usbPortMode_SuperSpeed2Enable 8
  7 - USB SS Speed dataline side B enabled

usbPortMode_USB2BoostEnable 9
  8 - USB2 Boost Mode Enabled

usbPortMode_USB3BoostEnable 10
  9 - USB3 Boost Mode Enabled

usbPortMode_AutoConnectEnable 11
  10 - Auto-connect Mode Enabled

usbPortMode_CC1Enable 12
  11 - CC1 Enabled

usbPortMode_CC2Enable 13
  12 - CC2 Enabled

usbPortMode_SBUEnable 14
  13 - SBU1 Enabled

usbPortMode_CCFlipEnable 15
  15 - Flip CC1 and CC2

usbPortMode_SSFlipEnable 16
  16 - Flip Super speed data lines

usbPortMode_SBUFlipEnable 17
  17 - Flip Side Band Unit lines.

usbPortMode_USB2FlipEnable 18
  18 - Flip Side Band Unit lines.
```

**usbPortMode\_CC1InjectEnable** 19

19 - Internal Use

**usbPortMode\_CC2InjectEnable** 20

20 - Internal Use

**usbHiSpeedDataEnable** 21

21 - Hi-Speed Data Enable option code.

**usbHiSpeedDataDisable** 22

22 - Hi-Speed Data Disable option code.

**usbSuperSpeedDataEnable** 23

23 - SuperSpeed Data Enable option code.

**usbSuperSpeedDataDisable** 24

24 - SuperSpeed Data Disable option code.

**usbDownstreamDataSpeed** 25

25 - Get downstream port speed option code.

**usbDownstreamDataSpeed\_na** 0

0 - Unknown

**usbDownstreamDataSpeed\_hs** 1

1 - Hi-Speed (2.0)

**usbDownstreamDataSpeed\_ss** 2

2 - SuperSpeed (3.0)

**usbDownstreamDataSpeed\_ls** 3

3 - TODO

**usbConnectMode** 26

26 USB connect mode option code

**usbManualConnect** 0

0 - Auto connect disabled

**usbAutoConnect** 1

1 - Auto connect enabled

**usbCC1Enable** 27

27 - CC1 Enable option code (USB Type C).

**usbCC2Enable** 28

28 - CC2 Disable option code (USB Type C).

**usbSBUEnable** 29

29 - SBU1/2 enable option code (USB Type C).

**usbCC1Current** 30

30 - CC1 get current option code (USB Type C).

**usbCC2Current** 31

31 - CC2 get current option code (USB Type C).

**usbCC1Voltage** 32

32 - CC1 get voltage option code (USB Type C).

**usbCC2Voltage** 33

33 - CC2 get voltage option code (USB Type C).

**usbPortState** 34

34 - TODO

**usbPortError** 35

35 - TODO

**usbCableFlip** 36

36 - TODO

**usbAltMode** 37

37 - USB Alt Mode configuration.

**usbAltMode\_disabled** 0

0 - Disabled mode

**usbAltMode\_normal** 1

1 - Normal mode (USB 3.1)

**usbAltMode\_4LaneDP\_ComToHost** 2

2 - Alt Mode - 4 lanes of display port “Common” side connected to host

**usbAltMode\_4LaneDP\_MuxToHost** 3

3 - Alt Mode - 4 lanes of display port “Mux” side connected to host

**usbAltMode\_2LaneDP\_ComToHost\_wUSB3** 4

4 - Alt Mode - 2 lanes of display port “Common” side connected to host with USB3.1

**usbAltMode\_2LaneDP\_MuxToHost\_wUSB3 5**

5 - Alt Mode - 2 lanes of display port “Mux” side connected to host with USB3.1

**usbAltMode\_2LaneDP\_ComToHost\_wUSB3\_Inverted 6**

6 - Alt Mode - 2 lanes of display port “Common” side connected to host with USB3.1 with channels 1,2 and 3,4 inverted

**usbAltMode\_2LaneDP\_MuxToHost\_wUSB3\_Inverted 7**

7 - Alt Mode - 2 lanes of display port “Mux” side connected to host with USB3.1 with channels 1,2 and 3,4 inverted

**usbSBU1Voltage 38**

38 - SBU1 get voltage option code (USB Type C).

**usbSBU2Voltage 39**

39 - SBU2 get voltage option code (USB Type C).

## Upgrade command

**cmdUPGRADE 95**

Upgrade command.

## Last command

**cmdLAST 95**

Last command.

## 3.4.10 aStream.h

**group aStream**

Platform Independent Stream Abstraction.

*aStream.h* provides a platform independent stream abstraction for common I/O streams. Provides facilities for creating and destroying as well as writing and reading from streams.

**typedef void \*aStreamRef**

Typedef *aStreamRef* Opaque reference to stream primitive.

**group StreamCallbacks**

## TypeDefs

typedef *aErr* (\**aStreamGetProc*)(*uint8\_t* \**pData*, *void* \**ref*)

Typedef *aStreamGetProc*.

This callback is defined to read one byte from the concrete stream implementation.

**Param pData**

The data Buffer to fill.

**Param ref**

Opaque reference to concrete stream implementation.

**Retval aErrNone**

Successfully read the byte.

**Retval aErrNotReady**

No bytes in stream to read.

**Retval aErrEOF**

Reached the end of the stream.

**Retval aErrIO**

An error encountered reading from stream.

**Return**

Function returns *aErr* values.

typedef *aErr* (\**aStreamPutProc*)(*const uint8\_t* \**pData*, *void* \**ref*)

Typedef *aStreamPutProc*.

This callback is defined to write one byte to the concrete stream implementation.

**Param pData**

The data Buffer to write.

**Param ref**

opaque reference to concrete stream implementation.

**Retval aErrNone**

Successfully wrote the byte.

**Retval aErrIO**

An error encountered reading from stream.

**Return**

Function returns *aErr* values.

typedef *aErr* (\**aStreamDeleteProc*)(*void* \**ref*)

Typedef *aStreamDeleteProc*.

This callback is defined to destroy the concrete stream implementation.

**Param ref**

opaque reference to concrete stream implementation.

**Retval aErrNone**

Successfully destroyed.

**Retval aErrParam**

Invalid ref.

**Return**

Function returns aErr values.

`typedef aErr (*aStreamWriteProc)(const uint8_t *pData, const size_t nSize, void *ref)`

Typedef *aStreamWriteProc*. (Optional)

Optional multi-byte write for efficiency, not required..

**Param pData**

The pointer to the data to write to the stream.

**Param nSize**

The size of the data buffer in bytes.

**Param ref**

Opaque reference to concrete stream implementation.

**Retval aErrNone**

Successfully destroyed.

**Retval aErrIO**

An error encountered reading from stream.

**Return**

Function returns aErr values.

**enum aBaudRate**

Enum *aBaudRate*.

Accepted serial stream baudrates.

*Values:*

**enumerator aBAUD\_2400**

2400 baud

**enumerator aBAUD\_4800**

4800 baud

**enumerator aBAUD\_9600**

9600 baud

**enumerator aBAUD\_19200**

19,200 baud

**enumerator aBAUD\_38400**

38,400 baud

**enumerator aBAUD\_57600**

57,600 baud

**enumerator aBAUD\_115200**

115,200 baud

enumerator **aBAUD\_230400**

230,400 baud

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 contents of the buffer.

Flushes the content of the buffer into the flushStream.

**Parameters**

- `bufferStreamRef` -- The buffered stream to flush.
- `flushStream` -- the stream to flush the buffer into.

**Return values**

- `aErrNone` -- The flush succeeded.
- `aErrParam` -- The bufferStream is invalid.
- `aErrIO` -- IO error writing to flushStream.

**Returns**

Function returns aErr values.

`aErr aStream_CreateLogStream (const aStreamRef streamToLog, const aStreamRef upStreamLog,  
const aStreamRef downStreamLog, aStreamRef *pLogStreamRef)`

Create a Logging stream.

Creates a stream which contains an upstream log stream and a downstream log stream. The logging stream logs reads to the upstream log and writes to the downstream log, while passing all data to and from the pLogStreamRef.

**Parameters**

- `streamToLog` -- The reference to the stream to log.
- `upStreamLog` -- Log stream for reads.
- `downStreamLog` -- Log stream for writes.
- `pLogStreamRef` -- The logged stream reference.

**Return values**

- `aErrNone` -- Successful creation.
- `aErrParam` -- The stream to log is invalid.

**Returns**

Function returns aErr values.

**Returns**

`aErrIO` - A communication error occurred creating the logging stream.

*aErr* **aStream\_Read** (*aStreamRef* streamRef, *uint8\_t* \*pBuffer, *const size\_t* length)

Read a byte array record from a stream.

Read a byte array record from a stream.

#### Parameters

- **streamRef** -- The reference to the stream to read from.
- **pBuffer** -- byte array buffer to read into.
- **length** -- the length of the read buffer.

#### Return values

- **aErrNone** -- Successful read.
- **aErrMode** -- The streamRef is not readable.
- **aErrIO** -- An error occurred reading the data.

#### Returns

Function returns aErr values.

*aErr* **aStream\_Write** (*aStreamRef* streamRef, *const uint8\_t* \*pBuffer, *const size\_t* length)

Write a byte array to a Stream.

Write a byte array to a Stream.

#### Parameters

- **streamRef** -- The reference to the stream to write to.
- **pBuffer** -- byte array to write out to the stream.
- **length** -- the byte array length

#### Return values

- **aErrNone** -- Successful write.
- **aErrMode** -- The streamRef is not writable.
- **aErrIO** -- An error occurred writing the data.

#### Returns

Function returns aErr values.

*aErr* **aStream\_ReadRecord** (*aStreamRef* streamRef, *uint8\_t* \*pBuffer, *size\_t* \*lengthRead, *const size\_t* maxLength, *const uint8\_t* \*recordTerminator, *const size\_t* terminatorLength)

Read a byte array record from a stream with a record terminator.

Read a byte array record from a stream with a record terminator.

#### Parameters

- **streamRef** -- The reference to the stream to read from.
- **pBuffer** -- Byte array buffer to read into.
- **lengthRead** -- The length of the read buffer.
- **maxLength** -- The Maximum record length.
- **recordTerminator** -- The byte array representing the record terminator.
- **terminatorLength** -- The length of the record terminator.

**Return values**

**aErrNone** -- Successful read.

**Returns**

Function returns aErr values.

**Returns**

**aErrMode** - The streamRef is not readable.

**Returns**

**aErrIO** - An error occurred reading the data.

*aErr aStream\_WriteRecord (aStreamRef streamRef, const uint8\_t \*pBuffer, const size\_t 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.

### 3.4.11 aTime.h

**group aTime**

Basic Time procedures Sleep and Get process tics.

*aTime.h* provides a platform independent interface for millisecond sleep, and for getting process tics.

**unsigned long aTime\_GetMSTicks (void)**

Get the current tick count in milliseconds.

This call returns a number of milliseconds. Depending on the platform, this can be the number of milliseconds since the last boot, or from the epoch start. As such, this call should not be used as an external reference clock. It is accurate when used as a differential, i.e. internal, measurement only.

**Returns**

unsigned long number of milliseconds elapsed.

`aErr aTime_MSSleep (const unsigned long msTime)`

Sleep the current process for msTime milliseconds.

Sleeps the current process. This is not an active sleep, there are no signals which will “wake” the process.

#### Parameters

`msTime` – Milliseconds to sleep.

#### Return values

- `aErrNone` – The call returned successfully.
- `aErrUnknown` – Um unknown what went wrong.

#### Returns

Function returns aErr values.

### 3.4.12 aUEI.h

*group aUEI*

UEI Utilities.

*aUEI.h* Provides structs and utilities for working with UEIs.

`enum dataType`

TypeDef Enum `dataType`.

UEI datatype

*Values:*

`enumerator aUEI_VOID`

Void datatype.

`enumerator aUEI_BYTE`

Char datatype.

`enumerator aUEI_SHORT`

Short datatype.

`enumerator aUEI_INT`

Int datatype.

`enumerator aUEI_BYTES`

Bytes datatype.

`struct uei`

TypeDef Struct `uei`.

UEI data struct.

## Public Members

`uint8_t module`

Module address.

`uint8_t command`

Command code.

`uint8_t option`

option code & UEI operation.

`uint8_t specifier`

Entity index & response specifier.

`uint8_t byteVal`

Char value union member.

`uint16_t shortVal`

Short value union member.

`uint32_t intVal`

Int value union member.

`dataType type`

Union dataType.

`uint16_t aUEI_RetrieveShort (const uint8_t *p)`

Retrieve a short from a UEI.

### Parameters

`p` - Pointer to byte array containing short.

### Returns

`uint16_t` The short value.

`void aUEI_StoreShort (uint8_t *p, uint16_t v)`

Store a short in a UEI.

### Parameters

- `p` - Pointer to uei shortVal.

- `v` - Short value to store.

`uint32_t aUEI_RetrieveInt (const uint8_t *p)`

Retrieve an Int from a UEI.

### Parameters

`p` - Pointer to byte array containing the Int.

### Returns

`uint32_t` The integer value.

---

```
void aUEI_StoreInt (uint8_t *p, uint32_t v)
```

Store an Int in a UEI.

#### Parameters

- **p** – Pointer to the IntVal of a UEI.
- **v** – The value to store.

### 3.4.13 aVersion.h

*group aVersion*

Library version interface.

[aVersion.h](#) Provides version information for the BrainStem2 library.

#### aVERSION\_MAJOR 2

Major revision level of library.

Major revision bumps will break compatibility with existing versions and may introduce protocol changes or other fundamental differences.

#### aVERSION\_MINOR 9

Minor revision level of library.

Minor revisions should largely be compatible, however new features may be added with a minor revision change.

#### aVERSION\_PATCH 25

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.

`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 *aVersionGetString (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.

## 3.5 .NET API Reference

Welcome to the BrainStem .NET API reference documentation. This documentation covers the .NET Acroname BrainStem library. This reference assumes that you understand the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation.

- *BrainStem Overview*
- *BrainStem Terminology*
- *Getting Started with the BrainStem*.

The Getting started guide is particularly useful for learning how to use the application tools we provide to communicate with your hardware.

---

### 3.5.1 Classes

#### class **AnalogClass**

The *AnalogClass* is the interface to analog entities on BrainStem modules. Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

#### class **AppClass**

The *AppClass* is used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when

#### class **ClockClass**

*ClockClass*. Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

---

**Note:** Clock time must be reset if power to the BrainStem module is lost.

---

#### class **DigitalClass**

The *DigitalClass* is the interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

#### class **EqualizerClass**

*EqualizerClass*. Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

#### class **I2CClass**

The *I2CClass* is the interface the I2C busses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entitie's bus.

#### class **ModuleClass**

*ModuleClass*. Provides a generic interface to a BrainStem hardware module. The Module class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

**class MuxClass**

*MuxClass*. A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexor) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

**class PointerClass**

*PointerClass*. Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via `setOffset`. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

**class RailClass**

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

**class RCServoClass**

The *RCServoClass* is the interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

**class RelayClass**

The *RelayClass* is the interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

**class SignalClass**

*SignalClass* is the interface to digital pins configured to produce square wave signals.

This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (`T3Time`) and the active portion of the cycle as (`T2Time`). See the entity overview section of the reference for more detail regarding the timing.

**class StoreClass**

*StoreClass*. The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

**class SystemClass**

*SystemClass*. The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc. The most common brainstem example uses the system entity to blink the User LED.

**class TemperatureClass**

*TemperatureClass*. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

**class TimerClass**

*TimerClass*. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

**class UARTClass**

*UARTClass*. A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines.

**class USBClass**

*USBClass*. The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

### 3.5.2 Errors

**enum class Acroname::BrainStem2CLI::aErr**

*Values:*

**enumerator aErrNone**

0 - Success, no error.

**enumerator aErrMemory**

1 - Memory allocation.

**enumerator aErrParam**

2 - Invalid parameter.

**enumerator aErrNotFound**

3 - Not found.

**enumerator aErrFileNameLength**

4 - File name too long.

**enumerator aErrBusy**

5 - Resource busy.

**enumerator aErrIO**

6 - Input/Output error.

**enumerator aErrMode**

7 - Invalid Mode.

enumerator **aErrWrite**  
8 - Write error.

enumerator **aErrRead**  
9 - Read error.

enumerator **aErrEOF**  
10 - End of file.

enumerator **aErrNotReady**  
11 - Not ready, no bytes available.

enumerator **aErrPermission**  
12 - Insufficient permissions.

enumerator **aErrRange**  
13 - Value out of range.

enumerator **aErrSize**  
14 - Invalid Size.

enumerator **aErrOverrun**  
15 - Buffer/queue overrun.

enumerator **aErrParse**  
16 - Parse error.

enumerator **aErrConfiguration**  
17 - Configuration error.

enumerator **aErrTimeout**  
18 - Timeout occurred.

enumerator **aErrInitialization**  
19 - Initialization error.

enumerator **aErrVersion**  
20 - Invalid version.

enumerator **aErrUnimplemented**  
21 - Functionality unimplemented.

enumerator **aErrDuplicate**  
22 - Duplicate request.

enumerator **aErrCancel**  
23 - Cancelation occurred, or did not complete.

enumerator **aErrPacket**  
24 - Packet unsigned char invalid.

enumerator **aErrConnection**  
25 - Connection error.

enumerator **aErrIndexRange**  
26 - Index out of range.

enumerator **aErrShortCommand**  
27 - BrainStem command too short.

enumerator **aErrInvalidEntity**

28 - Invalid entity error.

enumerator **aErrInvalidOption**

29 - Invalid option code.

enumerator **aErrResource**

30 - Resource unavailable.

enumerator **aErrMedia**

31 - Media error.

enumerator **aErrAsyncReturn**

32 - Asynchronous return.

enumerator **aErrUnknown**

32 - Unknown error.

### 3.5.3 BrainStem2 CLI Types

enum class Acroname::BrainStem2CLI::**aErr**

*Values:*

enumerator **aErrNone**

0 - Success, no error.

enumerator **aErrMemory**

1 - Memory allocation.

enumerator **aErrParam**

2 - Invalid parameter.

enumerator **aErrNotFound**

3 - Not found.

enumerator **aErrFileNameLength**

4 - File name too long.

enumerator **aErrBusy**

5 - Resource busy.

enumerator **aErrIO**

6 - Input/Output error.

enumerator **aErrMode**

7 - Invalid Mode.

enumerator **aErrWrite**

8 - Write error.

enumerator **aErrRead**

9 - Read error.

enumerator **aErrEOF**

10 - End of file.

enumerator **aErrNotReady**

11 - Not ready, no bytes available.

enumerator **aErrPermission**

12 - Insufficient permissions.

enumerator **aErrRange**

13 - Value out of range.

enumerator **aErrSize**

14 - Invalid Size.

enumerator **aErrOverrun**

15 - Buffer/queue overrun.

enumerator **aErrParse**

16 - Parse error.

enumerator **aErrConfiguration**

17 - Configuration error.

enumerator **aErrTimeout**

18 - Timeout occurred.

enumerator **aErrInitialization**

19 - Initialization error.

enumerator **aErrVersion**

20 - Invalid version.

enumerator **aErrUnimplemented**

21 - Functionality unimplemented.

enumerator **aErrDuplicate**

22 - Duplicate request.

enumerator **aErrCancel**

23 - 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 **aErrUnknown**

32 - Unknown error.

### 3.5.4 Analog Class

class **AnalogClass**

The [AnalogClass](#) is the interface to analog entities on BrainStem modules. Analog entities may be configured as 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** (Acroname::BrainStem::*AnalogClass* &analog)

Constructor.

**~AnalogClass ()**

Destructor.

**!AnalogClass ()**

Finalizer.

*aErr* **getValue** (unsigned short %value)

Get the raw ADC output value in bits.

**Note:** Not all modules are provide 16 useful bits; this value's least significant bits are zero-padded to 16 bits. Refer to the module's datasheet to determine analog bit depth and reference voltage.

### Parameters

**value** – 16 bit analog reading with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

### Returns

Returns common entity return values

*aErr* **setValue** (const unsigned short value)

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

**Note:** Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

### Parameters

**value** – 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

### Returns

Returns common entity return values

*aErr* **getVoltage** (int %microvolts)

Get the scaled micro volt value with refrence to ground.

**Note:** Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

### Parameters

**microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

### Returns

Returns common entity return values

`aErr setVoltage (const int microvolts)`

Set the voltage level of an analog output (DAC) in microvolts.

---

**Note:** Voltage range is dependent on the specific DAC channel range.

---

**Parameters**

`microvolts` – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

**Returns**

Returns common entity return values

`aErr getEnable (unsigned char %enable)`

Get the analog output enable status.

**Parameters**

`enable` – 0 if disabled 1 if enabled.

**Returns**

Returns common entity return values

`aErr setEnable (const unsigned char enable)`

Set the analog output enable state.

**Parameters**

`enable` – set 1 to enable or 0 to disable.

**Returns**

Returns common entity return values

`aErr getRange (unsigned char %range)`

Get the analog input range.

**Parameters**

`range` – 8 bit value corresponding to a discrete range option

**Returns**

Returns common entity return values

`aErr setRange (const unsigned char range)`

Set the analog input range.

**Parameters**

`range` – 8 bit value corresponding to a discrete range option

**Returns**

Returns common entity return values

`aErr getConfiguration (unsigned char %configuration)`

Get the analog configuration.

**Parameters**

`configuration` -- Current configuration of the analog entity.

**Returns**

Returns common entity return values

`aErr setConfiguration (const unsigned char configuration)`

Set the analog configuration.

**Parameters**

`configuration` -- bitAnalogConfigurationOutput configures the analog entity as an output.

**Return values**

`aErrConfiguration` -- Entity does not support this configuration.

**Returns**

EntityReturnValues “common entity” return values

*aErr* **getBulkCaptureSampleRate** (unsigned int %value)

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

**Parameters**

**value** – upon success filled with current sample rate in samples per second (Hertz).

**Returns**

Returns common entity return values

*aErr* **setBulkCaptureSampleRate** (const unsigned int value)

Set the sample rate for this analog when bulk capturing.

**Parameters**

**value** – sample rate in samples per second (Hertz). Minimum rate: 7,000 Hz  
Maximum rate: 200,000 Hz

**Returns**

Returns common entity return values

*aErr* **getBulkCaptureNumberOfSamples** (unsigned int %value)

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

**Parameters**

**value** – number of samples.

**Returns**

Returns common entity return values

*aErr* **setBulkCaptureNumberOfSamples** (const unsigned int value)

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

**Parameters**

**value** – number of samples. Minimum # of Samples: 0 Maximum # of Samples: (BRAINSTEM\_RAM\_SLOT\_SIZE / 2) = (3FFF / 2) = 1FFF = 8191

**Returns**

Returns common entity return values

*aErr* **initiateBulkCapture** (void)

Initiate a BulkCapture on this analog. Captured measurements are stored in the module's RAM store (RAM\_STORE) slot 0. Data is stored in a contiguous unsigned char array with each sample stored in two consecutive bytes, LSB first.

**Returns**

Returns common entity return values. When the bulk capture is complete *getBulkCaptureState()* will return either bulkCaptureFinished or bulkCaptureError.

*aErr* **getBulkCaptureState** (unsigned char %state)

get the current bulk capture state for this analog.

**Parameters**

**state** – the state of bulk capture.

- Idle: bulkCaptureIdle = 0
- Pending: bulkCapturePending = 1
- Finished: bulkCaptureFinished = 2
- Error: bulkCaptureError = 3

**Returns**

Returns common entity return values

### 3.5.5 App Class

```
class AppClass
```

The `AppClass` is used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when

#### Public Functions

```
AppClass (Acroname::BrainStem::AppClass &app)
```

Constructor.

```
~AppClass ()
```

Destructor.

```
!AppClass ()
```

Finalizer.

```
aErr execute (const unsigned int appParam)
```

Execute the app reflex on the module. Don't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

##### Parameters

`appParam` – The app parameter handed to the reflex.

##### Returns

`aErr::aErrNone` - success.

##### Returns

`aErr::aErrTimeout` - The request timed out waiting to start execution.

##### Returns

`aErr::aErrConnection` - No active link connection.

##### Returns

`aErr::aErrNotFound` - the app reflex was not found or not enabled on the module.

```
aErr execute (const unsigned int appParam, unsigned int %returnVal)
```

Execute the app reflex on the module. Wait for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

##### Parameters

- `appParam` – The app parameter handed to the reflex.
- `returnVal` – The return value filled in from the result of executing the reflex routine.

##### Returns

`aErr::aErrNone` - success.

##### Returns

`aErr::aErrTimeout` - The request timed out waiting for a response.

##### Returns

`aErr::aErrConnection` - No active link connection.

##### Returns

`aErr::aErrNotFound` - the app reflex was not found or not enabled on the module.

```
aErr execute (const unsigned int appParam, unsigned int %returnVal, const unsigned int msTimeout)
```

Execute the app reflex on the module. Wait for a return from the reflex execution for msTimeout milliseconds. This method will block for up to msTimeout.

**Parameters**

- **appParam** – The app parameter handed to the reflex.
- **returnVal** – The return value filled in from the result of executing the reflex routine.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

**Returns**

aErr::aErrNone - success.

**Returns**

aErr::aErrTimeout - The request timed out waiting for a response.

**Returns**

aErr::aErrConnection - No active link connection.

**Returns**

aErr::aErrNotFound - the app reflex was not found or not enabled on the module.

### 3.5.6 Clock Class

**class ClockClass**

**ClockClass**. Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

---

**Note:** Clock time must be reset if power to the BrainStem module is lost.

---

#### Public Functions

**ClockClass** (Acroname::BrainStem::*ClockClass* &clock)

Constructor.

**~ClockClass ()**

Destructor.

**!ClockClass ()**

Finalizer.

**aErr getYear** (unsigned short %year)

Get the four digit year value (0-4095).

**Parameters**

**year** – Get the year portion of the real-time clock value.

**Returns**

Returns common entity return values

**aErr setYear** (const unsigned short year)

Set the four digit year value (0-4095).

**Parameters**

**year** – Set the year portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **getMonth** (unsigned char %month)

Get the two digit month value (1-12).

**Parameters**

month – The two digit month portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **setMonth** (const unsigned char month)

Set the two digit month value (1-12).

**Parameters**

month – The two digit month portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **getDay** (unsigned char %day)

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

**Parameters**

day – The two digit day portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **setDay** (const unsigned char day)

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

**Parameters**

day – The two digit day portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **getHour** (unsigned char %hour)

Get the two digit hour value (0-23).

**Parameters**

hour – The two digit hour portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **setHour** (const unsigned char hour)

Set the two digit hour value (0-23).

**Parameters**

hour – The two digit hour portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **getMinute** (unsigned char %min)

Get the two digit minute value (0-59).

**Parameters**

min – The two digit minute portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **setMinute** (const unsigned char min)

Set the two digit minute value (0-59).

**Parameters**

min – The two digit minute portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **getSecond** (unsigned char %sec)

Get the two digit second value (0-59).

**Parameters**

**sec** – The two digit second portion of the real-time clock value.

**Returns**

Returns common entity return values

*aErr* **setSecond** (const unsigned char sec)

Set the two digit second value (0-59).

**Parameters**

**sec** – The two digit second portion of the real-time clock value.

**Returns**

Returns common entity return values

### 3.5.7 Digital Class

**class DigitalClass**

The *DigitalClass* is the interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

#### Public Functions

**DigitalClass** (Acroname::BrainStem::*DigitalClass* &digital)

Constructor.

**~DigitalClass ()**

Destructor.

**!DigitalClass ()**

Finalizer.

*aErr* **setState** (const unsigned char state)

Set the logical state.

**Parameters**

**state** – The state to be set. 0 is logic low, 1 is logic high.

**Returns**

Returns common entity return values

*aErr* **getState** (unsigned char %state)

Get the state.

**Parameters**

**state** – The current state of the digital entity. 0 is logic low, 1 is logic high.

Note: If in high Z state an error will be returned.

**Returns**

Returns common entity return values

*aErr* **setConfiguration** (const unsigned char configuration)

Set the digital configuration to one of the available 5 states. Note: Some configurations are only supported on specific pins.

**Parameters****configuration -**

- Digital Input: digitalConfigurationInput = 0
- Digital Output: digitalConfigurationOutput = 1
- RC Servo Input: FdigitalConfigurationRCServoInput = 2
- RC Servo 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 digits 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 digits varies across BrainStem modules. Refer to the datasheet for the capabilities of your module.

**Parameters****state** - The state of all digitals where bit 0 = digital 0, bit 1 = digital 1 etc. 0 is logic low, 1 is logic high.**Returns**

Returns common entity return values

### 3.5.8 Equalizer Class

**class EqualizerClass**

*EqualizerClass*. Provides receiver and transmitter gain/boost/emphasis settings for some of Acroname's products. Please see product documentation for further details.

## Public Functions

**`EqualizerClass`** (Acroname::BrainStem::*EqualizerClass* &equalizer)  
 Constructor.

**`~EqualizerClass`** (void)  
 Destructor.

**`!EqualizerClass (void)`**  
 Finalizer.

**`aErr setReceiverConfig`** (const unsigned char channel, const unsigned char config)  
 Sets the receiver configuration for a given channel.

**Parameters**

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

**Returns**  
 Returns common entity return values.

**`aErr getReceiverConfig`** (const unsigned char channel, unsigned char %config)  
 Gets the receiver configuration for a given channel.

**Parameters**

- `channel` – The equalizer receiver channel.
- `config` – Configuration of the receiver.

**Returns**  
 Returns common entity return values.

**`aErr setTransmitterConfig`** (const unsigned char config)  
 Sets the transmitter configuration

**Parameters**

- `config` – Configuration to be applied to the transmitter.

**Returns**  
 Returns common entity return values.

**`aErr getTransmitterConfig`** (unsigned char %config)  
 Gets the transmitter configuration

**Parameters**

- `config` – Configuration of the Transmitter.

**Returns**  
 Returns common entity return values.

## 3.5.9 I2C Class

class **I2CClass**

The *I2CClass* is the interface the I2C busses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entitie's bus.

## Public Functions

**I2CClass** (Acroname::BrainStem::*I2CClass* &i2c)

Constructor.

**~I2CClass ()**

Destructor.

**!I2CClass ()**

Finalizer.

**aErr read** (const unsigned char address, const unsigned char length, unsigned char %result)

Read from a device on this I2C bus.

### Parameters

- **address** -- The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** -- The length of the data to read in bytes.
- **result** -- The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAIN-STEM\_MAXPACKETBYTES - 5

### Returns

Returns common entity return values

**aErr write** (const unsigned char address, const unsigned char length, const unsigned char %data)

Write to a device on this I2C bus.

### Parameters

- **address** -- The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** -- The length of the data to read in bytes.
- **data** -- The data to send to the device, This array should be no larger than aBRAINSTEM\_MAXPACKETBYTES - 5

### Returns

Returns common entity return values

**aErr setPullup** (const bool bEnable)

Set bus pullup state. This call only works with stems that have software controlled pullups. Check the datasheet for more information. This parameter is saved when system.save is called.

### Parameters

**bEnable** -- true enables pullups false disables them.

### Returns

Returns common entity return values

**aErr setSpeed** (const unsigned char speed)

Set I2C bus speed.

This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz

### Parameters

**speed** -- The speed setting value.

### Returns

Returns common entity return values

**aErr getSpeed** (unsigned char %speed)

Get I2C bus speed.

This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz

**Parameters**

- **speed** -- The speed setting value.

**Returns**

Returns common entity return values

### 3.5.10 Module Class

**class ModuleClass**

*ModuleClass*. Provides a generic interface to a BrainStem hardware module. The Module class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

#### Public Functions

**ModuleClass (Acroname::BrainStem::Module &module)**

Constructor.

**~ModuleClass ()**

Destructor.

**!ModuleClass ()**

Finalizer.

**BrainStem2CLI::aErr discoverAndConnect (m\_linkType transport)**

A discovery-based connect. This member function will attempt to connect to the first BrainStem module found. If this module does not match the object type the connection will fail.

**Parameters**

- **transport** -- Transport on which to search for available BrainStem link modules. See the m\_linkType “transport” enum for supported transports.

**Returns**

- aErr::aErrBusy - if the module is already in use.

**Returns**

- aErr::aErrParam - if the transport type is undefined.

**Returns**

- aErr::aErrNotFound - if the module cannot be found.

**Returns**

- aErr::aErrNone If the connect was successful.

**BrainStem2CLI::aErr discoverAndConnect (m\_linkType transport, unsigned int serialNum)**

A discovery-based connect. This member function will connect to the first available BrainStem found on the given transport. If the serial number is passed, it will only connect to the module with that serial number. Passing 0 as the serial number will create a link to the first link module found on the specified transport. If a link module is found on the specified transport, a connection will

**Parameters**

- **transport** -- Transport on which to search for available BrainStem link modules. See the m\_linkType “transport” enum for supported transports.

- **serialNum** -- Specify a serial number to connect to a specific link module.  
Use 0 to connect to the first link module found.

**Returns**

aErr::aErrBusy - if the module is already in use.

**Returns**

aErr::aErrParam - if the transport type is undefined.

**Returns**

aErr::aErrNotFound - if the module cannot be found.

**Returns**

aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **connectFromSpec** (m\_linkSpec spec)

Connect to a link with a fully defined specifier.

**Parameters**

**spec** -- Connect to module with specifier.

**Returns**

aErr::aErrInitialization - If there is currently no link object.

**Returns**

aErr::aErrBusy - If the link is currently connected.

**Returns**

aErr::aErrParam - if the specifier is incorrect.

**Returns**

aErr::aErrNotFound - if the module cannot be found.

**Returns**

aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **connect** (m\_linkType transport)

Connect using the current link specifier.

**Parameters**

**transport** -- Transport on which to search for available BrainStem link modules. See the m\_linkType “transport” enum for supported transports.

**Returns**

aErr::aErrBusy - if the module is already in use.

**Returns**

aErr::aErrParam - if the type is incorrect or serialNum is not specified

**Returns**

aErr::aErrNotFound - if the module cannot be found.

**Returns**

aErr::aErrNone If the connect was successful.

BrainStem2CLI::*aErr* **connect** (m\_linkType transport, unsigned int serialNum)

Connect using the current link specifier.

**Parameters**

- **transport** -- Transport on which to search for available BrainStem link modules. See the m\_linkType “transport” enum for supported transports.
- **serialNum** -- Specify a serial number to connect to a specific link module.  
Use 0 to connect to the first link module found.

**Returns**

aErr::aErrBusy - if the module is already in use.

**Returns**

aErr::aErrParam - if the type is incorrect or serialNum is not specified

**Returns**

aErr::aErrNotFound - if the module cannot be found.

**Returns**

aErr::aErrNone If the connect was successful.

---

`BrainStem2CLI::aErr connectThroughLinkModule (BrainStem2CLI::ModuleClass ^module)`

Connect using link from another Module. This member function will connect to the same BrainStem used by given Module. If a link module is found on the specified transport, a connection will

**Parameters**

`module` -- Pointer to a valid Module class object.

**Returns**

`aErr::aErrParam` - if the module is undefined.

**Returns**

`aErr::aErrNone` - if the connect was successful.

`BrainStem2CLI::aErr disconnect ()`

Disconnect from the BrainStem module.

**Returns**

`aErr::aErrResource` - If the there is no valid connection.

**Returns**

`aErr::aErrConnection` - If the disconnect failed, due to a communication issue.

**Returns**

`aErr::aErrNone` If the disconnect was successful.

`BrainStem2CLI::aErr reconnect ()`

Reconnect using the current link specifier.

**Returns**

`aErr::aErrBusy` - if the module is already in use.

**Returns**

`aErr::aErrParam` - if the specifier is incorrect.

**Returns**

`aErr::aErrNotFound` - if the module cannot be found.

**Returns**

`aErr::aErrNone` If the connect was successful.

`bool isConnected ()`

Is the link connected to the BrainStem Module.

`void setModuleAddress (const unsigned char address)`

Accessor to set the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

**Parameters**

`address` -- The module address.

`unsigned char getModuleAddress ()`

Accessor to get the address of the BrainStem module associated with the instance on the host machine. (Not to be confused with the System entity which effects the device hardware.)

**Returns**

The module address.

`void setNetworkingMode (const bool mode)`

Sets the networking mode of the module object. By default the module object is configured to automatically adjust its address based on the devices current module address. So that, if the device has a software or hardware offset it will still be able to communicate with the device. If advanced networking is required the auto networking mode can be turned off.

## Parameters

**mode** – True/1 for Auto Networking, False/0 for manual networking

## Public Static Functions

```
static BrainStem2CLI::m_linkSpec findFirstModule (BrainStem2CLI::m_linkType type)
```

Finds the first module found on the given transport

### Parameters

**type** – The transport type on which to search for devices. Valid m\_linkType “linktypes” are accepted.

### Returns

If found, the linkspec will be populated with the devices information. Values will be all zeros otherwise.

Finds the module with the given serial number on the given transport type.

## Parameters

- **type** – The transport type on which search for devices. Valid m\_linkType “linktypes” are accepted
  - **serialNum** – Specify a serial number to connect to a specific link module. Use 0 to connect to the first link module found.

## Returns

If found the linkspec will be populated with the devices information. Values will be all zeros otherwise.

`sDiscover` is called with a specified transport to search for link modules on that transport. The devices list is filled with device sceptors. `sDiscover` returns `aErrNone` if the discovery process is successful, regardless of whether any links are found. An error is only returned if the link discovery process fails. Discovery can take some time.

## Parameters

- **type** – Transport to search for available BrainStem link modules on. See the `m_linkType` “transport” enum for supported transports.
  - **specList** – an empty list of specifiers that will be filled in.

## Returns

aErr::aErrNotFound if no devices were found.

## Returns

aErr::aErrNone on success

### 3.5.11 Mux Class

```
class MuxClass
```

**MuxClass**. A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexor) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

## Public Functions

**MuxClass** (Acroname::BrainStem::*MuxClass* &rcservo)

Constructor.

**~MuxClass ()**

Destructor.

**!MuxClass ()**

Finalizer.

**aErr getEnable** (unsigned char %bEnabled)

Get the mux enable/disable status

**Parameters**

**bEnabled** – true: mux is enabled, false: the mux is disabled.

**Returns**

Returns common entity return values

**aErr setEnable** (const unsigned char bEnable)

Enable the mux.

**Parameters**

**bEnable** – true: enables the mux for the selected channel.

**Returns**

Returns common entity return values

**aErr getChannel** (unsigned char %channel)

Get the current selected mux channel.

**Parameters**

**channel** – Indicates which channel is selected.

**Returns**

Returns common entity return values

**aErr setChannel** (const unsigned char channel)

Set the current mux channel.

**Parameters**

**channel** – mux channel to select.

**Returns**

Returns common entity return values

**aErr getChannelVoltage** (const unsigned char channel, int %microvolts)

Get the voltage of the indicated mux channel.

**Note:** Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

**Parameters**

- **channel** – The channel in which voltage was requested.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

**Returns**

Returns common entity return values

*aErr* **getConfiguration** (int %config)

Get the configuration of the mux.

**Parameters**

**config** – integer representing the mux configuration either default, or split-mode.

**Returns**

Returns common entity return values

*aErr* **setConfiguration** (const int config)

Set the configuration of the mux.

**Parameters**

**config** – integer representing the mux configuration either muxConfig\_default, or muxConfig\_splitMode.

**Returns**

Returns common entity return values

*aErr* **getSplitMode** (int %splitMode)

Get the current mux mode.

**Parameters**

**splitMode** – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

**Returns**

Returns common entity return values

*aErr* **setSplitMode** (const int splitMode)

Set the current mux mode.

**Parameters**

**splitMode** – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

**Returns**

Returns common entity return values

### 3.5.12 Pointer Class

**class PointerClass**

*PointerClass*. Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

## Public Functions

**PointerClass** (Acroname::BrainStem::*PointerClass* &pointer)

Constructor.

**~PointerClass ()**

Destructor.

**!PointerClass ()**

Finalizer.

**aErr getOffset** (unsigned short %offset)

Get the offset of the pointer

**Parameters**

**offset** – The value of the offset.

**Returns**

All possible standard UEI return values.

**aErr setOffset** (unsigned short offset)

Set the offset of the pointer

**Parameters**

**offset** – The value of the offset.

**Returns**

All possible standard UEI return values.

**aErr getMode** (unsigned char %mode)

Get the mode of the pointer

**Parameters**

**mode** – The mode: aPOINTER\_MODE\_STATIC or  
aPOINTER\_MODE\_AUTO\_INCREMENT.

**Returns**

All possible standard UEI return values.

**aErr setMode** (unsigned char mode)

Set the mode of the pointer

**Parameters**

**mode** – The mode: aPOINTER\_MODE\_STATIC or  
aPOINTER\_MODE\_AUTO\_INCREMENT.

**Returns**

All possible standard UEI return values.

**aErr getTransferStore** (unsigned char %handle)

Get the handle to the store.

**Parameters**

**handle** – The handle of the store.

**Returns**

All possible standard UEI return handles.

**aErr setTransferStore** (unsigned char handle)

Set the handle to the store.

**Parameters**

**handle** – The handle of the store.

**Returns**

All possible standard UEI return handles.

*aErr* **initiateTransferToStore** (unsigned char length)

Transfer data to the store.

**Parameters**

length – The length of the data transfer.

**Returns**

All possible standard UEI return values.

*aErr* **initiateTransferFromStore** (unsigned char length)

Transfer data from the store.

**Parameters**

length – The length of the data transfer.

**Returns**

All possible standard UEI return values.

*aErr* **getChar** (unsigned char %value)

Get a char (1 unsigned char) value from the pointer at this object's index, where elements are 1 unsigned char long.

**Parameters**

value – The value of a single character (1 unsigned char) stored in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **setChar** (const unsigned char value)

Set a char (1 unsigned char) value to the pointer at this object's element index, where elements are 1 unsigned char long.

**Parameters**

value – The single char (1 unsigned char) value to be stored in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **getShort** (unsigned short %value)

Get a short (2 unsigned char) value from the pointer at this objects index, where elements are 2 bytes long

**Parameters**

value – The value of a single short (2 unsigned char) stored in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **setShort** (const unsigned short value)

Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

**Parameters**

value – The single short (2 unsigned char) value to be set in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **getInt** (unsigned int %value)

Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

**Parameters**

value – The value of a single int (4 unsigned char) stored in the pointer.

**Returns**

All possible standard UEI return values.

*aErr* **setInt** (const unsigned int value)

Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

**Parameters**

**value** - The single int (4 unsigned char) value to be stored in the pointer.

**Returns**

All possible standard UEI return values.

### 3.5.13 Port Class

**class PortClass**

Port Class The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

#### Public Functions

**PortClass** (Acroname::BrainStem::*PortClass* &port)

Constructor.

**~PortClass ()**

Destructor.

**!PortClass ()**

Finalizer.

*aErr* **getVbusVoltage** (int %microvolts)

Gets the Vbus Voltage

**Parameters**

**microvolts** - The voltage in microvolts (1 == 1e-6V) currently present on Vbus.

**Returns**

Returns common entity return values

*aErr* **getVbusCurrent** (int %microamps)

Gets the Vbus Current

**Parameters**

**microamps** - The current in microamps (1 == 1e-6A) currently present on Vbus.

**Returns**

Returns common entity return values

*aErr* **getVconnVoltage** (int %microvolts)

Gets the Vconn Voltage

**Parameters**

**microvolts** - The voltage in microvolts (1 == 1e-6V) currently present on Vconn.

**Returns**

Returns common entity return values

`aErr getVconnCurrent (int %microamps)`  
Gets the Vconn Current  
**Parameters**  
    **microamps** – The current in microamps (1 == 1e-6A) currently present on Vconn.  
**Returns**  
    Returns common entity return values

`aErr getPowerMode (unsigned char %powerMode)`  
Gets the Port Power Mode: Convenience Function of get/setPortMode  
**Parameters**  
    **powerMode** – The current power mode.  
**Returns**  
    Returns common entity return values

`aErr setPowerMode (const unsigned char powerMode)`  
Sets the Port Power Mode: Convenience Function of get/setPortMode  
**Parameters**  
    **powerMode** – The power mode to be set.  
**Returns**  
    Returns common entity return values

`aErr getEnabled (unsigned char %enable)`  
Gets the current enable value of the port.  
**Parameters**  
    **enable** – 1 = Fully enabled port; 0 = One or more disabled components.  
**Returns**  
    Returns common entity return values

`aErr setEnabled (const unsigned char enable)`  
Enables or disables the entire port.  
**Parameters**  
    **enable** – 1 = Fully enable port; 0 = Fully disable port.  
**Returns**  
    Returns common entity return values

`aErr getDataEnabled (unsigned char %enable)`  
Gets the current enable value of the data lines.: Sub-component (Data) of getEnabled.  
**Parameters**  
    **enable** – 1 = Data enabled; 0 = Data disabled.  
**Returns**  
    Returns common entity return values

`aErr setDataEnabled (const unsigned char enable)`  
Enables or disables the data lines. Sub-component (Data) of setEnabled.  
**Parameters**  
    **enable** – 1 = Enable data; 0 = Disable data.  
**Returns**  
    Returns common entity return values

`aErr getDataHSEnabled (unsigned char %enable)`  
Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled.  
**Parameters**  
    **enable** – 1 = Data enabled; 0 = Data disabled.

**Returns**

Returns common entity return values

*aErr* **setDataHSEnabled** (const unsigned char enable)

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.

**Parameters**

enable - 1 = Enable data; 0 = Disable data.

**Returns**

Returns common entity return values

*aErr* **getDataHS1Enabled** (unsigned char %enable)

Gets the current enable value of the High Speed A side (HSA) data lines. Sub-component of getDataHSEnabled.

**Parameters**

enable - 1 = Data enabled; 0 = Data disabled.

**Returns**

Returns common entity return values

*aErr* **setDataHS1Enabled** (const unsigned char enable)

Enables or disables the High Speed A side (HSA) data lines. Sub-component of 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 High Speed B side (HSB) data lines. Sub-component of set-DataHSEnabled.

**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 setDataEn-abled.

**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 `setDataEnabled`.

**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 `setEnabled`.

**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 getAlllocatedPower` (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* **getDataHSRoutingBehavior** (unsigned char %mode)

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **setDataHSRoutingBehavior** (const unsigned char mode)

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **getDataSSRoutingBehavior** (unsigned char %mode)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – Variable to be filled with an enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **setDataSSRoutingBehavior** (const unsigned char mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

**mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **getVbusAccumulatedPower** (int %milliwatthours)

Gets the Vbus Accumulated Power

**Parameters**

**milliwatthours** – The accumulated power on Vbus in milliwatt-hours.

**Returns**

Returns common entity return values

*aErr* **resetVbusAccumulatedPower** (void)

Reset the Vbus Accumulated Power

**Returns**

Returns common entity return values

*aErr* **getVconnAccumulatedPower** (int %milliwatthours)

Gets the Vconn Accumulated Power

**Parameters**

**milliwatthours** – The accumulated power on Vconn in milliwatt-hours.

**Returns**

Returns common entity return values

*aErr* **resetVconnAccumulatedPower** (void)

Reset the Vconn Accumulated Power

**Returns**

Returns common entity return values

*aErr* **setHSBoost** (const unsigned char boost)

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Parameters**

**boost** – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **getHSBoost** (unsigned char %boost)

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Parameters**

**boost** – An enumerated representation of the boost range. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

*aErr* **resetEntityToFactoryDefaults** (void)

Resets the *PortClass* Entity to it factory default configuration.

**Returns**

Returns common entity return values

### 3.5.14 Power Delivery Class

**class PowerDeliveryClass**

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

## Public Functions

**PowerDeliveryClass** (Acroname::BrainStem::*PowerDeliveryClass* &pd)

Constructor.

**~PowerDeliveryClass** (void)

Destructor.

**!PowerDeliveryClass ()**

Finalizer.

**aErr getConnectionState** (unsigned char %state)

Gets the current state of the connection in the form of an enumeration.

**Parameters**

- **state** – Pointer to be filled with the current connection state.

**Returns**

Returns common entity return values

**aErr getNumberOfPowerDataObjects** (const unsigned char partner, const unsigned char powerRole, unsigned char %numRules)

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

**Parameters**

- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **numRules** – Variable to be filled with the number of PDOs.

**Returns**

Returns common entity return values

**aErr getPowerDataObject** (const unsigned char partner, const unsigned char powerRole, const unsigned char ruleIndex, unsigned int %pdo)

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

**Parameters**

- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Variable to be filled with the requested power rule.

**Returns**

Returns common entity return values

**aErr setPowerDataObject** (const unsigned char powerRole, const unsigned char ruleIndex, const unsigned int pdo)

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

**Parameters**

- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource

- Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Power Data Object to be set.

**Returns**

Returns common entity return values

*aErr* **resetPowerDataObjectToDefault** (const unsigned char powerRole, const unsigned char ruleIndex)

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

**Parameters**

- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

**Returns**

Returns common entity return values

*aErr* **getPowerDataObjectList** (unsigned int %buffer, const unsigned int bufLength, unsigned int %unloadedLength)

Gets all Power Data Objects (PDOs). Equivalent to calling *PowerDeliveryClass::getPowerDataObject()* on all partners, power roles, and index's.

**Parameters**

- **buffer** – pointer to the start of a c style buffer to be filled The order of which is:
  - Rules 1-7 Local Source
  - Rules 1-7 Local Sink
  - Rules 1-7 Partner Source
  - Rules 1-7 Partner Sink.
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled. On success this value should be 28 (7 rules \* 2 partners \* 2 power roles)

**Returns**

Returns common entity return values

*aErr* **getPowerDataObjectEnabled** (const unsigned char powerRole, const unsigned char ruleIndex, unsigned char %enabled)

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

**Parameters**

- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – Variable to be filled with enabled state.

**Returns**

Returns common entity return values

*aErr* **setPowerDataObjectEnabled** (const unsigned char powerRole, const unsigned char ruleIndex, const unsigned char enabled)

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is

made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

#### Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – The state to be set.

#### Returns

Returns common entity return values

*aErr* **getPowerDataObjectEnabledList** (const unsigned char powerRole, unsigned char %enabledList)

Gets all Power Data Object enables for a given power role. Equivalent of calling *PowerDeliveryClass::getPowerDataObjectEnabled()* for all indexes.

#### Parameters

- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **enabledList** – Variable to be filled with a mapped representation of the enabled PDOs for a given power role. Values align with a given rule index (bits 1-7, bit 0 is invalid)

#### Returns

Returns common entity return values

*aErr* **getRequestDataObject** (const unsigned char partner, unsigned int %rdo)

Gets the current Request Data Object (RDO) for a given partner. RDOs: Are provided by the sinking device. Exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

#### Parameters

- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **rdo** – Variable to be filled with the current RDO. Zero indicates the RDO is not active.

#### Returns

Returns common entity return values

*aErr* **setrequestDataObject** (const unsigned char partner, const unsigned int rdo)

Sets the current Request Data Object (RDO) for a given partner. (Only the local partner can be changed.) RDOs: Are provided by the sinking device. Exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

#### Parameters

- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
- **rdo** – Request Data Object to be set.

#### Returns

Returns common entity return values

*aErr* **getPowerRole** (unsigned char %powerRole)

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

#### Parameters

- **powerRole** – Variable to be 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* `get CableSpeedMax` (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* `get CableType` (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* `get CableOrientation` (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 (0)

**Returns**

Returns common entity return values

*aErr* `request` (const unsigned char request)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

**Parameters**

`request` – Request to be issued to the remote partner

- `pdRequestHardReset` (0)
- `pdRequestSoftReset` (1)
- `pdRequestDataReset` (2)
- `pdRequestPowerRoleSwap` (3)
- `pdRequestPowerFastRoleSwap` (4)
- `pdRequestDataRoleSwap` (5)
- `pdRequestVconnSwap` (6)
- `pdRequestSinkGoToMinimum` (7)
- `pdRequestRemoteSourcePowerDataObjects` (8)
- `pdRequestRemoteSinkPowerDataObjects` (9)

**Returns**

The returned error represents the success of the request being sent to the partner only. The success of the request being serviced by the remote partner can be obtained through `PowerDeliveryClass::requestStatus()`. Returns common entity return values

**aErr requestStatus (unsigned int %status)**

Gets the status of the last request command sent.

**Parameters**

`status` – Variable to be filled with the status

**Returns**

Returns common entity return values

**aErr getOverride (unsigned int %overrides)**

Gets the current enabled overrides

**Parameters**

`overrides` – Bit mapped representation of the current override configuration.

**Returns**

Returns common entity return values

**aErr setOverride (const unsigned int overrides)**

Sets the current enabled overrides

**Parameters**

`overrides` – Overrides to be set in a bit mapped representation.

**Returns**

Returns common entity return values

**aErr resetEntityToFactoryDefaults (void)**

Resets the `PowerDeliveryClass` Entity to it factory default configuration.

**aErr getFlagMode (const unsigned char flag, unsigned char %mode)**

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters**

- `flag` – Flag/Advertisement to be modified
- `mode` – Variable to be filled with the current mode.
  - Disabled (0)
  - Enabled (1)
  - Auto (2) default

**Returns**

Returns common entity return values

**aErr setFlagMode (const unsigned char flag, const unsigned char mode)**

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters**

- `flag` – Flag/Advertisement to be modified
- `mode` – Value to be applied.
  - Disabled (0)
  - Enabled (1)
  - Auto (2) default

**Returns**

Returns common entity return values

**aErr getPeakCurrentConfiguration (unsigned char %configuration)**

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

**Parameters**

**configuration** – An enumerated value referring to the current configuration.

- Allowable values are 0 - 4

**Returns**

Returns common entity return values

**aErr setPeakCurrentConfiguration (const unsigned char configuration)**

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

**Parameters**

**configuration** – An enumerated value referring to the configuration to be set

- Allowable values are 0 - 4

**Returns**

Returns common entity return values

**aErr getFastRoleSwapCurrent (unsigned char %swapCurrent)**

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

**Parameters**

**swapCurrent** – An enumerated value referring to current swap value.

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

**Returns**

Returns common entity return values

**aErr setFastRoleSwapCurrent (const unsigned char swapCurrent)**

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

**Parameters**

**swapCurrent** – An enumerated value referring to value to be set.

- 0A (0)
- 900mA (1)
- 1.5A (2)
- 3A (3)

**Returns**

Returns common entity return values

### 3.5.15 Rail Class

```
class RailClass
```

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

#### Public Functions

```
RailClass (Acroname::BrainStem::RailClass &pointer)
```

Constructor.

```
~RailClass ()
```

Destructor.

```
!RailClass ()
```

Finalizer.

```
aErr getCurrent (int %microamps)
```

Get the rail current.

##### Parameters

**microamps** – The current in micro-amps (1 == 1e-6A).

##### Returns

Returns common entity return values

```
aErr setCurrentSetpoint (const int microamps)
```

Set the rail setpoint current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

##### Parameters

**microamps** – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

##### Returns

Returns common entity return values

```
aErr getCurrentSetpoint (int %microamps)
```

Get the rail setpoint current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

##### Parameters

**microamps** – The current in micro-amps (1 == 1e-6A) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setCurrent interface. Refer to the module datasheet to 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 ( $1 == 1e-6^{\circ}C$ ). The temperature may be associated with the module's internal rail circuitry or an externally connected temperature sensors. Refer to the module datasheet for definition of the temperature measurement location and specific capabilities.

**Returns**

Returns common entity return values

***aErr getEnable (unsigned char %bEnable)***

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

**Parameters**

**bEnable** – true: enabled: connected to the supply rail voltage; false: disabled: disconnected from the supply rail voltage

**Returns**

Returns common entity return values

***aErr setEnable (const unsigned char bEnable)***

Set the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

**Parameters**

**bEnable** – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

**Returns**

Returns common entity return values

***aErr getVoltage (int %microvolts)***

Get the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

**microvolts** – The voltage in micro-volts ( $1 == 1e-6V$ ) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

***aErr setVoltageSetpoint (const int microvolts)***

Set the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

**microvolts** – The voltage in micro-volts ( $1 == 1e-6V$ ) to be supply by the rail.

**Returns**

Returns common entity return values

**aErr getVoltageSetpoint** (int %microvolts)

Get the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

**microvolts** - The voltage in micro-volts (1 == 1e-6V) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setVoltage interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

**aErr setVoltageMinLimit** (const int microvolts)

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

**microvolts** - The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

**aErr getVoltageMinLimit** (int %microvolts)

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

**microvolts** - The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

**aErr setVoltageMaxLimit** (const int microvolts)

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

**microvolts** - The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

**aErr getVoltageMaxLimit** (int %microvolts)

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

**microvolts** - The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

**aErr getPower** (int %milliwatts)

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

**milliwatts** - The power in milli-watts (1 == 1e-3W) currently supplied by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

***aErr setPowerSetpoint*** (const int milliwatts)

Set the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

**milliwatts** – The power in milli-watts (1 == 1e-3W) to be supplied by the rail.

**Returns**

Returns common entity return values

***aErr getPowerSetpoint*** (int %milliwatts)

Get the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

**milliwatts** – The power in milli-watts (1 == 1e-3W) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the setPower interface. Refer to the module datasheet to 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 %millionohms)

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**

**millionohms** – The resistance in milli-ohms (1 == 1e-3Ohms) currently drawn by the rail. On some modules this is a measured value so it may not exactly match what was previously set via the setResistance interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

***aErr setResistanceSetpoint*** (const int millionohms)

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**

**millionohms** – The power in milli-ohms (1 == 1e-3Ohms) to be drawn by the rail.

**Returns**

Returns common entity return values

`aErr getResistanceSetpoint (int %milliohms)`

Get the rail setpoint resistance. Rail resistance control capabilities vary between modules.  
Refer to the module datasheet for definition of the rail resistance capabilities.

**Parameters**

`milliohms` – The resistance in milli-ohms (1 == 1e-3Ohms) the rail is trying to achieve. On some modules this is a measured value so it may not exactly match what was previously set via the `setResistance` interface. Refer to the module datasheet to determine if this is a measured or stored value.

**Returns**

Returns common entity return values

`aErr setKelvinSensingEnable (const unsigned char bEnable)`

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

`bEnable` – enable or disable kelvin sensing.

**Returns**

Returns common entity return values

`aErr getKelvinSensingEnable (unsigned char %bEnable)`

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

`bEnable` – Kelvin sensing is enabled or disabled.

**Returns**

Returns common entity return values

`aErr getKelvinSensingState (unsigned char %state)`

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

`state` – Kelvin sensing is enabled or disabled.

**Returns**

Returns common entity return values

`aErr setOperationalMode (const unsigned char mode)`

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

**Parameters**

`mode` – The operational mode to employ.

**Returns**

Returns common entity return values

`aErr getOperationalMode (unsigned char %mode)`

Determine the current operational mode of the system. Refer to the module datasheet for definition of the rail operational mode capabilities.

**Parameters**

`mode` – The current operational mode setting.

**Returns**

Returns common entity return values

`aErr getOperationalState (unsigned int %state)`

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

**Parameters**

`state` – The current operational state.

**Returns**

Returns common entity return values

*aErr* **clearFaults** (void)

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

**Returns**

Returns common entity return values

### 3.5.16 RCServo Class

**class RCServoClass**

The *RCServoClass* is the interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

#### Public Functions

**RCServoClass** (Acroname::BrainStem::*RCServoClass* &rcservo)

Constructor.

**~RCServoClass ()**

Destructor.

**!RCServoClass ()**

Finalizer.

*aErr* **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

### 3.5.17 Relay Class

class **RelayClass**

The *RelayClass* is the interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

#### Public Functions

**RelayClass** (Acroname::BrainStem::*RelayClass* &relay)

Constructor.

**~RelyClass ()**

Destructor.

**!RelyClass ()**

Finalizer.

*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 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

### 3.5.18 Signal Class

See the *Signal Entity* for generic information.

class **SignalClass**

*SignalClass* is the interface to digital pins configured to produce square wave signals.

This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

#### Public Functions

**SignalClass** (Acroname::BrainStem::*SignalClass* &signal)

Constructor.

**~SignalClass ()**

Destructor.

**!SignalClass ()**

Finalizer.

*aErr* **setEnable** (const unsigned char enable)

Enable/Disable the signal output.

**Parameters**

**enable** – True to enable, false to disable

**Returns**

Returns common entity return values

`aErr getEnable (unsigned char %enable)`

Get the Enable/Disable of the signal.

**Parameters**

`enable` – True to enable, false to disable

**Returns**

Returns common entity return values

`aErr setInvert (const unsigned char invert)`

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

**Parameters**

`invert` – True to invert, false for normal mode.

**Returns**

Returns common entity return values

`aErr getInvert (unsigned char %invert)`

Get the invert status the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

**Parameters**

`invert` – True to invert, false for normal mode.

**Returns**

Returns common entity return values

`aErr setT3Time (const int t3_nsec)`

Set the signal period or T3 in nanoseconds.

**Parameters**

`t3_nsec` – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

**Returns**

Returns common entity return values

`aErr getT3Time (unsigned int %t3_nsec)`

Get the signal period or T3 in nanoseconds.

**Parameters**

`t3_nsec` – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

**Returns**

Returns common entity return values

`aErr setT2Time (const int t2_nsec)`

Set the signal active period or T2 in nanoseconds.

**Parameters**

`t2_nsec` – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

**Returns**

Returns common entity return values

`aErr getT2Time (unsigned int %t2_nsec)`

Get the signal active period or T2 in nanoseconds.

**Parameters**

`t2_nsec` – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

**Returns**

Returns common entity return values

### 3.5.19 Store Class

**class StoreClass**

*StoreClass*. The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

#### Public Functions

**StoreClass (Acroname::BrainStem::StoreClass &store)**

Constructor.

**~StoreClass ()**

Destructor.

**!StoreClass ()**

Finalizer.

**aErr getSlotState (const unsigned char slot, unsigned char %state)**

Get slot state.

**Parameters**

- **slot** - The slot number.
- **state** - true: enabled, false: disabled.

**Returns**

Returns common entity return values

**aErr loadSlot (const unsigned char slot, const unsigned char %pData, const unsigned short length)**

Load the slot.

**Parameters**

- **slot** - The slot number.
- **pData** - The data.
- **length** - The data length.

**Returns**

Returns common entity return values

**aErr unloadSlot (const unsigned char slot, const unsigned int dataLength, unsigned char %pData, unsigned int %unloadedLength)**

Unload the slot data.

**Parameters**

- **pData** - Byte array that the unloaded data will be placed into.
- **dataLength** - - The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.

- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than dataLength.
- **slot** – The slot number.

**Returns**

Returns common entity return values

*aErr* **slotEnable** (const unsigned char slot)

Enable slot.

**Parameters**

- **slot** – The slot number.

**Returns**

Returns common entity return values

*aErr* **slotDisable** (const unsigned char slot)

Disable slot.

**Parameters**

- **slot** – The slot number.

**Returns**

Returns common entity return values

*aErr* **getSlotCapacity** (const unsigned char slot, unsigned int %capacity)

Get the slot capacity.

**Parameters**

- **slot** – The slot number.
- **capacity** – The slot capacity.

**Returns**

Returns common entity return values

*aErr* **getSlotSize** (const unsigned char slot, unsigned int %size)

Get the slot size

**Parameters**

- **slot** – The slot number.
- **size** – The slot size.

**Returns**

Returns common entity return values

### 3.5.20 System Class

class **SystemClass**

*SystemClass*. The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc. The most common brainstem example uses the system entity to blink the User LED.

## Public Functions

**SystemClass** (Acroname::BrainStem::*SystemClass* &system)

Constructor.

**~SystemClass ()**

Destructor.

**!SystemClass ()**

Finalizer.

**aErr getModule** (unsigned char %address)

Get the current address the module uses on the BrainStem network.

**Parameters**

**address** – The address the module is using on the BrainStem network.

**Returns**

    Returns common entity return values

**aErr getModuleBaseAddress** (unsigned char %address)

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

**Parameters**

**address** – The address the module is using on the BrainStem network.

**Returns**

    Returns common entity return values

**aErr setRouter** (const unsigned char address)

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

**Parameters**

**address** – The router address to be used.

**Returns**

    Returns common entity return values

**aErr getRouter** (unsigned char %address)

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

**Parameters**

**address** – The address.

**Returns**

    Returns common entity return values

**aErr setHBInterval** (const unsigned char interval)

Set the delay between heartbeat packets which are sent from the module. For link modules, these heartbeats are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

**Parameters**

**interval** – The desired heartbeat delay.

**Returns**

    Returns common entity return values

*aErr* **getHBInterval** (unsigned char %interval)

Get the delay between heartbeat packets which are sent from the module. For link modules, these these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

**Parameters**

**interval** – The current heartbeat delay.

**Returns**

    Returns common entity return values

*aErr* **setLED** (const unsigned char bOn)

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

**Parameters**

**bOn** – true: turn the LED on, false: turn LED off.

**Returns**

    Returns common entity return values

*aErr* **getLED** (unsigned char %bOn)

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

**Parameters**

**bOn** – true: LED on, false: LED off.

**Returns**

    Returns common entity return values

*aErr* **setBootSlot** (const unsigned char slot)

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

**Parameters**

**slot** – The slot number in aSTORE\_INTERNAL to be marked as a boot slot.

**Returns**

    Returns common entity return values

*aErr* **getBootSlot** (unsigned char %slot)

Get the store slot which is mapped when the module boots.

**Parameters**

**slot** – The slot number in aSTORE\_INTERNAL that is mapped after the module boots.

**Returns**

    Returns common entity return values

*aErr* **getVersion** (unsigned int %build)

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

**Parameters**

**build** – The build version date code.

*aErr* **getModel** (unsigned char %model)

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

**Parameters**

**model** – The module's model enumeration.

**Returns**

Returns common entity return values

**aErr getHardwareVersion** (unsigned int %hardwareVersion)

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

**Parameters**

**hardwareVersion** – The module's hardware version information.

**Returns**

Returns common entity return values

**aErr getSerialNumber** (unsigned int %serialNumber)

Get the module's serial number. The serial number is a unique 32bit integer which is usually communicated in hexadecimal format.

**Parameters**

**serialNumber** – The module's serial number.

**Returns**

Returns common entity return values

**aErr save** (void)

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heartbeat interval, module address, module router address

**Returns**

Returns common entity return values

**aErr reset** (void)

Reset the system.

**Returns**

Returns common entity return values

**aErr logEvents** (void)

Saves system log events to a slot defined by the module (usually ram slot 0).

**Returns**

Returns common entity return values

**aErr getUptime** (unsigned int %uptimeCounter)

Get the module's accumulated uptime in minutes

**Parameters**

**uptimeCounter** – The module's accumulated uptime in minutes.

**Returns**

Returns common entity return values

**aErr getTemperature** (int %temperature)

Get the module's current temperature in micro-C

**Parameters**

**temperature** – The module's system temperature in micro-C

**Returns**

Returns common entity return values

**aErr getMinimumTemperature** (int %minTemperature)

Get the module's minimum temperature in micro-C

**Parameters**

**minTemperature** – The module's minimum system temperature in micro-C

**Returns**

Returns common entity return values

*aErr* **getMaximumTemperature** (int %maxTemperature)

Get the module's maximum temperature in micro-C

**Parameters**

**maxTemperature** – The module's maximum system temperature in micro-C

**Returns**

Returns common entity return values

*aErr* **getInputVoltage** (unsigned int %inputVoltage)

Get the module's input voltage.

**Parameters**

**inputVoltage** – The module's input voltage reported in microvolts.

**Returns**

Returns common entity return values

*aErr* **getInputCurrent** (unsigned int %inputCurrent)

Get the module's input current.

**Parameters**

**inputCurrent** – The module's input current reported in microamps.

**Returns**

Returns common entity return values

*aErr* **getModuleHardwareOffset** (unsigned char %offset)

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

**Parameters**

**offset** – The module address offset.

**Returns**

Returns common entity return values

*aErr* **setModuleSoftwareOffset** (const unsigned char address)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

**Parameters**

**address** – The address for the module. Value must be even from 0-254.

**Returns**

Returns common entity return values

*aErr* **getModuleSoftwareOffset** (unsigned char %address)

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

**Parameters**

**address** – The address for the module. Value must be even from 0-254.

**Returns**

Returns common entity return values

*aErr* **getRouterAddressSetting** (unsigned char %address)

Get the router address system setting. This setting may not be the same as the current

router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

**Parameters**

**address** – The address for the module. Value must be even from 0-254.

**Returns**

Returns common entity return values

**aErr routeToMe (const unsigned char bOn)**

Enables/Disables the route to me function. This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

**Parameters**

**bOn** – Enable or disable of the route to me function 1 = enable.

**Returns**

Returns common entity return values

**aErr getErrors (unsigned int %errors)**

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

**Parameters**

**errors** – Bit mapped field representing the devices errors

**Returns**

Returns common entity return values

### 3.5.21 Temperature Class

**class TemperatureClass**

*TemperatureClass*. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

#### Public Functions

**TemperatureClass (Acroname::BrainStem::TemperatureClass &temperature)**

Constructor.

**~TemperatureClass ()**

Destructor.

**!TemperatureClass ()**

Finalizer.

**aErr getValue (int %microcelsius)**

Get the temperature.

**Parameters**

**microcelsius** – The temperature in micro-Celsius (1 == 1e-6C).

**Returns**

Returns common entity return values

`aErr getValueMin (int %minTemperature)`  
Get the module's minimum temperature in micro-C  
**Parameters**  
    `minTemperature` – The module's minimum system temperature in micro-C  
**Returns**  
    Returns common entity return values

`aErr getValueMax (int %maxTemperature)`  
Get the module's maximum temperature in micro-C  
**Parameters**  
    `maxTemperature` – The module's maximum system temperature in micro-C  
**Returns**  
    Returns common entity return values

### 3.5.22 Timer Class

`class TimerClass`

*TimerClass*. The Timer Class provides access to a simple scheduler. Reflex routines can be written which will be executed upon expiration of the timer entity. The timer can be set to fire only once, or to repeat at a certain interval.

#### Public Functions

`TimerClass (Acroname::BrainStem::TimerClass &timer)`

Constructor.

`~TimerClass ()`

Destructor.

`!TimerClass ()`

Finalizer.

`aErr getExpiration (unsigned int %usecDuration)`

Get the currently set expiration time in microseconds. This is not a “live” timer. That is, it shows the expiration time originally set with `setExpiration`; it does not “tick down” to show the time remaining before expiration.

##### Parameters

`usecDuration` – The timer expiration duration in microseconds.

##### Returns

    Returns common entity return values

`aErr setExpiration (const int usecDuration)`

Set the expiration time for the timer entity. When the timer expires, it will fire the associated `timer[index]()` reflex.

##### Parameters

`usecDuration` – The duration before timer expiration in microseconds.

##### Returns

    Returns common entity return values

`aErr getMode (unsigned char %mode)`

Get the mode of the timer which is either single or repeat mode.

**Parameters**

`mode` - The mode of the time. aTIMER\_MODE\_REPEAT or aTIMER\_MODE\_SINGLE.

**Returns**

Returns common entity return values

`aErr setMode (const unsigned char mode)`

Set the mode of the timer which is either single or repeat mode.

**Parameters**

`mode` - The mode of the timer. aTIMER\_MODE\_REPEAT or aTIMER\_MODE\_SINGLE.

**Returns**

Returns common entity return values

**Returns**

`aErr::aErrNone` - Action completed successfully.

### 3.5.23 UART Class

`class UARTClass`

`UARTClass`. A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines.

#### Public Functions

`UARTClass (Acroname::BrainStem::UARTClass &uart)`

Constructor.

`~UARTClass ()`

Destructor.

`!UARTClass ()`

Finalizer.

`aErr 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

### 3.5.24 USB Class

class **USBClass**

**USBClass**. The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

#### Public Functions

**USBClass** (Acroname::BrainStem::*USBClass* &**USB**)

Constructor.

**~USBClass** ()

Destructor.

**!USBClass** ()

Finalizer.

*aErr* **setPortEnable** (const unsigned char **channel**)

Enable both power and data lines for a port.

**Parameters**

**channel** – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setPortDisable** (const unsigned char channel)

Disable both power and data lines for a port.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setHiSpeedDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setHiSpeedDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setSuperSpeedDataEnable** (const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setSuperSpeedDataDisable** (const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, Super-Speed (3.0) only.

**Parameters**

channel – The USB sub channel.

**Returns**

Returns common entity return values

*aErr* **setPowerEnable** (const unsigned char channel)

Enable only the power line for a port without changing the state of the data lines.

**Parameters**

**channel** – The USB sub channel.

**Returns**

    Returns common entity return values

*aErr* **setPowerDisable** (const unsigned char **channel**)

Disable only the power line for a port without changing the state of the data lines.

**Parameters**

**channel** – The USB sub channel.

**Returns**

    Returns common entity return values

*aErr* **getPortCurrent** (const unsigned char **channel**, int %**microamps**)

Get the current through the power line for a port.

**Parameters**

- **channel** – The USB sub channel.

- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

**Returns**

    Returns common entity return values

*aErr* **getPortVoltage** (const unsigned char **channel**, int %**microvolts**)

Get the voltage on the power line for a port.

**Parameters**

- **channel** – The USB sub channel.

- **microvolts** – The USB channel voltage in microvolts (1 == 1e-6V).

**Returns**

    Returns common entity return values

*aErr* **getHubMode** (unsigned int %**state**)

Get a bit mapped representation of the hub mode; see the product datasheet for state mapping. Usually represents the hub's downstream ports data and power line enable/disable state.

**Parameters**

**state** – The USB hub state.

**Returns**

    Returns common entity return values

*aErr* **setHubMode** (const unsigned int **state**)

Set a bit mapped hub state; see the product datasheet for state mapping. Usually represents the hub's downstream ports data and power line enable/disable state.

**Parameters**

**state** – The USB hub state.

**Returns**

    Returns common entity return values

*aErr* **clearPortErrorStatus** (const unsigned char **channel**)

Clear the error status for the given channel.

**Parameters**

**channel** – the port to clear error status for.

**Returns**

    Returns common entity return values

*aErr* **getUpstreamMode** (unsigned char %**mode**)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

**Parameters**

- mode** – The Upstream port mode.

**Returns**

Returns common entity return values

*aErr* **setUpstreamMode** (const unsigned char **mode**)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0 and usbUpstreamModePort1

**Parameters**

- mode** – The Upstream port mode.

**Returns**

Returns common entity return values

*aErr* **getUpstreamState** (unsigned char %**state**)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

**Parameters**

- state** – The Upstream port state.

**Returns**

Returns common entity return values

*aErr* **setEnumerationDelay** (const unsigned int **ms\_delay**)

Set the interport enumeration delay in milliseconds. This setting should be saved with a `stem.system.save()` call.

**Parameters**

- ms\_delay** – 100ms delay increment.

**Returns**

Returns common entity return values

*aErr* **getEnumerationDelay** (unsigned int %**ms\_delay**)

Get the interport enumeration delay.

**Parameters**

- ms\_delay** – 100ms delay increment.

**Returns**

Returns common entity return values

*aErr* **setPortCurrentLimit** (const unsigned char **channel**, const unsigned int **microamps**)

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a `stem.system.save()` call.

**Parameters**

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

**Returns**

Returns common entity return values

*aErr* **getPortCurrentLimit** (const unsigned char **channel**, unsigned int %**microamps**)

Get the current limit for the port. This reflects the limit setting currently in effect.

**Parameters**

- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

**Returns**

Returns common entity return values

*aErr* **setPortMode** (const unsigned char channel, const unsigned int mode)

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device datasheet for a complete list of capabilities. There is a unified bit mapping for port mode at `usbPortMode`

**Parameters**

- `channel` – USB downstream channel to set the mode on.
- `mode` – The port mode setting as packet bit mask.

**Returns**

Returns common entity return values

*aErr* **getPortMode** (const unsigned char channel, unsigned int %mode)

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device datasheet for a complete list of capabilities. There is a unified bit mapping for port mode at `usbPortMode`

**Parameters**

- `channel` – USB downstream channel.
- `mode` – The port mode setting. Mode will be filled with the current setting.  
Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

*aErr* **getPortState** (const unsigned char channel, unsigned int %state)

Get the current State for the Port.

**Parameters**

- `channel` – USB downstream channel.
- `state` – The port mode setting. Mode will be filled with the current setting.  
Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

*aErr* **getPortError** (const unsigned char channel, unsigned int %error)

Get the current error for the Port.

**Parameters**

- `channel` – USB downstream channel.
- `error` – The port mode setting. Mode will be filled with the current setting.  
Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

*aErr* **setUpstreamBoostMode** (const unsigned char setting)

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through "pogo" pins. Possible modes are 0 - no boost, 1 - 4\* boost, 2 - 8\* boost, 3 - 12\* boost. This setting is not applied until a `stem.system.save()` call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0\* boost is restored.

**Parameters**

- `setting` – Upstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

**aErr setDownstreamBoostMode** (const unsigned char setting)

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4\* boost, 2 - 8\* boost, 3 - 12\* boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0\* boost is restored.

**Parameters**

**setting** – Downstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

**aErr getUpstreamBoostMode** (unsigned char %setting)

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4\* boost, 2 - 8\* boost, 3 - 12\* boost.

**Parameters**

**setting** – The current Upstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

**aErr getDownstreamBoostMode** (unsigned char %setting)

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4\* boost, 2 - 8\* boost, 3 - 12\* boost.

**Parameters**

**setting** – The current Downstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

**aErr getDownstreamDataSpeed** (const unsigned char channel, unsigned char %speed)

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

**Parameters**

- **channel** – USB downstream channel to check.
- **speed** – Filled with the current port data speed
  - N/A: usbDownstreamDataSpeed\_na = 0
  - Hi Speed: usbDownstreamDataSpeed\_hs = 1
  - SuperSpeed: usbDownstreamDataSpeed\_ss = 2

**Returns**

Returns common entity return values

**aErr setConnectMode** (const unsigned char channel, const unsigned char mode)

Sets the connect mode of the switch.

**Parameters**

- **channel** – The USB sub channel.
- **mode** – The connect mode
  - usbManualConnect = 0
  - usbAutoConnect = 1

**Returns**

Returns common entity return values

**aErr getConnectMode** (const unsigned char channel, unsigned char %mode)

Gets the connect mode of the switch.

**Parameters**

- **channel1** – 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**

- **channel1** -- 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**

- **channel1** -- 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**

- **channel1** -- 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**

- **channel1** -- 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**

- **channel1** – 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**

- **channel1** – 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**

- **channel1** – 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**

- **channel1** – 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**

- **channel1** – 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**

- **channel1** – 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**

- **channel1** – 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**

- **channel1** – The USB sub channel.
- **pEnable** – The enable/disable status of cable flip.

**Returns**

Returns common entity return values

*aErr* **setAltModeConfig** (const unsigned char channel, const unsigned int configuration)

Set USB Alt Mode Configuration.

**Parameters**

- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

**Returns**

Returns common entity return values

*aErr* **getAltModeConfig** (const unsigned char channel, unsigned int %configuration)

Get USB Alt Mode Configuration.

**Parameters**

- **channel** – The USB sub channel
- **configuration** – The USB configuration for the given channel.

**Returns**

Returns common entity return values

*aErr* **getSBU1Voltage** (const unsigned char channel, int %microvolts)

Get the voltage of SBU1 for a port.

**Parameters**

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

*aErr* **getSBU2Voltage** (const unsigned char channel, int %microvolts)

Get the voltage of SBU2 for a port.

**Parameters**

- **channel** – The USB sub channel.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

### 3.5.25 **USBSystem Class**

**class USBSystemClass**

USBSystem Class The USBSystem class provides high level control of the lower level Port Class.

#### Public Functions

**USBSystemClass** (Acroname::BrainStem::*USBSystemClass* &usb)

Constructor.

**~USBSystemClass** ()

Destructor.

**!USBSystemClass** ()

Finalizer.

`aErr getUpstream (unsigned char %port)`

Gets the upstream port.

**Parameters**

`port` – The current upstream port.

**Returns**

Returns common entity return values

`aErr setUpstream (const unsigned char port)`

Sets the upstream port.

**Parameters**

`port` – The upstream port to set.

**Returns**

Returns common entity return values

`aErr getEnumerationDelay (unsigned int %msDelay)`

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

**Parameters**

`msDelay` – the current inter-port delay in milliseconds.

**Returns**

Returns common entity return values

`aErr setEnumerationDelay (const unsigned int msDelay)`

Sets the inter-port enumeration delay in milliseconds. This setting should be saved with a `stem.system.save()` call. Delay is applied upon hub enumeration.

**Parameters**

`msDelay` – The delay in milliseconds to be applied between port enables

**Returns**

Returns common entity return values

`aErr getDataRoleList (unsigned int %roleList)`

Gets the data role of all ports with a single call. Equivalent to calling `Port-Class::getDataRole()` on each individual port.

**Parameters**

`roleList` – A bit packed representation of the data role for all ports.

**Returns**

Returns common entity return values

`aErr getEnabledList (unsigned int %enabledList)`

Gets the current enabled status of all ports with a single call. Equivalent to calling `Port-Class::setEnabled()` on each port.

**Parameters**

`enabledList` – Bit packed representation of the enabled status for all ports.

**Returns**

Returns common entity return values

`aErr setEnabledList (const unsigned int enabledList)`

Sets the enabled status of all ports with a single call. Equivalent to calling `Port-Class::setEnabled()` on each port.

**Parameters**

`enabledList` – Bit packed representation of the enabled status for all ports to be applied.

**Returns**

Returns common entity return values

**aErr** **getModeList** (unsigned int %buffer, const unsigned int bufLength, unsigned int %unloadedLength)

Gets the current mode of all ports with a single call. Equivalent to calling *PortClass::getMode()* on each port.

**Parameters**

- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be 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

## 3.6 LabVIEW API Reference

Welcome to the BrainStem LabVIEW API reference documentation. This documentation covers the LabVIEW Acroname BrainStem library. This reference assumes that you understand the BrainStem system. If you would like to get started using BrainStem, please see the following sections of the Reference documentation.

- [BrainStem Overview](#)
- [BrainStem Terminology](#)
- [Getting Started with the BrainStem](#).

The Getting started guide is particularly useful for learning how to use the application tools we provide to communicate with your hardware.

---

### 3.6.1 Entities

#### `group AnalogEntity`

AnalogClass. Interface to analog entities on BrainStem modules. Analog entities may be configured as 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.

#### `group AppEntity`

AppClass. Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

#### `group ClockEntity`

ClockClass. Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

---

**Note:** Clock time must be reset if power to the BrainStem module is lost.

---

#### `group DigitalEntity`

DigitalClass. Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCServo Input, RCServo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

#### `group EqualizerEntity`

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

#### `group I2CEntity`

I2CClass. Interface the I2C buses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entities bus.

**group ModuleEntity**

The Module Entity provides a generic interface to a BrainStem hardware module. The Module Class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

**group MuxEntity**

MuxClass. A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

**group PointerEntity**

PointerClass. Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

**group RailEntity**

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

**group RCServoEntity**

RCServoClass. Interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

**group RelayEntity**

RelayClass. Interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

**group SignalEntity**

SignalClass. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

**group StoreEntity**

StoreClass. The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

**group SystemEntity**

SystemClass. The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc. The most common brainstem example uses the system entity to blink the User LED.

**group TemperatureEntity**

TemperatureClass. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

**group TimerEntity**

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

**group UARTEntity**

UART Class. A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines.

**group USBEntity**

USBClass. The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

## 3.6.2 Analog Entity

**group AnalogEntity**

AnalogClass. Interface to analog entities on BrainStem modules. Analog entities may be configured as a input or output depending on hardware capabilities. Some modules are capable of providing actual voltage readings, while other simply return the raw analog-to-digital converter (ADC) output value. The resolution of the voltage or number of useful bits is also hardware dependent.

**void analog\_getValue (unsigned int \*id, struct Result \*result, const int index)**

Get the raw ADC output value in bits.

### Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **analog\_getVoltage** (unsigned int \*id, struct Result \*result, const int index)

Get the scaled micro volt value with reference to ground.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **analog\_getRange** (unsigned int \*id, struct Result \*result, const int index)

Get the analog input range.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **analog\_getEnable** (unsigned int \*id, struct Result \*result, const int index)

Get the analog output enable status.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **analog\_setValue** (unsigned int \*id, struct Result \*result, const int index, const int value)

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

---

**Note:** Not all modules are provide 16 useful bits; the least significant bits are discarded. E.g. for a 10 bit DAC, 0xFFC0 to 0x0040 is the useful range. Refer to the module's datasheet to determine analog bit depth and reference voltage.

---

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – 16 bit analog set point with 0 corresponding to the negative analog voltage reference and 0xFFFF corresponding to the positive analog voltage reference.

**Returns**

Returns common entity return values

**void analog\_setVoltage (unsigned int \*id, struct Result \*result, const int index, const int microvolts)**

Set the voltage level of an analog output (DAC) in microvolts.

**Note:** Voltage range is dependent on the specific DAC channel range.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

**Returns**

Returns common entity return values

**void analog\_setRange (unsigned int \*id, struct Result \*result, const int index, const unsigned char range)**

Set the analog input range.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **range** – 8 bit value corresponding to a discrete range option

**Returns**

Returns common entity return values

**void analog\_setEnable (unsigned int \*id, struct Result \*result, const int index, const unsigned char enable)**

Set the analog output enable state.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – set 1 to enable or 0 to disable.

**Returns**

Returns common entity return values

```
void analog_setConfiguration(unsigned int *id, struct Result *result, const int index, const unsigned char configuration)
```

Set the analog configuration.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **configuration** -- bitAnalogConfigurationOutput configures the analog entity as an output.

#### Return values

aErrConfiguration -- Entity does not support this configuration.

#### Returns

EntityReturnValues “common entity” return values

```
void analog_getConfiguration(unsigned int *id, struct Result *result, const int index)
```

Get the analog configuration.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void analog_setBulkCaptureSampleRate(unsigned int *id, struct Result *result, const int index, const unsigned int value)
```

Set the sample rate for this analog when bulk capturing.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – sample rate in samples per second (Hertz). Minimum rate: 7,000 Hz Maximum rate: 200,000 Hz

#### Returns

Returns common entity return values

```
void analog_getBulkCaptureSampleRate(unsigned int *id, struct Result *result, const int index)
```

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

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void analog_setBulkCaptureNumberOfSamples (unsigned int *id, struct Result *result, const int index,
                                         const unsigned int value)
```

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

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – number of samples. Minimum # of Samples: 0 Maximum # of Samples: (BRAINSTEM\_RAM\_SLOT\_SIZE / 2) = (3FFF / 2) = 1FFF = 8191

**Returns**

Returns common entity return values

```
void analog_getBulkCaptureNumberOfSamples (unsigned int *id, struct Result *result, const int index)
```

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

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void analog_initiateBulkCapture (unsigned int *id, struct Result *result, const int index)
```

Initiate a BulkCapture on this analog. Captured measurements are stored in the module's RAM store (RAM\_STORE) slot 0. Data is stored in a contiguous byte array with each sample stored in two consecutive bytes, LSB first.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values. When the bulk capture is complete getBulkCaptureState() will return either bulkCaptureFinished or bulkCaptureError.

```
void analog_getBulkCaptureState (unsigned int *id, struct Result *result, const int index)
```

Get the current bulk capture state for this analog.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

### 3.6.3 App Entity

**group AppEntity**

AppClass. Used to send a cmdAPP packet to the BrainStem network. These commands are used for either host-to-stem or stem-to-stem interactions. BrainStem modules can implement a reflex origin to complete an action when a cmdAPP packet is addressed to the module.

void **app\_execute** (unsigned int \*id, struct Result \*result, const int index, const unsigned int appParam)

Execute the app reflex on the module. Don't wait for a return value from the execute call; this call returns immediately upon execution of the module's reflex.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **appParam** – The app parameter handed to the reflex.

**Returns**

::aErrNone - success.

**Returns**

::aErrTimeout - The request timed out waiting to start execution.

**Returns**

::aErrConnection - No active link connection.

**Returns**

::aErrNotFound - the app reflex was not found or not enabled on the module.

void **app\_executeAndReturn** (unsigned int \*id, struct Result \*result, const int index, const unsigned int appParam, const unsigned int msTimeout)

Execute the app reflex on the module. Wait for a return from the reflex execution for msTimoue milliseconds. This method will block for up to msTimeout.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **returnVal** – The return value filled in from the result of executing the reflex routine.
- **msTimeout** – The amount of time to wait for the return value from the reflex routine. The default value is 1000 milliseconds if not specified.

**Returns**

::aErrNone - success.

**Returns**

::aErrTimeout - The request timed out waiting for a response.

**Returns**

::aErrConnection - No active link connection.

**Returns**

::aErrNotFound - the app reflex was not found or not enabled on the module.

### 3.6.4 Clock Entity

#### group ClockEntity

ClockClass. Provides an interface to a real-time clock entity on a BrainStem module. The clock entity may be used to get and set the real time of the system. The clock entity has a one second resolution.

---

**Note:** Clock time must be reset if power to the BrainStem module is lost.

---

void **clock\_getYear** (unsigned int \*id, struct Result \*result, const int index)

Get the four digit year value (0-4095).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **clock\_setYear** (unsigned int \*id, struct Result \*result, const int index, const int year)

Set the four digit year value (0-4095).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **year** – Set the year portion of the real-time clock value.

**Returns**

Returns common entity return values

void **clock\_getMonth** (unsigned int \*id, struct Result \*result, const int index)

Get the two digit month value (1-12).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

`void clock_setMonth (unsigned int *id, struct Result *result, const int index, const unsigned char month)`

Set the two digit month value (1-12).

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **month** – The two digit month portion of the real-time clock value.

#### Returns

Returns common entity return values

`void clock_getDay (unsigned int *id, struct Result *result, const int index)`

Get the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void clock_setDay (unsigned int *id, struct Result *result, const int index, const unsigned char day)`

Set the two digit day of month value (1-28, 29, 30 or 31 depending on the month).

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **day** – The two digit day portion of the real-time clock value.

#### Returns

Returns common entity return values

`void clock_getHour (unsigned int *id, struct Result *result, const int index)`

Get the two digit hour value (0-23).

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void clock_setHour (unsigned int *id, struct Result *result, const int index, const unsigned char hour)`

Set the two digit hour value (0-23).

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **hour** – The two digit hour portion of the real-time clock value.

**Returns**

Returns common entity return values

void **clock\_getMinute** (unsigned int \*id, struct Result \*result, const int index)

Get the two digit minute value (0-59).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **clock\_setMinute** (unsigned int \*id, struct Result \*result, const int index, const unsigned char min)

Set the two digit minute value (0-59).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **min** – The two digit minute portion of the real-time clock value.

**Returns**

Returns common entity return values

void **clock\_getSecond** (unsigned int \*id, struct Result \*result, const int index)

Get the two digit second value (0-59).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **clock\_setSecond** (unsigned int \*id, struct Result \*result, const int index, const unsigned char sec)

Set the two digit second value (0-59).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **sec** – The two digit second portion of the real-time clock value.

**Returns**

Returns common entity return values

### 3.6.5 Digital Entity

**group DigitalEntity**

DigitalClass. Interface to digital entities on BrainStem modules. Digital entities have the following 5 possibilities: Digital Input, Digital Output, RCservo Input, RCservo Output, and HighZ. Other capabilities may be available and not all pins support all configurations. Please see the product datasheet.

```
void digital_setConfiguration (unsigned int *id, struct Result *result, const int index, const unsigned char configuration)
```

Set the digital configuration to one of the available 5 states. Note: Some configurations are only supported on specific pins.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **configuration** –
  - Digital Input: digitalConfigurationInput = 0
  - Digital Output: digitalConfigurationOutput = 1
  - RCservo Input: digitalConfigurationRCServoInput = 2
  - RCservo Output: digitalConfigurationRCServoOutput = 3
  - High Z State: digitalConfigurationHiZ = 4
  - Digital Input: digitalConfigurationInputPullUp = 0
  - Digital Input: digitalConfigurationInputNoPull = 4
  - Digital Input: digitalConfigurationInputPullDown = 5

**Returns**

Returns common entity return values

**Returns**

::aErrConfiguration - Entity does not support this configuration.

```
void digital_getConfiguration (unsigned int *id, struct Result *result, const int index)
```

Get the digital configuration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

`void digital_setState (unsigned int *id, struct Result *result, const int index, const unsigned char state)`  
Set the logical state.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **state** – The state to be set. 0 is logic low, 1 is logic high.

#### Returns

Returns common entity return values

`void digital_getState (unsigned int *id, struct Result *result, const int index)`  
Get the state.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void digital_setStateAll (unsigned int *id, struct Result *result, const int index, const unsigned int state)`

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

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **state** – The state to be set for all digitals in a bit mapped representation. 0 is logic low, 1 is logic high. Where bit 0 = digital 0, bit 1 = digital 1 etc.

#### Returns

Returns common entity return values

`void digital_getStateAll (unsigned int *id, struct Result *result, const int index)`

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

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

### 3.6.6 Equalizer Entity

**group EqualizerEntity**

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

**void equalizer\_setReceiverConfig** (unsigned int \*id, struct Result \*result, const int index, const unsigned char channel, const unsigned char config)

Sets the receiver configuration for a given channel.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – The equalizer receiver channel.
- **config** – Configuration to be applied to the receiver.

#### Returns

Returns common entity return values.

**void equalizer\_getReceiverConfig** (unsigned int \*id, struct Result \*result, const int index, const unsigned char channel)

Gets the receiver configuration for a given channel.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **config** – Configuration of the receiver.

#### Returns

Returns common entity return values.

**void equalizer\_setTransmitterConfig** (unsigned int \*id, struct Result \*result, const int index, const unsigned char config)

Sets the transmitter configuration

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **config** – Configuration to be applied to the transmitter.

#### Returns

Returns common entity return values.

---

```
void equalizer_getTransmitterConfig(unsigned int *id, struct Result *result, const int index)
```

Gets the transmitter configuration

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values.

### 3.6.7 I2C Entity

#### group I2CEntity

I2CClass. Interface the I2C buses on BrainStem modules. The class provides a way to send read and write commands to I2C devices on the entities bus.

```
void i2c_read(unsigned int *id, struct Result *result, const int index, const int address, const int
               bufferLength, unsigned char *buffer)
```

Read from a device on this I2C bus.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to read.
- **length** – The length of the data to read in bytes.
- **result** – The array of bytes that will be filled with the result, upon success. This array should be larger or equivalent to aBRAINSTEM\_MAXPACKETBYTES - 5

#### Returns

Returns common entity return values

```
void i2c_write(unsigned int *id, struct Result *result, const int index, const int address, const int
                  bufferLength, unsigned char *buffer)
```

Write to a device on this I2C bus.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **address** – The I2C address (7bit <XXXX-XXX0>) of the device to write.
- **length** – The length of the data to write in bytes.
- **data** – The data to send to the device, This array should be no larger than aBRAINSTEM\_MAXPACKETBYTES - 5

### Returns

Returns common entity return values

`void i2c_setPullup (unsigned int *id, struct Result *result, const int index, const bool bEnable)`

Set bus pull-up state. This call only works with stems that have software controlled pull-ups. Check the datasheet for more information. This parameter is saved when system.save is called.

### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** -- true enables pull-ups false disables them.

### Returns

Returns common entity return values

`void i2c_setSpeed (unsigned int *id, struct Result *result, const int index, const unsigned char speed)`

Set I2C bus speed.

This call sets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz

### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **speed** -- The speed setting value.

### Returns

Returns common entity return values

`void i2c_getSpeed (unsigned int *id, struct Result *result, const int index)`

Get I2C bus speed.

This call gets the communication speed for I2C transactions through this API. Speed is an enumeration value which can take the following values. 1 - 100Khz 2 - 400Khz 3 - 1MHz

### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

### Returns

Returns common entity return values

### 3.6.8 Module Entity

#### group **ModuleEntity**

The Module Entity provides a generic interface to a BrainStem hardware module. The Module Class is the parent class for all BrainStem modules. Each module inherits from Module and implements its hardware specific features.

**void module\_createStem** (unsigned int \*id, struct Result \*result, unsigned int sn = 0)

Creates a generic BrainStem/stem object to be used in the program

#### Parameters

- **id** – This value will be assigned by the underlying library if a serial number is not provided and will be replaced by the devices serial number once a connection is made.
- **result** – object, containing NO\_ERROR or a non zero Error code.
- **sn** – Serial number of the device you want to create. Not required.

**void module\_disconnectAndDestoryStem** (unsigned int \*id, struct Result \*result)

Disconnects from device defined by the ID and will destroy any internal memory associated with the Device.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – object, containing NO\_ERROR or a non zero Error code.

**void module\_discoverAndConnect** (unsigned int \*id, struct Result \*result, int transport = (int)USB)

Finds and connects to the first device found on the given transport. If a serial number was provided when module\_createStem was called then it will only connect to that specific id.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – object, containing NO\_ERROR or a non zero Error code.
- **transport** – Defines what connection method should be searched for BrainStem devices. (i.e. USB, TCPIP, etc.)

**void module\_sDiscover** (unsigned int \*id, struct Result \*result, struct deviceListItem \*stemList, int \*stemListLength, int transport = (int)USB)

Discovers all of the BrainStem devices on a given transport. The return list is filled with device specifiers which contains information about the device.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – object, containing NO\_ERROR or a non zero Error code.
- **stemList** – List of device specifiers for each of the discovered devices
- **stemListLength** – Indicates how long the list is.
- **transport** – Defines what connection method should be searched for BrainStem devices. (i.e. USB, TCPIP, etc.)

`void module_disconnect (unsigned int *id, struct Result *result)`

Disconnects device, but does not destroy underlying object. i.e. “module\_reconnect” could be called without calling “module\_createStem” again.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – object, containing NO\_ERROR or a non zero Error code.

`void module_reconnect (unsigned int *id, struct Result *result)`

Reestablishes a connection with a preexisting stem object. The original stem would of been created with “module\_createStem”.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – object, containing NO\_ERROR or a non zero Error code.

`void module_connectThroughLinkModule (unsigned int *id, unsigned int *id_linkStem, struct Result *result)`

Establishes connection through another stems link/connection (i.e. transport: USB, TCPIP). Refer to BrainStem Networking at [www.acroname.com/support](http://www.acroname.com/support)

#### Parameters

- `id` – The id assigned by the create stem vi.
- `id_linkStem` – The link stem’s id assigned by the create stem vi. (The stem providing the connection.)
- `result` – object, containing NO\_ERROR or a non zero Error code.

`void module_setModuleAddress (unsigned int *id, struct Result *result, int address)`

Changes the module address of the stem object that was created via “module\_createStem”. Refer to BrainStem Networking at [www.acroname.com/support](http://www.acroname.com/support)

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – object, containing NO\_ERROR or a non zero Error code.
- `address` – New address to be set.

`void module_getModuleAddress (unsigned int *id, struct Result *result)`

Retrieves the module address of the stem object that was created via “module\_createStem”. Refer to BrainStem Networking at [www.acroname.com/support](http://www.acroname.com/support)

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – object, containing NO\_ERROR and the module/stems current module address or a non zero Error code.

`void module_isConnected (unsigned int *id, struct Result *result)`

Returns the current state of the module/stem’s connection. Refer to BrainStem Networking at [www.acroname.com/support](http://www.acroname.com/support)

#### Parameters

- `id` – The id assigned by the create stem vi.

- **result** - object, containing NO\_ERROR and the status of the connection or a non zero Error code. (0 = disconnected; 1 = connected.)

**void module\_setNetworkingMode (unsigned int \*id, struct Result \*result, int mode)**

Changes the networking mode of the stem object. Auto mode is enabled by default which allows automatic adjustment of the module/stems networking configuration. Refer to BrainStem Networking at [www.acroname.com/support](http://www.acroname.com/support)

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - object, containing NO\_ERROR or a non zero Error code.
- **mode** - New mode to be set.

**void module\_clearAllStems ()**

Disconnects and destroys all modules/stems. During development the dll doesn't always "detach" between runs. This can create problematic scenarios if there are not subsequent disconnect and destroy calls for every create call made. This function can be a helpful post-execution/tear-down process when bringing up new BrainStem Networks. However, not required if connections are handles correctly. Refer to BrainStem Networking at [www.acroname.com/support](http://www.acroname.com/support)

### 3.6.9 Mux Entity

**group MuxEntity**

MuxClass. A MUX is a multiplexer that takes one or more similar inputs (bus, connection, or signal) and allows switching to one or more outputs. An analogy would be the switchboard of a telephone operator. Calls (inputs) come in and by re-connecting the input to an output, the operator (multiplexer) can direct that input to one or more outputs.

One possible output is to not connect the input to anything which essentially disables that input's connection to anything.

Not every MUX has multiple inputs. Some may simply be a single input that can be enabled (connected to a single output) or disabled (not connected to anything).

**void mux\_getEnable (unsigned int \*id, struct Result \*result, const int index)**

Get the mux enable/disable status

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** - The index of the entity in question.

#### Returns

Returns common entity return values

**void mux\_setEnable (unsigned int \*id, struct Result \*result, const int index, const unsigned char bEnable)**

Enable the mux.

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone on success. Non-zero error code on failure.

- **index** – The index of the entity in question.
- **bEnable** – true: enables the mux for the selected channel.

**Returns**

Returns common entity return values

void **mux\_getChannel** (unsigned int \*id, struct Result \*result, const int index)

Get the current selected mux channel.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **mux\_setChannel** (unsigned int \*id, struct Result \*result, const int index, const unsigned char channel)

Set the current mux channel.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **channel** – mux channel to select.

**Returns**

Returns common entity return values

void **mux\_getChannelVoltage** (unsigned int \*id, struct Result \*result, const int index, const unsigned char channel)

Get the voltage of the indicated mux channel.

---

**Note:** Not all modules provide 32 bits of accuracy; Refer to the module's datasheet to determine the analog bit depth and reference voltage.

---

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – 32 bit signed integer (in microvolts) based on the board's ground and reference voltages.

**Returns**

Returns common entity return values

```
void mux_getConfiguration (unsigned int *id, struct Result *result, const int index)
```

Get the configuration of the mux.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void mux_setConfiguration (unsigned int *id, struct Result *result, const int index, const int config)
```

Set the configuration of the mux.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **config** – integer representing the mux configuration either muxConfig\_default, or muxConfig\_splitMode.

#### Returns

Returns common entity return values

```
void mux_getSplitMode (unsigned int *id, struct Result *result, const int index)
```

Get the current split mode mux configuration.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void mux_setSplitMode (unsigned int *id, struct Result *result, const int index, const int splitMode)
```

Sets the mux's split mode configuration.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **splitMode** – integer representing the channel selection for each sub-channel within the mux. See the data-sheet for the device for specific information.

#### Returns

Returns common entity return values

### 3.6.10 Pointer Entity

#### group PointerEntity

PointerClass. Access the reflex scratchpad from a host computer.

The Pointers access the pad which is a shared memory area on a BrainStem module. The interface allows the use of the brainstem scratchpad from the host, and provides a mechanism for allowing the host application and brainstem reflexes to communicate.

The Pointer allows access to the pad in a similar manner as a file pointer accesses the underlying file. The cursor position can be set via setOffset. A read of a character short or int can be made from that cursor position. In addition the mode of the pointer can be set so that the cursor position automatically increments or set so that it does not this allows for multiple reads of the same pad value, or reads of multi-record values, via and incrementing pointer.

`void pointer_getOffset (unsigned int *id, struct Result *result, const int index)`

Get the offset of the pointer

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

All possible standard UEI return values.

`void pointer_setOffset (unsigned int *id, struct Result *result, const int index, int offset)`

Set the offset of the pointer

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **offset** – The value of the offset.

#### Returns

All possible standard UEI return values.

`void pointer_getMode (unsigned int *id, struct Result *result, const int index)`

Get the mode of the pointer

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

All possible standard UEI return values.

`void pointer_setMode (unsigned int *id, struct Result *result, const int index, unsigned char mode)`

Set the mode of the pointer

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – The mode: aPOINTER\_MODE\_STATIC or aPOINTER\_MODE\_AUTO\_INCREMENT.

#### Returns

All possible standard UEI return values.

`void pointer_getTransferStore (unsigned int *id, struct Result *result, const int index)`

Get the handle to the store.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

All possible standard UEI return handles.

`void pointer_setTransferStore (unsigned int *id, struct Result *result, const int index, unsigned char handle)`

Set the handle to the store.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **handle** – The handle of the store.

#### Returns

All possible standard UEI return handles.

`void pointer_initiateTransferToStore (unsigned int *id, struct Result *result, const int index, unsigned char length)`

Transfer data to the store.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **length** – The length of the data transfer.

#### Returns

All possible standard UEI return values.

```
void pointer_initiateTransferFromStore (unsigned int *id, struct Result *result, const int index,  
                                     unsigned char length)
```

Transfer data from the store.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **length** – The length of the data transfer.

#### Returns

All possible standard UEI return values.

```
void pointer_getChar (unsigned int *id, struct Result *result, const int index)
```

Get a char (1 byte) value from the pointer at this object's index, where elements are 1 byte long.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

All possible standard UEI return values.

```
void pointer_setChar (unsigned int *id, struct Result *result, const int index, const unsigned char value)
```

Set a char (1 byte) value to the pointer at this object's element index, where elements are 1 byte long.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – The single char (1 byte) value to be stored in the pointer.

#### Returns

All possible standard UEI return values.

```
void pointer_getShort (unsigned int *id, struct Result *result, const int index)
```

Get a short (2 byte) value from the pointer at this objects index, where elements are 2 bytes long

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

All possible standard UEI return values.

`void pointer_setShort` (unsigned int \*id, struct Result \*result, const int index, const int value)  
Set a short (2 bytes) value to the pointer at this object's element index, where elements are 2 bytes long.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – The single short (2 byte) value to be set in the pointer.

#### Returns

All possible standard UEI return values.

`void pointer_getInt` (unsigned int \*id, struct Result \*result, const int index)  
Get an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

All possible standard UEI return values.

`void pointer_setInt` (unsigned int \*id, struct Result \*result, const int index, const unsigned int value)  
Set an int (4 bytes) value from the pointer at this objects index, where elements are 4 bytes long

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – The single int (4 byte) value to be stored in the pointer.

#### Returns

All possible standard UEI return values.

### 3.6.11 Port Entity

#### group PortEntity

Port Class The Port Entity provides software control over the most basic items related to a USB Port. This includes everything from the complete enable and disable of the entire port to the individual control of specific pins. Voltage and Current measurements are also included for devices which support the Port Entity.

`void port_getVbusVoltage` (unsigned int \*id, struct Result \*result, const int index)  
Gets the Vbus Voltage

#### Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **port\_getVbusCurrent** (unsigned int \*id, struct Result \*result, const int index)

Gets the Vbus Current

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **port\_getVconnVoltage** (unsigned int \*id, struct Result \*result, const int index)

Gets the Vconn Voltage

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **port\_getVconnCurrent** (unsigned int \*id, struct Result \*result, const int index)

Gets the Vconn Current

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **port\_getPowerMode** (unsigned int \*id, struct Result \*result, const int index)

Gets the Port Power Mode: Convenience Function of get/setPortMode

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void port_setPowerMode (unsigned int *id, struct Result *result, const int index, const unsigned char powerMode)
```

Sets the Port Power Mode: Convenience Function of getPortMode

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerMode** – The power mode to be set.

**Returns**

Returns common entity return values

```
void port_getEnabled (unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void port_setEnabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the entire port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Fully enable port; 0 = Fully disable port.

**Returns**

Returns common entity return values

```
void port_getDataEnabled (unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the data lines.: Sub-component (Data) of getEnabled.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void port_setDataEnabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the data lines. Sub-component (Data) of setEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

#### Returns

Returns common entity return values

```
void port_getDataHSEnabled (unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the High Speed (HS) data lines. Sub-component of getDataEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void port_setDataHSEnabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the High Speed (HS) data lines. Sub-component of setDataEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

#### Returns

Returns common entity return values

```
void port_getDataHS1Enabled (unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the High Speed A side (HSA) data lines.: Sub-component of getDataHSEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

---

```
void port_setDataHS1Enabled(unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the High Speed A side (HSA) data lines. Sub-component of setDataHSEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

#### Returns

Returns common entity return values

```
void port_getDataHS2Enabled(unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the High Speed B side (HSB) data lines.: Sub-component of getDataHSEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void port_setDataHS2Enabled(unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the High Speed B side (HSB) data lines. Sub-component of setDataHSEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

#### Returns

Returns common entity return values

```
void port_getDataSSEnabled(unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the Super Speed (SS) data lines. Sub-component of getDataEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void port_setDataSSEnabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

#### Returns

Returns common entity return values

```
void port_getDataSS1Enabled (unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the Super Speed A side (SSA) data lines.: Sub-component of getDataSSEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void port_setDataSS1Enabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the Super Speed (SS) data lines. Sub-component of setDataEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

#### Returns

Returns common entity return values

```
void port_getDataSS2Enabled (unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the Super Speed B side (SSB) data lines.: Sub-component of getDataSSEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

---

```
void port_setDataSS2Enabled(unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or disables the Super Speed B side (SSB) data lines. Sub-component of setDataSSEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable data; 0 = Disable data.

#### Returns

Returns common entity return values

```
void port_getPowerEnabled(unsigned int *id, struct Result *result, const int index)
```

Gets the current enable value of the power lines.: Sub-component (Power) of getEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void port_setPowerEnabled(unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enables or Disables the power lines. Sub-component (Power) of setEnable.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable power; 0 = Disable disable.

#### Returns

Returns common entity return values

```
void port_getDataRole(unsigned int *id, struct Result *result, const int index)
```

Gets the Port Data Role.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_getVconnEnabled (unsigned int *id, struct Result *result, const int index)`

Gets the current enable value of the Vconn lines.: Sub-component (Vconn) of getEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_setVconnEnabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)`

Enables or disables the Vconn lines. Sub-component (Vconn) of setEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn lines; 0 = Disable Vconn lines.

#### Returns

Returns common entity return values

`void port_getVconn1Enabled (unsigned int *id, struct Result *result, const int index)`

Gets the current enable value of the Vconn1 lines. Sub-component of getVconnEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_setVconn1Enabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)`

Enables or disables the Vconn1 lines. Sub-component of setVconnEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn1 lines; 0 = Disable Vconn1 lines.

#### Returns

Returns common entity return values

`void port_getVconn2Enabled (unsigned int *id, struct Result *result, const int index)`

Gets the current enable value of the Vconn2 lines. Sub-component of getVconnEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_setVconn2Enabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)`

Enables or disables the Vconn2 lines. Sub-component of setVconnEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable Vconn2 lines; 0 = Disable Vconn2 lines.

#### Returns

Returns common entity return values

`void port_getCCEnabled (unsigned int *id, struct Result *result, const int index)`

Gets the current enable value of the CC lines.: Sub-component (CC) of getEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_setCCEnabled (unsigned int *id, struct Result *result, const int index, const unsigned char enable)`

Enables or disables the CC lines. Sub-component (CC) of setEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC lines; 0 = Disable CC lines.

#### Returns

Returns common entity return values

`void port_getCC1Enabled(unsigned int *id, struct Result *result, const int index)`

Gets the current enable value of the CC1 lines. Sub-component of getCCEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_setCC1Enabled(unsigned int *id, struct Result *result, const int index, const unsigned char enable)`

Enables or disables the CC1 lines. Sub-component of setCCEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC1 lines; 0 = Disable CC1 lines.

#### Returns

Returns common entity return values

`void port_getCC2Enabled(unsigned int *id, struct Result *result, const int index)`

Gets the current enable value of the CC2 lines. Sub-component of getCCEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_setCC2Enabled(unsigned int *id, struct Result *result, const int index, const unsigned char enable)`

Enables or disables the CC2 lines. Sub-component of setCCEnabled.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – 1 = Enable CC2 lines; 0 = Disable CC2 lines.

#### Returns

Returns common entity return values

`void port_getVoltageSetpoint (unsigned int *id, struct Result *result, const int index)`

Gets the current voltage setpoint value for the port.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_setVoltageSetpoint (unsigned int *id, struct Result *result, const int index, const unsigned int value)`

Sets the current voltage setpoint value for the port.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **value** – the voltage setpoint of the port in uV.

#### Returns

Returns common entity return values

`void port_getState (unsigned int *id, struct Result *result, const int index)`

A bit mapped representation of the current state of the port. Reflects what he port IS which may differ from what was requested.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

`void port_getDataSpeed (unsigned int *id, struct Result *result, const int index)`

Gets the speed of the enumerated device.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void port_getMode (unsigned int *id, struct Result *result, const int index)`

Gets current mode of the port

#### Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_setMode (unsigned int \*id, struct Result \*result, const int index, const unsigned int mode)**

Sets the mode of the port

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – Port mode to be set. See product datasheet for details.

**Returns**

Returns common entity return values

**void port\_getErrors (unsigned int \*id, struct Result \*result, const int index)**

Returns any errors that are present on the port. Calling this function will clear the current errors. If the error persists it will be set again.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void portGetCurrentLimit (unsigned int \*id, struct Result \*result, const int index)**

Gets the current limit of the port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_setCurrentLimit (unsigned int \*id, struct Result \*result, const int index, const unsigned int limit)**

Sets the current limit of the port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **limit** – Current limit to be applied in microAmps (uA).

#### Returns

Returns common entity return values

**void port\_getCurrentLimitMode** (unsigned int \*id, struct Result \*result, const int index)

Gets the current limit mode. The mode determines how the port will react to an over current condition.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

**void port\_setCurrentLimitMode** (unsigned int \*id, struct Result \*result, const int index, const unsigned char mode)

Sets the current limit mode. The mode determines how the port will react to an over current condition.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the current limit mode. Available modes are product specific. See the reference documentation.

#### Returns

Returns common entity return values

**void port\_getAvailablePower** (unsigned int \*id, struct Result \*result, const int index)

Gets the current available power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

**void port\_getAllocatedPower** (unsigned int \*id, struct Result \*result, const int index)

Gets the currently allocated power. This value is determined by the power manager which is responsible for budgeting the systems available power envelope.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

`void port_getPowerLimit (unsigned int *id, struct Result *result, const int index)`

Gets the user defined power limit for the port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

`void port_setPowerLimit (unsigned int *id, struct Result *result, const int index, const unsigned int limit)`

Sets a user defined power limit for the port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **limit** – Power limit to be applied in milli-watts (mW).

**Returns**

Returns common entity return values

`void port_getPowerLimitMode (unsigned int *id, struct Result *result, const int index)`

Gets the power limit mode. The mode determines how the port will react to an over power condition.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

`void port_setPowerLimitMode (unsigned int *id, struct Result *result, const int index, const unsigned char mode)`

Sets the power limit mode. The mode determines how the port will react to an over power condition.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the power limit mode to be applied Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

```
void port_getName (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const int bufferLength)
```

Gets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

```
void port_setName (unsigned int *id, struct Result *result, const int index, unsigned char *buffer, const int bufferLength)
```

Sets a user defined name of the port. Helpful for identifying ports/devices in a static environment.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

```
void port_getDataHSRoutingBehavior (unsigned int *id, struct Result *result, const int index)
```

Gets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void port_setDataHSRoutingBehavior (unsigned int *id, struct Result *result, const int index, const unsigned char mode)
```

Sets the HighSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

**void port\_getDataSSRoutingBehavior** (unsigned int \*id, struct Result \*result, const int index)

Gets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_setDataSSRoutingBehavior** (unsigned int \*id, struct Result \*result, const int index, const unsigned char mode)

Sets the SuperSpeed Data Routing Behavior. The mode determines how the port will route the data lines.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – An enumerated representation of the routing behavior. Available modes are product specific. See the reference documentation.

**Returns**

Returns common entity return values

**void port\_getVbusAccumulatedPower** (unsigned int \*id, struct Result \*result, const int index)

Gets the Vbus Accumulated Power

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_resetVbusAccumulatedPower** (unsigned int \*id, struct Result \*result, const int index)

Resets the Vbus Accumulated Power to zero.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_getVconnAccumulatedPower** (unsigned int \*id, struct Result \*result, const int index)

Gets the Vconn Accumulated Power

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_resetVconnAccumulatedPower** (unsigned int \*id, struct Result \*result, const int index)

Resets the Vconn Accumulated Power to zero.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_setHSBoost** (unsigned int \*id, struct Result \*result, const int index, const unsigned char boost)

Sets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **boost** – An enumerated representation of the boost range. Available value are product specific. See the reference documentation.

**Returns**

Returns common entity return values

**void port\_getHSBoost** (unsigned int \*id, struct Result \*result, const int index)

Gets the ports USB 2.0 High Speed Boost Settings The setting determines how much additional drive the USB 2.0 signal will have in High Speed mode.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void port\_resetEntityToFactoryDefaults (unsigned int \*id, struct Result \*result, const int index)**

Resets the PortClass Entity to it factory default configuration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

### 3.6.12 PowerDelivery Entity

**group PowerDeliveryEntity**

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

**void powerdelivery\_getConnectionState (unsigned int \*id, struct Result \*result, const int index)**

Gets the current state of the connection in the form of an enumeration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void powerdelivery\_getNumberOfPowerDataObjects (unsigned int \*id, struct Result \*result, const int index, const unsigned char partner, const unsigned char powerRole)**

Gets the number of Power Data Objects (PDOs) for a given partner and power role.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **numRules** – Variable to be filled with the number of PDOs.

**Returns**

Returns common entity return values

```
void powerdelivery_getPowerDataObject (unsigned int *id, struct Result *result, const int index,
                                      const unsigned char partner, const unsigned char
                                      powerRole, const unsigned char ruleIndex)
```

Gets the Power Data Object (PDO) for the requested partner, powerRole and index.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Variable to be filled with the requested power rule.

**Returns**

Returns common entity return values

```
void powerdelivery_setPowerDataObject (unsigned int *id, struct Result *result, const int index,
                                       const unsigned char powerRole, const unsigned char
                                       ruleIndex, const unsigned int pdo)
```

Sets the Power Data Object (PDO) of the local partner for a given power role and index.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **pdo** – Power Data Object to be set.

**Returns**

Returns common entity return values

```
void powerdelivery_resetPowerDataObjectToDefault (unsigned int *id, struct Result *result, const
                                                 int index, const unsigned char powerRole,
                                                 const unsigned char ruleIndex)
```

Resets the Power Data Object (PDO) of the Local partner for a given power role and index.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.

#### Returns

Returns common entity return values

```
void powerdelivery_getPowerDataObjectList (unsigned int *id, struct Result *result, const int index,  
                                         unsigned int *buffer, const int bufferLength)
```

Gets all Power Data Objects (PDOs). Equivalent to calling PowerDeliveryClass::getPowerDataObject() on all partners, power roles, and index's.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled The order of which is:
  - Rules 1-7 Local Source
  - Rules 1-7 Local Sink
  - Rules 1-7 Partner Source
  - Rules 1-7 Partner Sink.
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled. On success this value should be 28 (7 rules \* 2 partners \* 2 power roles)

#### Returns

Returns common entity return values

```
void powerdelivery_getPowerDataObjectEnabled (unsigned int *id, struct Result *result, const int  
                                              index, const unsigned char powerRole, const  
                                              unsigned char ruleIndex)
```

Gets the enabled state of the Local Power Data Object (PDO) for a given power role and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – Variable to be filled with enabled state.

**Returns**

Returns common entity return values

```
void powerdelivery_setPowerDataObjectEnabled(unsigned int *id, struct Result *result, const int index, const unsigned char powerRole, const unsigned char ruleIndex, const unsigned char enabled)
```

Sets the enabled state of the Local Power Data Object (PDO) for a given powerRole and index. Enabled refers to whether the PDO will be advertised when a PD connection is made. This does not indicate the currently active rule index. This information can be found in Request Data Object (RDO).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Indicates which power role of PD connection is in question.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – The index of the PDO in question. Valid index are 1-7.
- **enabled** – The state to be set.

**Returns**

Returns common entity return values

```
void powerdelivery_getPowerDataObjectEnabledList(unsigned int *id, struct Result *result, const int index, const unsigned char powerRole)
```

Gets all Power Data Object enables for a given power role. Equivalent of calling PowerDeliveryClass::getPowerDataObjectEnabled() for all indexes.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enabledList** – Variable to be filled with a mapped representation of the enabled PDOs for a given power role. Values align with a given rule index (bits 1-7, bit 0 is invalid)

**Returns**

Returns common entity return values

```
void powerdelivery_getRequestDataObject(unsigned int *id, struct Result *result, const int index, const unsigned char partner)
```

Gets the current Request Data Object (RDO) for a given partner. RDOs: Are provided by the sinking device. Exist only after a successful PD negotiation (Otherwise zero). Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **rdo** – Variable to be filled with the current RDO. Zero indicates the RDO is not active.

**Returns**

Returns common entity return values

```
void powerdelivery_setRequestDataObject (unsigned int *id, struct Result *result, const int index,  
const unsigned char partner, const unsigned int rdo)
```

Sets the current Request Data Object (RDO) for a given partner. (Only the local partner can be changed.)  
RDOs: Are provided by the sinking device. Exist only after a successful PD negotiation (Otherwise zero).  
Only one RDO can exist at a time. i.e. Either the Local or Remote partner RDO

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **partner** – Indicates which side of the PD connection is in question.
  - Local = 0 = powerdeliveryPartnerLocal
- **rdo** – Request Data Object to be set.

**Returns**

Returns common entity return values

```
void powerdelivery_getPowerRole (unsigned int *id, struct Result *result, const int index)
```

Gets the power role that is currently being advertised by the local partner. (CC Strapping).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void powerdelivery_setPowerRole (unsigned int *id, struct Result *result, const int index, const  
unsigned char powerRole)
```

Set the current power role to be advertised by the Local partner. (CC Strapping).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Value to be applied.
  - Disabled = 0 = powerdeliveryPowerRoleDisabled
  - Source = 1= powerdeliveryPowerRoleSource

- Sink = 2 = powerdeliveryPowerRoleSink
- Source/Sink = 3 = powerdeliveryPowerRoleSourceSink (Dual Role Port)

**Returns**

Returns common entity return values

**void powerdelivery\_getPowerRolePreferred** (unsigned int \*id, struct Result \*result, const int index)

Gets the preferred power role currently being advertised by the Local partner. (CC Strapping).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void powerdelivery\_setPowerRolePreferred** (unsigned int \*id, struct Result \*result, const int index, const unsigned char powerRole)

Set the preferred power role to be advertised by the Local partner (CC Strapping).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **powerRole** – Value to be applied.
  - Disabled = 0 = powerdeliveryPowerRoleDisabled
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink

**Returns**

Returns common entity return values

**void powerdelivery\_getCableVoltageMax** (unsigned int \*id, struct Result \*result, const int index)

Gets the maximum voltage capability reported by the e-mark of the attached cable.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void powerdelivery\_getCableCurrentMax** (unsigned int \*id, struct Result \*result, const int index)

Gets the maximum current capability report by the e-mark of the attached cable.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **powerdelivery\_getCableSpeedMax** (unsigned int \*id, struct Result \*result, const int index)

Gets the maximum data rate capability reported by the e-mark of the attached cable.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **powerdelivery\_getCableType** (unsigned int \*id, struct Result \*result, const int index)

Gets the cable type reported by the e-mark of the attached cable.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **powerdelivery\_getCableOrientation** (unsigned int \*id, struct Result \*result, const int index)

Gets the current orientation being used for PD communication

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

void **powerdelivery\_request** (unsigned int \*id, struct Result \*result, const int index, const unsigned char request)

Requests an action of the Remote partner. Actions are not guaranteed to occur.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **request** – Request to be issued to the remote partner

- pdRequestHardReset (0)
- pdRequestSoftReset (1)
- pdRequestDataReset (2)
- pdRequestPowerRoleSwap (3)
- pdRequestPowerFastRoleSwap (4)
- pdRequestDataRoleSwap (5)
- pdRequestVconnSwap (6)
- pdRequestSinkGoToMinimum (7)
- pdRequestRemoteSourcePowerDataObjects (8)
- pdRequestRemoteSinkPowerDataObjects (9)

**Returns**

The returned error represents the success of the request being sent to the partner only. The success of the request being serviced by the remote partner can be obtained through PowerDeliveryClass::requestStatus() Returns common entity return values

**void powerdelivery\_requestStatus (unsigned int \*id, struct Result \*result, const int index)**

Gets the status of the last request command sent.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void powerdelivery\_getOverride (unsigned int \*id, struct Result \*result, const int index)**

Gets the current enabled overrides

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void powerdelivery\_setOverride (unsigned int \*id, struct Result \*result, const int index, const unsigned int overrides)**

Sets the current enabled overrides

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

- **overrides** – Overrides to be set in a bit mapped representation.

**Returns**

Returns common entity return values

```
void powerdelivery_resetEntityToFactoryDefaults (unsigned int *id, struct Result *result, const int index)
```

```
void powerdelivery_getFlagMode (unsigned int *id, struct Result *result, const int index, const unsigned char flag)
```

Gets the current mode of the local partner flag/advertisement. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – Variable to be filled with the current mode.
  - Disabled (0)
  - Enabled (1)
  - Auto (2) default

**Returns**

Returns common entity return values

```
void powerdelivery_setFlagMode (unsigned int *id, struct Result *result, const int index, const unsigned char flag, const unsigned char mode)
```

Sets how the local partner flag/advertisement is managed. These flags are apart of the first Local Power Data Object and must be managed in order to accurately represent the system to other PD devices. This API allows overriding of that feature. Overriding may lead to unexpected behaviors.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **flag** – Flag/Advertisement to be modified
- **mode** – Value to be applied.
  - Disabled (0)
  - Enabled (1)
  - Auto (2) default

**Returns**

Returns common entity return values

---

```
void powerdelivery_getPeakCurrentConfiguration(unsigned int *id, struct Result *result, const
                                              int index)
```

Gets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void powerdelivery_setPeakCurrentConfiguration(unsigned int *id, struct Result *result, const
                                              int index, const unsigned char configuration)
```

Sets the Peak Current Configuration for the Local Source. The peak current configuration refers to the allowable tolerance/overload capabilities in regards to the devices max current. This tolerance includes a maximum value and a time unit.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **configuration** – An enumerated value referring to the configuration to be set
  - Allowable values are 0 - 4

#### Returns

Returns common entity return values

```
void powerdelivery_getFastRoleSwapCurrent(unsigned int *id, struct Result *result, const int
                                         index)
```

Gets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void powerdelivery_setFastRoleSwapCurrent(unsigned int *id, struct Result *result, const int index,
                                         const unsigned char swapCurrent)
```

Sets the Fast Role Swap Current The fast role swap current refers to the amount of current required by the Local Sink in order to successfully preform the swap.

#### Parameters

- **id** – The id assigned by the create stem vi.

- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **index** - The index of the entity in question.
- **swapCurrent** - An enumerated value referring to value to be set.
  - 0A (0)
  - 900mA (1)
  - 1.5A (2)
  - 3A (3)

#### Returns

Returns common entity return values

```
void powerdelivery_packDataObjectAttributes (unsigned int *id, struct Result *result, const int
                                             index, const unsigned char partner, const
                                             unsigned char powerRole, const unsigned char
                                             ruleIndex)
```

Helper function for packing Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** - The index of the entity in question.
- **partner** - Indicates which side of the PD connection.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** - Indicates which power role of PD connection.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** - Data object index.

#### Returns

aErrNone on success; aErrParam with bad input.

```
void powerdelivery_unpackDataObjectAttributes (unsigned int *id, struct Result *result, const int
                                              index, const unsigned char attributes,
                                              unsigned char *partner, unsigned char
                                              *powerRole, unsigned char *ruleIndex)
```

Helper function for unpacking Data Object attributes. This value is used as a subindex for all Data Object calls with the BrainStem Protocol.

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **index** - The index of the entity in question.
- **attributes** - variable to be filled with packed values.

- **partner** – Indicates which side of the PD connection.
  - Local = 0 = powerdeliveryPartnerLocal
  - Remote = 1 = powerdeliveryPartnerRemote
- **powerRole** – Indicates which power role of PD connection.
  - Source = 1 = powerdeliveryPowerRoleSource
  - Sink = 2 = powerdeliveryPowerRoleSink
- **ruleIndex** – Data object index.

**Returns**

aErrNone on success; aErrParam with bad input.

### 3.6.13 Rail Entity

#### *group RailEntity*

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

**void rail\_getCurrent (unsigned int \*id, struct Result \*result, const int index)**

Get the rail current.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_setCurrentSetpoint (unsigned int \*id, struct Result \*result, const int index, const int microamps)**

Set the rail supply current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microamps** – The current in micro-amps (1 == 1e-6A) to be supply by the rail.

**Returns**

Returns common entity return values

**void rail\_getCurrentSetpoint (unsigned int \*id, struct Result \*result, const int index)**

Get the rail setpoint current. Rail current control capabilities vary between modules. Refer to the module datasheet for definition of the rail current capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_setCurrentLimit** (unsigned int \*id, struct Result \*result, const int index, const int microamps)

Set the rail current limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microamps** – The current in micro-amps (1 == 1e-6A).

**Returns**

Returns common entity return values

**void rail\_getCurrentLimit** (unsigned int \*id, struct Result \*result, const int index)

Get the rail current limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_getTemperature** (unsigned int \*id, struct Result \*result, const int index)

Get the rail temperature.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_getEnable** (unsigned int \*id, struct Result \*result, const int index)

Get the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_setEnable (unsigned int \*id, struct Result \*result, const int index, const unsigned char bEnable)**

Set the state of the external rail switch. Not all rails can be switched on and off. Refer to the module datasheet for capability specification of the rails.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – true: enable and connect to the supply rail voltage; false: disable and disconnect from the supply rail voltage

**Returns**

Returns common entity return values

**void rail\_getVoltage (unsigned int \*id, struct Result \*result, const int index)**

Get the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_setVoltageSetpoint (unsigned int \*id, struct Result \*result, const int index, const int microvolts)**

Set the rail supply voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V) to be supplied by the rail.

**Returns**

Returns common entity return values

**void rail\_getVoltageSetpoint (unsigned int \*id, struct Result \*result, const int index)**

Get the rail setpoint voltage. Rail voltage control capabilities vary between modules. Refer to the module datasheet for definition of the rail voltage capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void rail_setVoltageMinLimit (unsigned int *id, struct Result *result, const int index, const int  
microvolts)
```

Set the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

```
void rail_getVoltageMinLimit (unsigned int *id, struct Result *result, const int index)
```

Get the rail voltage minimum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void rail_setVoltageMaxLimit (unsigned int *id, struct Result *result, const int index, const int  
microvolts)
```

Set the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **microvolts** – The voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

```
void rail_getVoltageMaxLimit (unsigned int *id, struct Result *result, const int index)
```

Get the rail voltage maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_getPower** (unsigned int \*id, struct Result \*result, const int index)

Get the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_setPowerSetpoint** (unsigned int \*id, struct Result \*result, const int index, const int milliwatts)

Set the rail supply power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **milliwatts** – The power in milli-watts ( $1 == 1e-3W$ ) to be supplied by the rail.

**Returns**

Returns common entity return values

**void rail\_getPowerSetpoint** (unsigned int \*id, struct Result \*result, const int index)

Get the rail setpoint power. Rail power control capabilities vary between modules. Refer to the module datasheet for definition of the rail power capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_setPowerLimit** (unsigned int \*id, struct Result \*result, const int index, const int milliwatts)

Set the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

- **index** – The index of the entity in question.
- **milliwatts** – The power in milli-watts (mW).

**Returns**

Returns common entity return values

**void rail\_getPowerLimit** (unsigned int \*id, struct Result \*result, const int index)

Get the rail power maximum limit setting. (Check product datasheet to see if this feature is available)

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_getResistance** (unsigned int \*id, struct Result \*result, const int index)

Get the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rail\_setResistanceSetpoint** (unsigned int \*id, struct Result \*result, const int index, const int milliohms)

Set the rail load resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **milliohms** – The resistance in milli-ohms ( $1 == 1e-3\text{Ohms}$ ) to be drawn by the rail.

**Returns**

Returns common entity return values

**void rail\_getResistanceSetpoint** (unsigned int \*id, struct Result \*result, const int index)

Get the rail setpoint resistance. Rail resistance control capabilities vary between modules. Refer to the module datasheet for definition of the rail resistance capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void rail_setKelvinSensingEnable (unsigned int *id, struct Result *result, const int index, const
                                  unsigned char bEnable)
```

Enable or Disable kelvin sensing on the module. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – enable or disable kelvin sensing.

**Returns**

Returns common entity return values

```
void rail_getKelvinSensingEnable (unsigned int *id, struct Result *result, const int index)
```

Determine whether kelvin sensing is enabled or disabled. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void rail_getKelvinSensingState (unsigned int *id, struct Result *result, const int index)
```

Determine whether kelvin sensing has been disabled by the system. Refer to the module datasheet for definition of the rail kelvin sensing capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void rail_setOperationalMode (unsigned int *id, struct Result *result, const int index, const unsigned
                             char mode)
```

Set the operational mode of the rail. Refer to the module datasheet for definition of the rail operational capabilities.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **mode** – The operational mode to employ.

#### Returns

Returns common entity return values

`void rail_getOperationalMode (unsigned int *id, struct Result *result, const int index)`

Determine the current operational mode of the system. Refer to the module datasheet for definition of the rail operational mode capabilities.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void rail_getOperationalState (unsigned int *id, struct Result *result, const int index)`

Determine the current operational state of the system. Refer to the module datasheet for definition of the rail operational states.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

`void rail_clearFaults (unsigned int *id, struct Result *result, const int index)`

Clears the current fault state of the rail. Refer to the module datasheet for definition of the rail faults.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

### 3.6.14 RCServo Entity

#### group RCServoEntity

RCServoClass. Interface to servo entities on BrainStem modules. Servo entities are built upon the digital input/output pins and therefore can also be inputs or outputs. Please see the product datasheet on the configuration limitations.

```
void rcservo_setEnable(unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enable the servo channel

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – The state to be set. 0 is disabled, 1 is enabled.

#### Returns

Returns common entity return values

```
void rcservo_getEnable(unsigned int *id, struct Result *result, const int index)
```

Get the enable status of the servo channel.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void rcservo_setPosition(unsigned int *id, struct Result *result, const int index, const unsigned char position)
```

Set the position of the servo channel

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **position** – The position to be set. Default 64 = a 1ms pulse and 192 = a 2ms pulse.

#### Returns

Returns common entity return values

```
void rcservo_getPosition(unsigned int *id, struct Result *result, const int index)
```

Get the position of the servo channel

#### Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void rcservo\_setReverse** (unsigned int \*id, struct Result \*result, const int index, const unsigned char reverse)

Set the output to be reversed on the servo channel

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **reverse** – Reverses the value set by “setPosition”. ie. if the position is set to 64 (1ms pulse) the output will now be 192 (2ms pulse); however, “getPostion” will return the set value of 64. 0 = not reversed, 1 = reversed.

**Returns**

Returns common entity return values

**void rcservo\_getReverse** (unsigned int \*id, struct Result \*result, const int index)

Get the reverse status of the servo channel

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

### 3.6.15 Relay Entity

**group RelayEntity**

RelayClass. Interface to relay entities on BrainStem modules. Relay entities can be set, and the voltage read. Other capabilities may be available, please see the product datasheet.

**void relay\_setEnable** (unsigned int \*id, struct Result \*result, const int index, const unsigned char bEnable)

Set the enable/disable state.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bEnable** – False or 0 = Disabled, True or 1 = Enabled

**Returns**

Returns common entity return values

```
void relay_getEnable (unsigned int *id, struct Result *result, const int index)
```

Get the state.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void relay_getVoltage (unsigned int *id, struct Result *result, const int index)
```

Get the scaled micro volt value with reference to ground.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

### 3.6.16 Signal Entity

#### *group SignalEntity*

SignalClass. Interface to digital pins configured to produce square wave signals. This class is designed to allow for square waves at various frequencies and duty cycles. Control is defined by specifying the wave period as (T3Time) and the active portion of the cycle as (T2Time). See the entity overview section of the reference for more detail regarding the timing.

```
void signal_setEnable (unsigned int *id, struct Result *result, const int index, const unsigned char enable)
```

Enable/Disable the signal output.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enable** – True to enable, false to disable

**Returns**

Returns common entity return values

void **signal\_getEnable** (unsigned int \*id, struct Result \*result, const int index)

Get the Enable/Disable of the signal.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **signal\_setInvert** (unsigned int \*id, struct Result \*result, const int index, const unsigned char invert)

Invert the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **invert** – to invert, false for normal mode.

#### Returns

Returns common entity return values

void **signal\_getInvert** (unsigned int \*id, struct Result \*result, const int index)

Get the invert status the signal output.

Normal mode is High on t0 then low at t2. Inverted mode is Low at t0 on period start and high at t2.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

void **signal\_setT3Time** (unsigned int \*id, struct Result \*result, const int index, const unsigned int t3\_nsec)

Set the signal period or T3 in nanoseconds.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **t3\_nsec** – Integer not larger than unsigned 32 bit max value representing the wave period in nanoseconds.

**Returns**

Returns common entity return values

**void signal\_getT3Time (unsigned int \*id, struct Result \*result, const int index)**

Get the signal period or T3 in nanoseconds.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void signal\_setT2Time (unsigned int \*id, struct Result \*result, const int index, const unsigned int t2\_nsec)**

Set the signal active period or T2 in nanoseconds.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **t2\_nsec** – Integer not larger than unsigned 32 bit max value representing the wave active period in nanoseconds.

**Returns**

Returns common entity return values

**void signal\_getT2Time (unsigned int \*id, struct Result \*result, const int index)**

Get the signal active period or T2 in nanoseconds.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

### 3.6.17 Store Entity

#### group **StoreEntity**

StoreClass. The store provides a flat file system on modules that have storage capacity. Files are referred to as slots and they have simple zero-based numbers for access. Store slots can be used for generalized storage and commonly contain compiled reflex code (files ending in .map) or templates used by the system. Slots simply contain bytes with no expected organization but the code or use of the slot may impose a structure. Stores have fixed indices based on type. Not every module contains a store of each type. Consult the module datasheet for details on which specific stores are implemented, if any, and the capacities of implemented stores.

```
void store_getSlotState (unsigned int *id, struct Result *result, const int index, const unsigned char slot)
```

Get slot state.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **state** – true: enabled, false: disabled.

#### Returns

Returns common entity return values

```
void store_loadSlot (unsigned int *id, struct Result *result, const int index, const int slot, unsigned char *buffer, const int bufferLength)
```

Load the slot.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **pData** – The data.
- **length** – The data length.

#### Returns

Returns common entity return values

```
void store_unloadSlot (unsigned int *id, struct Result *result, const int index, const int slot, unsigned char *buffer, const int bufferLength)
```

Unload the slot data.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **pData** – Byte array that the unloaded data will be placed into.
- **dataLength** – The length of pData buffer in bytes. This is the maximum number of bytes that should be unloaded.
- **unloadedLength** – Length of data that was unloaded. Unloaded length will never be larger than dataLength.
- **slot** – The slot number.

#### Returns

Returns common entity return values

---

`void store_slotEnable (unsigned int *id, struct Result *result, const int index, const unsigned char slot)`  
Enable slot.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.

#### Returns

Returns common entity return values

`void store_slotDisable (unsigned int *id, struct Result *result, const int index, const unsigned char slot)`  
Disable slot.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.

#### Returns

Returns common entity return values

`void store_getSlotCapacity (unsigned int *id, struct Result *result, const int index, const unsigned char slot)`

Get the slot capacity. Returns the Capacity of the slot, i.e. The number of bytes it can hold.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **capacity** – The slot capacity.

#### Returns

Returns common entity return values

`void store_getSlotSize (unsigned int *id, struct Result *result, const int index, const unsigned char slot)`

Get the slot size. The slot size represents the size of the data currently filling the slot in bytes.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number.
- **size** – The slot size.

**Returns**

Returns common entity return values

```
void store_getSlotLocked (unsigned int *id, struct Result *result, const int index, const unsigned char slot)
```

Gets the current lock state of the slot Allows for write protection on a slot.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **lock** – Variable to be filed with the locked state.

**Returns**

Returns common entity return values

```
void store_setSlotLocked (unsigned int *id, struct Result *result, const int index, const unsigned char slot, const unsigned char lock)
```

Sets the locked state of the slot Allows for write protection on a slot.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **slot** – The slot number
- **lock** – state to be set.

**Returns**

Returns common entity return values

### 3.6.18 System Entity

*group SystemEntity*

SystemClass. The System class provides access to the core settings, configuration and system information of the BrainStem module. The class provides access to the model type, serial number and other static information as well as the ability to set boot reflexes, toggle the user LED, as well as affect module and router addresses etc. The most common brainstem example uses the system entity to blink the User LED.

```
void system_getModule (unsigned int *id, struct Result *result)
```

Get the current address the module uses on the BrainStem network.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

```
void system_getModuleBaseAddress (unsigned int *id, struct Result *result)
```

Get the base address of the module. Software offsets and hardware offsets are added to this base address to produce the effective module address.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

```
void system_setRouter (unsigned int *id, struct Result *result, const unsigned char address)
```

Set the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network. This setting must be saved and the board reset before the setting becomes active. Warning: changing the router address may cause the module to “drop off” the BrainStem network if the new router address is not in use by a BrainStem module. Please review the BrainStem network fundamentals before modifying the router address.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **address** – The router address to be used.

**Returns**

Returns common entity return values

```
void system_getRouter (unsigned int *id, struct Result *result)
```

Get the router address the module uses to communicate with the host and heartbeat to in order to establish the BrainStem network.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

```
void system_setHBIInterval (unsigned int *id, struct Result *result, const unsigned char interval)
```

Set the delay between heartbeat packets which are sent from the module. For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments Valid values are 1-255; default is 10 (256 milliseconds).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **interval** – The desired heartbeat delay.

**Returns**

Returns common entity return values

```
void system_getHBInterval (unsigned int *id, struct Result *result)
```

Get the delay between heartbeat packets which are sent from the module. For link modules, these heartbeat are sent to the host. For non-link modules, these heartbeats are sent to the router address. Interval values are in 25.6 millisecond increments.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

```
void system_setLED (unsigned int *id, struct Result *result, const unsigned char bOn)
```

Set the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **bOn** – true: turn the LED on, false: turn LED off.

#### Returns

Returns common entity return values

```
void system_getLED (unsigned int *id, struct Result *result)
```

Get the system LED state. Most modules have a blue system LED. Refer to the module datasheet for details on the system LED location and color.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

```
void system_setBootSlot (unsigned int *id, struct Result *result, const unsigned char slot)
```

Set a store slot to be mapped when the module boots. The boot slot will be mapped after the module boots from powers up, receives a reset signal on its reset input, or is issued a software reset command. Set the slot to 255 to disable mapping on boot.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **slot** – The slot number in aSTORE\_INTERNAL to be marked as a boot slot.

#### Returns

Returns common entity return values

```
void system_getBootSlot (unsigned int *id, struct Result *result)
```

Get the store slot which is mapped when the module boots.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void system\_getVersion (unsigned int \*id, struct Result \*result)**

Get the modules firmware version number. The version number is packed into the return value. Utility functions in the aVersion module can unpack the major, minor and patch numbers from the version number which looks like M.m.p.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**void system\_getModel (unsigned int \*id, struct Result \*result)**

Get the module's model enumeration. A subset of the possible model enumerations is defined in BrainStem.h under "BrainStem model codes". Other codes are be used by Acroname for proprietary module types.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void system\_getHardwareVersion (unsigned int \*id, struct Result \*result)**

Get the module's hardware revision information. The content of the hardware version is specific to each Acroname product and used to indicate behavioral differences between product revisions. The codes are not well defined and may change at any time.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void system\_getSerialNumber (unsigned int \*id, struct Result \*result)**

Get the module's serial number. The serial number is a unique 32bit integer which is usually communicated in hexadecimal format.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

void **system\_save** (unsigned int \*id, struct Result \*result)

Save the system operating parameters to the persistent module flash memory. Operating parameters stored in the system flash will be loaded after the module reboots. Operating parameters include: heart-beat interval, module address, module router address

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

void **system\_reset** (unsigned int \*id, struct Result \*result)

Reset the system.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

void **system\_logEvents** (unsigned int \*id, struct Result \*result)

Saves system log events to a slot defined by the module (usually ram slot 0).

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

void **system\_getUptime** (unsigned int \*id, struct Result \*result)

Get the module's accumulated uptime in minutes

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

void **system\_getTemperature** (unsigned int \*id, struct Result \*result)

Get the module's current temperature in micro-C

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

```
void system_getMinimumTemperature (unsigned int *id, struct Result *result)
```

Get the module's minimum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

```
void system_getMaximumTemperature (unsigned int *id, struct Result *result)
```

Get the module's maximum temperature ever recorded in micro-C (uC) This value will persists through a power cycle.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

```
void system_getInputVoltage (unsigned int *id, struct Result *result)
```

Get the module's input voltage.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

```
void system_getInputCurrent (unsigned int *id, struct Result *result)
```

Get the module's input current.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

```
void system_getModuleHardwareOffset (unsigned int *id, struct Result *result)
```

Get the module hardware address offset. This is added to the base address to allow the module address to be configured in hardware. Not all modules support the hardware module address offset. Refer to the module datasheet.

#### Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

void **system\_setModuleSoftwareOffset** (unsigned int \*id, struct Result \*result, const unsigned char address)

Set the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **address** – The address for the module. Value must be even from 0-254.

#### Returns

Returns common entity return values

void **system\_getModuleSoftwareOffset** (unsigned int \*id, struct Result \*result)

Get the software address offset. This software offset is added to the module base address, and potentially a module hardware address to produce the final module address. You must save the system settings and restart for this to take effect. Please review the BrainStem network fundamentals before modifying the module address.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

void **system\_getRouterAddressSetting** (unsigned int \*id, struct Result \*result)

Get the router address system setting. This setting may not be the same as the current router address if the router setting was set and saved but no reset has occurred. Please review the BrainStem network fundamentals before modifying the module address.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

void **system\_routeToMe** (unsigned int \*id, struct Result \*result, const unsigned char bOn)

Enables/Disables the route to me function. This function allows for easy networking of BrainStem modules. Enabling (1) this function will send an I2C General Call to all devices on the network and request that they change their router address to the of the calling device. Disabling (0) will cause all devices on the BrainStem network to revert to their default address.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **bOn** – Enable or disable of the route to me function 1 = enable.

**Returns**

Returns common entity return values

**void system\_getPowerLimit (unsigned int \*id, struct Result \*result)**

Reports the amount of power the system has access to and thus how much power can be budgeted to sinking devices.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**void system\_getPowerLimitMax (unsigned int \*id, struct Result \*result)**

Gets the user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void system\_setPowerLimitMax (unsigned int \*id, struct Result \*result, const unsigned int power)**

Sets a user defined maximum power limit for the system. Provides mechanism for defining an unregulated power supplies capability.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **power** – Limit in milli-Watts (mW) to be set.

**Returns**

Returns common entity return values

**void system\_getPowerLimitState (unsigned int \*id, struct Result \*result)**

Gets a bit mapped representation of the factors contributing to the power limit. Active limit can be found through PowerDeliverClass::getPowerLimit().

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

void **system\_getUnregulatedVoltage** (unsigned int \*id, struct Result \*result)

Gets the voltage present at the unregulated port.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

void **system\_getUnregulatedCurrent** (unsigned int \*id, struct Result \*result)

Gets the current passing through the unregulated port.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

void **system\_getInputPowerSource** (unsigned int \*id, struct Result \*result)

Provides the source of the current power source in use.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

void **system\_getInputPowerBehavior** (unsigned int \*id, struct Result \*result)

Gets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

void **system\_setInputPowerBehavior** (unsigned int \*id, struct Result \*result, const unsigned char behavior)

Sets the systems input power behavior. This behavior refers to where the device sources its power from and what happens if that power source goes away.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **behavior** – An enumerated representation of behavior to be set.

**Returns**

Returns common entity return values

```
void system_getInputPowerBehaviorConfig (unsigned int *id, struct Result *result, unsigned char
                                         *buffer, const int bufferLength)
```

Gets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

```
void system_setInputPowerBehaviorConfig (unsigned int *id, struct Result *result, const unsigned
                                         int bufLength)
```

Sets the input power behavior configuration Certain behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

```
void system_getName (unsigned int *id, struct Result *result, unsigned char *buffer, const int bufferLength)
```

Gets a user defined name of the device. Helpful for identifying ports/devices in a static environment.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

```
void system_setName (unsigned int *id, struct Result *result, unsigned char *buffer, const int 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**

- **id** – The id assigned by the create stem vi.

- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **buffer** - Pointer to the start of a c style buffer to be transferred.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

**void system\_resetEntityToFactoryDefaults (unsigned int \*id, struct Result \*result)**

Resets the SystemClass Entity to it factory default configuration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** - Object containing aErrNone on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void system\_resetDeviceToFactoryDefaults (unsigned int \*id, struct Result \*result)**

Resets the device to it factory default configuration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** - Object containing aErrNone on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void system\_getLinkInterface (unsigned int \*id, struct Result \*result)**

Gets the link interface configuration. This refers to which interface is being used for control by the device.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** - Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void system\_setLinkInterface (unsigned int \*id, struct Result \*result, const unsigned char linkInterface)**

Sets the link interface configuration. This refers to which interface is being used for control by the device.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **linkInterface** – An enumerated representation of interface to be set.
  - 0 = Auto= systemLinkAuto
  - 1 = Control Port = systemLinkUSBControl
  - 2 = Hub Upstream Port = systemLinkUSBHub

**Returns**

Returns common entity return values

### 3.6.19 Temperature Entity

**group TemperatureEntity**

TemperatureClass. This entity is only available on certain modules, and provides a temperature reading in microcelsius.

**void temperature\_getValue (unsigned int \*id, struct Result \*result, const int index)**

Get the modules temperature in micro-C

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

**void temperature\_getValueMin (unsigned int \*id, struct Result \*result, const int index)**

Get the module's minimum temperature in micro-C since the last power cycle.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

**void temperature\_getValueMax (unsigned int \*id, struct Result \*result, const int index)**

Get the module's maximum temperature in micro-C since the last power cycle.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

**void temperature\_resetEntityToFactoryDefaults (unsigned int \*id, struct Result \*result, const int index)**

### 3.6.20 Timer Entity

*group TimerEntity*

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

**void timer\_getExpiration** (unsigned int \*id, struct Result \*result, const int index)

Get the currently set expiration time in microseconds. This is not a “live” timer. That is, it shows the expiration time originally set with setExpiration; it does not “tick down” to show the time remaining before expiration.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

**void timer\_setExpiration** (unsigned int \*id, struct Result \*result, const int index, const unsigned int usecDuration)

Set the expiration time for the timer entity. When the timer expires, it will fire the associated timer[index]() reflex.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **usecDuration** – The duration before timer expiration in microseconds.

#### Returns

Returns common entity return values

**void timer\_getMode** (unsigned int \*id, struct Result \*result, const int index)

Get the mode of the timer which is either single or repeat mode.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

**void timer\_setMode** (unsigned int \*id, struct Result \*result, const int index, const unsigned char mode)

Set the mode of the timer which is either single or repeat mode.

#### Parameters

- **id** – The id assigned by the create stem vi.

- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **index** - The index of the entity in question.
- **mode** - The mode of the timer. aTIMER\_MODE\_REPEAT or aTIMER\_MODE\_SINGLE.

**Returns**

Returns common entity return values

**Returns**

::aErrNone - Action completed successfully.

### 3.6.21 UART Entity

#### *group UARTEntity*

UART Class. A UART is a “Universal Asynchronous Receiver/Transmitter. Many times referred to as a COM (communication), Serial, or TTY (teletypewriter) port.

The UART Class allows the enabling and disabling of the UART data lines.

**void uart\_setEnable (unsigned int \*id, struct Result \*result, const int index, const unsigned char bEnable)**  
Enable the UART channel.

**Parameters**

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **index** - The index of the entity in question.
- **bEnable** - False or 0 = Disabled, True or 1 = Enabled

**Returns**

Returns common entity return values

**void uart\_getEnable (unsigned int \*id, struct Result \*result, const int index)**  
Get the UART channel state.

**Parameters**

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** - The index of the entity in question.

**Returns**

Returns common entity return values

### 3.6.22 USB Entity

#### group **USBEntity**

USBClass. The USB class provides methods to interact with a USB hub and USB switches. Different USB hub products have varying support; check the datasheet to understand the capabilities of each product.

**void `usb_setPortEnable`** (unsigned int \***id**, struct Result \***result**, const unsigned char **channel**)

Enable both power and data lines for a port.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setPortDisable`** (unsigned int \***id**, struct Result \***result**, const unsigned char **channel**)

Disable both power and data lines for a port.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setDataEnable`** (unsigned int \***id**, struct Result \***result**, const unsigned char **channel**)

Enable the only the data lines for a port without changing the state of the power line.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setDataDisable`** (unsigned int \***id**, struct Result \***result**, const unsigned char **channel**)

Disable only the data lines for a port without changing the state of the power line.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **channel** – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setHiSpeedDataEnable`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone on success. Non-zero error code on failure.
- `channel` – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setHiSpeedDataDisable`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, Hi-Speed (2.0) only.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone on success. Non-zero error code on failure.
- `channel` – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setSuperSpeedDataEnable`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Enable the only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone on success. Non-zero error code on failure.
- `channel` – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setSuperSpeedDataDisable`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Disable only the data lines for a port without changing the state of the power line, SuperSpeed (3.0) only.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone on success. Non-zero error code on failure.
- `channel` – The USB sub channel.

#### Returns

Returns common entity return values

**void `usb_setPowerEnable`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Enable only the power line for a port without changing the state of the data lines.

#### Parameters

- `id` – The id assigned by the create stem vi.

- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **channel** - The USB sub channel.

#### Returns

Returns common entity return values

void **usb\_setPowerDisable** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Disable only the power line for a port without changing the state of the data lines.

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone on success. Non-zero error code on failure.
- **channel** - The USB sub channel.

#### Returns

Returns common entity return values

void **usb\_getPortCurrent** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get the current through the power line for a port.

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** - The USB channel current in micro-amps (1 == 1e-6A).

#### Returns

Returns common entity return values

void **usb\_getPortVoltage** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get the voltage on the power line for a port.

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** - The USB channel voltage in microvolts (1 == 1e-6V).

#### Returns

Returns common entity return values

void **usb\_getHubMode** (unsigned int \*id, struct Result \*result)

Get a bit mapped representation of the hubs mode; see the product datasheet for mode mapping and meaning.

#### Parameters

- **id** - The id assigned by the create stem vi.
- **result** - Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

**void `usb_setHubMode`** (unsigned int \*id, struct Result \*result, const unsigned int mode)

Set a bit mapped hub state; see the product datasheet for state mapping and meaning.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone on success. Non-zero error code on failure.
- `mode` – The USB hub mode.

#### Returns

Returns common entity return values

**void `usb_clearPortErrorStatus`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Clear the error status for the given port.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone on success. Non-zero error code on failure.
- `channel` – The port to clear error status for.

#### Returns

Returns common entity return values

**void `usb_getUpstreamMode`** (unsigned int \*id, struct Result \*result)

Get the upstream switch mode for the USB upstream ports. Returns auto, port 0 or port 1.

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

**void `usb_setUpstreamMode`** (unsigned int \*id, struct Result \*result, const unsigned char mode)

Set the upstream switch mode for the USB upstream ports. Values are usbUpstreamModeAuto, usbUpstreamModePort0 and usbUpstreamModePort1

#### Parameters

- `id` – The id assigned by the create stem vi.
- `result` – Object containing aErrNone on success. Non-zero error code on failure.
- `mode` – The Upstream port mode.

#### Returns

Returns common entity return values

**void `usb_getUpstreamState`** (unsigned int \*id, struct Result \*result)

Get the upstream switch state for the USB upstream ports. Returns 2 if no ports plugged in, 0 if the mode is set correctly and a cable is plugged into port 0, and 1 if the mode is set correctly and a cable is plugged into port 1.

#### Parameters

- `id` – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void usb\_setEnumerationDelay (unsigned int \*id, struct Result \*result, const unsigned int ms\_delay)**

Set the inter-port enumeration delay in milliseconds.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **ms\_delay** – Millisecond delay in 100mS increments (100, 200, 300 etc.)

**Returns**

Returns common entity return values

**void usb\_getEnumerationDelay (unsigned int \*id, struct Result \*result)**

Get the inter-port enumeration delay in milliseconds.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

**void usb\_setPortCurrentLimit (unsigned int \*id, struct Result \*result, const unsigned char channel, const unsigned int microamps)**

Set the current limit for the port. If the set limit is not achievable, devices will round down to the nearest available current limit setting. This setting can be saved with a stem.system.save() call.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – USB downstream channel to limit.
- **microamps** – The current limit setting.

**Returns**

Returns common entity return values

**void usb\_getPortCurrentLimit (unsigned int \*id, struct Result \*result, const unsigned char channel)**

Get the current limit for the port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** – The current limit setting.

**Returns**

Returns common entity return values

```
void usb_setPortMode (unsigned int *id, struct Result *result, const unsigned char channel, const
                      unsigned int mode)
```

Set the mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device datasheet for a complete list of capabilities. Some devices use a common bit mapping for port mode at usbPortMode

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – USB downstream channel to set the mode on.
- **mode** – The port mode setting as packet bit mask.

**Returns**

Returns common entity return values

```
void usb_getPortMode (unsigned int *id, struct Result *result, const unsigned char channel)
```

Get the current mode for the Port. The mode is a bitmapped representation of the capabilities of the usb port. These capabilities change for each of the BrainStem devices which implement the usb entity. See your device datasheet for a complete list of capabilities. Some devices implement a common bit mapping for port mode at usbPortMode

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **mode** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

```
void usb_getPortState (unsigned int *id, struct Result *result, const unsigned char channel)
```

Get the current State for the Port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **state** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

```
void usb_getPortError (unsigned int *id, struct Result *result, const unsigned char channel)
```

Get the current error for the Port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **error** – The port mode setting. Mode will be filled with the current setting. Mode bits that are not used will be marked as don't care

**Returns**

Returns common entity return values

```
void usb_SetUpstreamBoostMode (unsigned int *id, struct Result *result, const unsigned char setting)
```

Set the upstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **setting** – Upstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

```
void usb_SetDownstreamBoostMode (unsigned int *id, struct Result *result, const unsigned char setting)
```

Set the downstream boost mode. Boost mode increases the drive strength of the USB data signals (power signals are not changed). Boosting the data signal strength may help to overcome connectivity issues when using long cables or connecting through “pogo” pins. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost. This setting is not applied until a stem.system.save() call and power cycle of the hub. Setting is then persistent until changed or the hub is reset. After reset, default value of 0% boost is restored.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **setting** – Downstream boost setting 0, 1, 2, or 3.

**Returns**

Returns common entity return values

```
void usb_getUpstreamBoostMode (unsigned int *id, struct Result *result)
```

Get the upstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

**Returns**

Returns common entity return values

---

`void usb_getDownstreamBoostMode (unsigned int *id, struct Result *result)`

Get the downstream boost mode. Possible modes are 0 - no boost, 1 - 4% boost, 2 - 8% boost, 3 - 12% boost.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.

#### Returns

Returns common entity return values

`void usb_getDownstreamDataSpeed (unsigned int *id, struct Result *result, const unsigned char channel)`

Get the current data transfer speed for the downstream port. The data speed can be Hi-Speed (2.0) or SuperSpeed (3.0) depending on what the downstream device attached is using

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **speed** – Filled with the current port data speed
  - N/A: usbDownstreamDataSpeed\_na = 0
  - Hi Speed: usbDownstreamDataSpeed\_hs = 1
  - SuperSpeed: usbDownstreamDataSpeed\_ss = 2

#### Returns

Returns common entity return values

`void usb_setConnectMode (unsigned int *id, struct Result *result, const unsigned char channel, const unsigned char mode)`

Sets the connect mode of the switch.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel.
- **mode** – The connect mode
  - usbManualConnect = 0
  - usbAutoConnect = 1

#### Returns

Returns common entity return values

`void usb_getConnectMode (unsigned int *id, struct Result *result, const unsigned char channel)`

Gets the connect mode of the switch.

#### Parameters

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **mode** – The current connect mode

**Returns**

Returns common entity return values

```
void usb_setCC1Enable (unsigned int *id, struct Result *result, const unsigned char channel, const  
                      unsigned char bEnable)
```

Set Enable/Disable on the CC1 line.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** -- USB channel.
- **bEnable** –
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

```
void usb_getCC1Enable (unsigned int *id, struct Result *result, const unsigned char channel)
```

Get Enable/Disable on the CC1 line.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** –
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

```
void usb_setCC2Enable (unsigned int *id, struct Result *result, const unsigned char channel, const  
                      unsigned char bEnable)
```

Set Enable/Disable on the CC2 line.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** -- USB channel.
- **bEnable** –
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

**void usb\_getCC2Enable** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get Enable/Disable on the CC1 line.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** –
  - Disabled: 0
  - Enabled: 1

**Returns**

Returns common entity return values

**void usb\_getCC1Current** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get the current through the CC1 for a port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

**Returns**

Returns common entity return values

**void usb\_getCC2Current** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get the current through the CC2 for a port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microamps** – The USB channel current in micro-amps (1 == 1e-6A).

**Returns**

Returns common entity return values

**void usb\_getCC1Voltage** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get the voltage of CC1 for a port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

**void `usb_getCC2Voltage`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get the voltage of CC2 for a port.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

#### Returns

Returns common entity return values

**void `usb_setSBUEnable`** (unsigned int \*id, struct Result \*result, const unsigned char channel, const unsigned char bEnable)

Enable/Disable only the SBU1/2 based on the configuration of the usbPortMode settings.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel.
- **bEnable** –
  - Disabled: 0
  - Enabled: 1

#### Returns

Returns common entity return values

**void `usb_getSBUEnable`** (unsigned int \*id, struct Result \*result, const unsigned char channel)

Get the Enable/Disable status of the SBU

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** – The enable/disable status of the SBU

#### Returns

Returns common entity return values

**void `usb_setCableFlip`** (unsigned int \*id, struct Result \*result, const unsigned char channel, const unsigned char bEnable)

Set Cable flip. This will flip SBU, CC and SS data lines.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel.
- **bEnable** –

- Disabled: 0
- Enabled: 1

**Returns**

Returns common entity return values

**void usb\_getCableFlip (unsigned int \*id, struct Result \*result, const unsigned char channel)**

Get Cable flip setting.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **pEnable** – The enable/disable status of cable flip.

**Returns**

Returns common entity return values

**void usb\_setAltModeConfig (unsigned int \*id, struct Result \*result, const unsigned char channel, const unsigned int configuration)**

Set USB Alt Mode Configuration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **channel** – The USB sub channel
- **configuration** – The USB configuration to be set for the given channel.

**Returns**

Returns common entity return values

**void usb\_getAltModeConfig (unsigned int \*id, struct Result \*result, const unsigned char channel)**

Get USB Alt Mode Configuration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **configuration** – The USB configuration for the given channel.

**Returns**

Returns common entity return values

**void usb\_getSBU1Voltage (unsigned int \*id, struct Result \*result, const unsigned char channel)**

Get the voltage of SBU1 for a port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

**void `usb_getSBU2Voltage`** (unsigned int \***id**, struct Result \***result**, const unsigned char **channel**)

Get the voltage of SBU2 for a port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **microvolts** – The USB channel voltage in micro-volts (1 == 1e-6V).

**Returns**

Returns common entity return values

### 3.6.23 USBSYSTEM Entity

**group USBSYSTEMEntity**

USBSYSTEM Class The USBSYSTEM class provides high level control of the lower level Port Class.

**void `usbsystem_getUpstream`** (unsigned int \***id**, struct Result \***result**, const int **index**)

Gets the upstream port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void `usbsystem_setUpstream`** (unsigned int \***id**, struct Result \***result**, const int **index**, const unsigned char **port**)

Sets the upstream port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **port** – The upstream port to set.

**Returns**

Returns common entity return values

**void `usbsystem_getEnumerationDelay`** (unsigned int \***id**, struct Result \***result**, const int **index**)

Gets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void usbsystem_setEnumerationDelay (unsigned int *id, struct Result *result, const int index, const
                                    unsigned int msDelay)
```

Sets the inter-port enumeration delay in milliseconds. Delay is applied upon hub enumeration.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **msDelay** – The delay in milliseconds to be applied between port enables

**Returns**

Returns common entity return values

```
void usbsystem_getDataRoleList (unsigned int *id, struct Result *result, const int index)
```

Gets the data role of all ports with a single call Equivalent to calling PortClass::getDataRole() on each individual port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void usbsystem_getEnabledList (unsigned int *id, struct Result *result, const int index)
```

Gets the current enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

```
void usbsystem_setEnabledList (unsigned int *id, struct Result *result, const int index, const unsigned
                               int enabledList)
```

Sets the enabled status of all ports with a single call. Equivalent to calling PortClass::setEnabled() on each port.

**Parameters**

- **id** – The id assigned by the create stem vi.

- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **enabledList** – Bit packed representation of the enabled status for all ports to be applied.

**Returns**

Returns common entity return values

```
void usbsystem_getModeList (unsigned int *id, struct Result *result, const int index, unsigned int *buffer,  
                           const int bufferLength)
```

Gets the current mode of all ports with a single call. Equivalent to calling PortClass::getMode() on each port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

```
void usbsystem_setModeList (unsigned int *id, struct Result *result, const int index, const unsigned int  
                           bufLength)
```

Sets the mode of all ports with a single call. Equivalent to calling PortClass::setMode() on each port

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

```
void usbsystem_getStateList (unsigned int *id, struct Result *result, const int index, unsigned int  
                           *buffer, const int bufferLength)
```

Gets the state for all ports with a single call. Equivalent to calling PortClass::getState() on each port.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed

- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

**void usbsystem\_getPowerBehavior** (unsigned int \*id, struct Result \*result, const int index)

Gets the behavior of the power manager. The power manager is responsible for budgeting the power of the system. i.e. What happens when requested power greater than available power.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

**Returns**

Returns common entity return values

**void usbsystem\_setPowerBehavior** (unsigned int \*id, struct Result \*result, const int index, const unsigned char behavior)

Sets the behavior of how available power is managed. i.e. What happens when requested power is greater than available power.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **behavior** – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

**Returns**

Returns common entity return values

**void usbsystem\_getPowerBehaviorConfig** (unsigned int \*id, struct Result \*result, const int index, unsigned int \*buffer, const int bufferLength)

Gets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

```
void usbsystem_setPowerBehaviorConfig (unsigned int *id, struct Result *result, const int index,
                                      const unsigned int bufLength)
```

Sets the current power behavior configuration Certain power behaviors use a list of ports to determine priority when budgeting power.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bufLength** – Length of the buffer to be transferred.

#### Returns

Returns common entity return values

```
void usbsystem_getDataRoleBehavior (unsigned int *id, struct Result *result, const int index)
```

Gets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.

#### Returns

Returns common entity return values

```
void usbsystem_setDataRoleBehavior (unsigned int *id, struct Result *result, const int index, const
                                    unsigned char behavior)
```

Sets the behavior of how upstream and downstream ports are determined. i.e. How do you manage requests for data role swaps and new upstream connections.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **behavior** – An enumerated representation of behavior. Available behaviors are product specific. See the reference documentation.

#### Returns

Returns common entity return values

```
void usbsystem_getDataRoleBehaviorConfig (unsigned int *id, struct Result *result, const int index,
                                         unsigned int *buffer, const int bufferLength)
```

Gets the current data role behavior configuration Certain data role behaviors use a list of ports to determine priority host priority.

#### Parameters

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone on success. Non-zero error code on failure.

- **index** – The index of the entity in question.
- **buffer** – pointer to the start of a c style buffer to be filled
- **bufLength** – Length of the buffer to be filed
- **unloadedLength** – Length that was actually received and filled.

**Returns**

Returns common entity return values

```
void usbsystem_setDataRoleBehaviorConfig(unsigned int *id, struct Result *result, const int index,  
const unsigned int bufLength)
```

Sets the current data role behavior configuration Certain data role behaviors use a list of ports to determine host priority.

**Parameters**

- **id** – The id assigned by the create stem vi.
- **result** – Object containing aErrNone and the requested value on success. Non-zero error code on failure.
- **index** – The index of the entity in question.
- **bufLength** – Length of the buffer to be transferred.

**Returns**

Returns common entity return values

**Warning:** doxygenfunction: Cannot find function “usbsystem\_getSelectorMode” in doxygen xml output for project “BrainStem2\_LabVIEW” from directory: doxml/labVIEW/xml

**Warning:** doxygenfunction: Cannot find function “usbsystem\_setSelectorMode” in doxygen xml output for project “BrainStem2\_LabVIEW” from directory: doxml/labVIEW/xml

```
void usbsystem_resetEntityToFactoryDefaults(unsigned int *id, struct Result *result, const int  
index)
```

## 3.7 Reflex Language Reference

**re-flex:** 'rē fleks/

(*noun: reflex; plural noun: reflexes*) An action that is performed as a response to a stimulus and without conscious thought.

### 3.7.1 Introduction

The Reflex language is a high level, C-like language designed to support rapid development of embedded applications that bridge the gap between hardware and software. The reflex language leverages the BrainStem command protocol to make it easy for developers to turn a network of devices into a functional system.

The following sections in the BrainStem Language documentation will dive into the Reflex language.

### 3.7.2 Working with Reflex files

Writing reflexes requires a few tools. You'll need access to your favorite text editor, and the programs provided in the BrainStem support download. These tools include:

- arc - The reflex language compiler
- Python - The BrainStem python library can now load and unload reflex files.

The BrainStem support download can be obtained from the [download](#) page on the [acroname](#) website.

#### A Note about Directories

Acroname prefers to distribute a set of directories you can place anywhere on your system rather than rely on platform specific installation routines, and default program locations.

The BrainStem Support download includes the following directories:

##### • **acroname** - Top level folder

- **bin** - Executable apps and libraries
- **development** - Examples and API libraries (Python and C++)
- **include** - Header files for reflexes

#### Arc, the Acroname Reflex Compiler

Arc can be run from the command line. It will compile reflex source files from within the **bin** folder or from a relative file path (absolute paths will not work). The reflex files will be compiled into map files and placed into the local directory. Map files, which are binary byte code files are executed by the Reflex Virtual Machine on BrainStem modules.

```
$> ./arc -h
```

This will print information about using the arc compiler. Arc has a couple of useful flags useful for working with reflex file output, but first let us look at the basic command line for compiling reflex files.

```
$> ./arc ../development/reflex_examples/Blink_LED.reflex
```

This will read in the Blink\_LED.reflex source file located in the **development/reflex\_examples** folder, and produce the compiled result at **bin/Blink\_LED.map**.

```
$>./arc -p ../development/reflex_examples/Blink_LED.reflex
```

This will read in the Blink\_LED.reflex source file in the **development/reflex\_examples** directory, and output the resulting tokenized output of the lexer portion of the compilation process.

```
$>./arc -a ../development/reflex_examples/Blink_LED.reflex
```

This will read in the Blink\_LED.reflex source file located in the **development/reflex\_examples** folder, and will output the AST for the reflex.

```
$>./arc -d Blink_LED.map
```

This will read in the compiled Blink\_LED.map binary located in the **bin** folder, and will generate a .dsm human readable disassembly file in the **bin** directory. See [Appendix II: Reflex map file Disassembly](#) for more information about understanding the format of the disassembly.

### Loading reflex .map files with ReflexLoader

**ReflexLoader** is our new Command Line Interface (CLI) tool for loading .map files into any BrainStem device. It can be found in the BrainStem development kit under the **bin** folder.

```
$>./ReflexLoader -H
```

This will print the usage information for ReflexLoader. If you are having trouble getting something to work this is one of the best places to look for more information. One thing that is important to make note of is that specifying a device '-d' is required. Additionally, that means that you must also have a device (SN), store and slot specified. Failure to do so will cause the program to return an error. With that said a command is also required if you were wanting the program to actually do something; however, failure to include one will not cause an error.

```
$>./ReflexLoader -L -i /location/of/file.map -d 0x40F5849A INTERNAL 0
```

This command will load '-L' the input file '-i' /location/of/file.map to the device '-d' into the INTERNAL store at slot 0.

Unloading a slot is just as easy but instead of using the '-L' and '-i' you would use '-U' and '-o' to specify where you would like the file to go. ReflexLoader will not create any file paths that do not exist so you must give a valid path.

```
$>./ReflexLoader -E -d 0x40F5849A RAM 0
```

This command will enable '-E' the reflex that is loaded into the RAM store at slot 0 in the specified device 0x40F5849A. Keep in mind that although you can load a reflex into RAM it will not survive a power cycle as it is volatile memory.

To disable this slot just swap the '-E' for '-D'

```
$>./ReflexLoader -B -d 0x40F5849A INTERNAL 0
```

This last command will make a slot bootable. Once this has been set and a power cycle occurs the BrainStem device will automatically enable the reflex at the designated slot. In order to disable a boot slot you will need to replace the slot parameter '0' in this case with 255.

## Loading reflex .map files with Python

The BrainStem python library can be used to load compiled map files onto BrainStem modules. To load .map files with python you will need to follow the installation instructions of the [getting started guide](#) in the Python section of this reference.

To load a reflex map file into a slot on your brainstem module. First compile the reflex with arc. The resulting map file can be found in your aObject directory. Then start the python interpreter, and import the brainstem package. Instantiate your module and connect. Once you're connected, the following code block will show you how to load, enable and disable the .map file.

```
>>> fh = open('path/to/map/mymap.map', 'rb')
>>> mapfile = fh.read()
>>> fh.close()
>>> stem.store[0].loadSlot(0, mapfile, len(mapfile)) #loads map into store 0 slot 0
0
>>> stem.store[0].slotEnable(0)
0
>>> stem.store[0].slotDisable(0)
```

The above code simply opens the map file for reading in binary mode, reads the contents of the file into the mapfile variable, loads that data into store 0 slot 0 and enables and disables the slot.

## Boot Reflexes

You can also set your stem to enable the map file on boot. To do this you will need to load the map file into a slot on the internal store of your module (the internal store is index 0). Then set the boot slot on the module by accessing the setBootSlot procedure of the system module

```
>>> stem.system.setBootSlot(0)
0
>>> stem.system.setBootSlot(255) # will disable the slot from being enabled on boot.
```

If you do not have python installed on your system, you can also use the aConsole utility to load .map files.

### 3.7.3 A Basic “Hello World” Example.

```
1 #include <a40PinModule.reflex>
2 #define ON 1
3 #define OFF 0
4
5 a40PinModule stem;
6
7 reflex mapEnable()
8 {
9     stem.system.setLED(ON);
10 }
11
12 reflex mapDisable()
13 {
14     stem.system.setLED(OFF);
15 }
```

## Wait Where's the “Hello World?”

We don't actually print “hello world” in this example. Since we're working with a BrainStem, the equivalent action is to turn on and off the User LED. This is exactly what the reflex above accomplishes. Next we will break this example down line by line to illuminate some of the interesting bits. Finally we will look at a slightly more complex example which outlines one of the ways the BrainStem architecture is different (and better) than other embedded languages you may be familiar with.

## Includes

Line one includes the BrainStem module definition file we will be using in the example. There is generally one of these for each type of BrainStem module you may encounter. See [Appendix I](#) to have a look at one of these files. The BrainStem module definition file tells the reflex compiler what the BrainStem is capable of doing.

## Defines

We follow a C-like convention for the majority of our language constructs, and the `#define` preprocessor directive is one of the preprocessor directives we support. Lines 2 and 3 define some human readable names for the UserLED states.

## The Module declaration

```
a40PinModule stem;
```

The module declaration creates a named “instance” of a BrainStem. Instance is in quotes because there is no real concept of an object in the reflex language, but in essence the declaration provides us with a way to refer to a particular BrainStem. Each module in a BrainStem network has a unique module address, by default `a40PinModule stem;` without an argument refers to the module on which the reflex file is loaded, while a module declaration with a module address, like `a40PinModule stem2(8);` refers to the module with address 8 and is not necessarily the module on which the reflex is loaded.

## Keywords and builtins

We try to minimize the number of keywords, and builtin bits that you as a user have to learn, but there are a couple of keywords in the reflex language that are used all the time. `reflex` defines a top level routine that is attached to an event or entity in the BrainStem system. There are 4 built in reflexes that can be defined by the user to perform behaviors on certain system events. The first two are represented in this example `mapEnable` and `mapDisable` are startup and teardown reflexes that are called whenever a reflex file is enabled and disabled. The second set of built in routines is `linkUp` and `linkDown`, and are called when a connection to a host is established or disconnected.

A reflex declaration looks like the following snippet. It starts with the keyword `reflex`, then declares the entity for which the reflex is executed, is enclosed with `{` and `}`. We will see an example of an entity reflex declaration in the second code example.

```
reflex 'entity'()
{
    ...
}
```

There is no `main` routine in the reflex world. When you enable a map, you are essentially attaching behaviors to certain entities or events which occur in the system. Enable does not necessarily execute code, code execution is a side effect of defining a behavior to the `mapEnable` reflex. In the case of the example, this is what we want.

### There is no `main`, only Zuul!

It is worth reiterating this point because it is one of the main issues new users encounter with the Reflex system. Most programming languages have a single entry point, typically called “main”. There is no such concept when writing reflexes. Reflexes are meant to respond to changing conditions within the BrainStem system. As such, relying on `mapEnable` as a main-like routine is a discouraged practice, and can have negative consequences for the speed and performance of your reflex. So, no main ... got it. Moving on.

### The Rest

The rest of the example is relatively straight forward. The `mapEnable` routine as defined on lines 7–10 turns on the user LED when the mapfile is enabled, and the `mapDisable` routine on lines 12–15 turns the LED off when the mapfile is disabled. The final interesting bit is the command sent to the BrainStem to enable or disable the user LED.

```
stem.system.setLED(ON);
```

This command is fairly typical of many BrainStem commands. It consists of three parts, the module where we will send the command, in this case the current module, the BrainStem entity `system` to which the `setLED` command belongs, and the argument `ON` which sets the state of the user LED.

Next we'll take a look at a more complex example, where we blink the LED multiple times, which highlights one of the main strengths of the BrainStem Virtual Machine: a BrainStem is a multiprocess machine, designed to do more than one thing at a time.

---

**Note:** A note about includes. We follow the convention that an include file in <> brackets comes from the `aInclude` directory, and is usually an 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.

---

### 3.7.4 Blink My LED Example

This second reflex code example highlights some of the syntactical differences in the reflex language from that of basic Ansi C, and introduces you to a couple of the fundamental reflex concepts that make it different from many other embedded languages. In this example we will see the preferred method for writing a timed loop, and we will also introduce the `ScratchPad`, and its use as shared variable.

Here is the entire reflex file. Enabling the reflex will start the user LED blinking at a half second interval until we disable it.

```
1 // file: flashmyled.reflex
2
3 #include <a40PinModule.reflex>
4 a40PinModule stem;
5
6 // 1/2 second delay
7 #define DELAY 500000
```

(continues on next page)

(continued from previous page)

```

8 #define ON 1
9 #define OFF 0
10
11 pad[0:0] unsigned char state;
12
13 reflex mapEnable()
{
14     state = ON;
15     stem.system.setLED(state);
16     stem.timer[0].setMode(timerModeRepeat);
17     stem.timer[0].setExpiration(DELAY);
18
19 } // end of mapEnable
20
21
22 reflex timer[0].expiration()
{
23     if (state == ON) {
24         state = OFF;
25     } else {
26         state = ON;
27     }
28     stem.system.setLED(state);
29
30 } // end of timer reflex
31
32
33 reflex mapDisable()
{
34     stem.timer[0].setExpiration(0);
35     stem.timer[0].setMode(timerModeSingle);
36     stem.system.setLED(OFF);
37
38 }
39

```

## The Timer Entity

Timer is a BrainStem entity class that every BrainStem module includes. It allows users to schedule events to be executed at some time in the future. Its range is approximately 1 microsecond to 4200 seconds, and its resolution is in microseconds. There is more information about the timer entity in the BrainStem Entities reference section, however there are two commands that we will use in this example.

`timer setExpiration` is the method of scheduling a timer event to occur in the future, it takes one argument which is a 32bit integer representing the number of microseconds in the future that the timer should execute.

`timer setMode` is the second command we use in the example. Timers have two modes, single (which is the default) and repeat. Single timers execute once, and, as you guessed, repeat timers execute on the interval defined in `setExpiration`. For a timer in repeat mode, setting the expiration to 0 will stop execution of the timer event.

```

stem.timer[0].setMode(timerModeRepeat);
stem.timer[0].setExpiration(DELAY);

```

Lines 17 and 18 in the code listing set up a repeat timer with an interval of 500000 microseconds or 1/2 second. You will have noticed the array like syntax of the command, BrainStem entities sometimes come in groups. There are generally 4 or 8 timers in a BrainStem module, and we address each timer by its index just as if we had set up an array of timers.

## The Timer Reflex

```
reflex timer[0].expiration()
{
    if (state == ON) {
        state = OFF;
    } else {
        state = ON;
    }
    stem.system.setLED(state);
}
```

This is the first time we have seen an “entity” reflex declaration. This code defines the behavior that should occur when the timer expires and the timer triggers.

```
reflex timer[0].expiration() { ... }
```

Notice that we declare the timer index we want to attach this behavior to in the declaration. Timers don’t receive any arguments, but other Entity types like analogs and digitals do.

In this case we are setting the User LED either on or off depending on the current state. This leads naturally to the question of how to share state between reflex routines.

## The ScratchPad

Individual reflex routines are essentially separate processes within the BrainStem system. They have their own execution space, and variable scope. The only way to share information between two reflex routines is to use the shared data space we call the ScratchPad.

```
pad[0:0] unsigned char state;
```

Line 11 in the code listing declares a single byte in the pad for use as our shared LED state variable. We define it as an unsigned char, and we declare its extents with the array-like syntax [0:0]. If we were declaring a short or an int their extents might be something like [0:1] and [0:3] respectively. When declaring multiple pad variables, the extents must be mutually exclusive. In other words, variables can not overlap. Declaring one variable with the extent [0:3] and another with the extent [2:5] would not make any sense. Generally pads are around 300 bytes, but this is module specific, so check your data sheet.

Now that we’ve declared our shared variable, it can be used in any of our reflexes.

This doesn’t seem correct. If it is not thread safe there is no guarantee that a read won’t happen in the middle of a write: In general the pad is not safe from multiple concurrent modifications, so the idiomatic pattern is to have one producer which modifies a pad variable, and one or more consumers which read it.

## The Rest

The rest of the example should be familiar to you if you followed us from the hello world example. The `mapEnable` routine (lines 13–20) sets up our state, turns on the userLED, and sets the timer to expire at 1/2 second intervals. The `timer[0].expiration` routine (lines 22–31) toggles the userLED from off to on or vice versa. Finally the `mapDisable` routine (lines 33–39) shuts down the timer, and turns off the userLED.

Now that we have traced through two basic examples, the remainder of the reflex reference describes the reflex language in-depth. For more information about BrainStem entities and capabilities see the Entities section of this reference. The entity concept is useful for those wishing to write host code in C, C++, or Python.

---

**Note:** There are few native datatypes in the Reflex language, essentially these are all numeric values. They are `char`, `short` and `int`, and they all come in `signed` and `unsigned` flavors. Types are signed by default unless specified with `unsigned` keyword.

---

### 3.7.5 Built in reflex origins

BrainStem devices have a number of built in reflex origins that can be mapped to provide reflex functionality when certain system events occur. Each reflex routine in the reflex file is declared similarly. Built in origins do not have arguments passed in. for Example:

```
reflex mapEnable() {
    // ... Reflex statements
}
```

#### *mapEnable*

This reflex is triggered when the reflex map file is enabled.

#### *transportUp*

On a reset of a module this reflex is triggered when its transport becomes ready (USB is enumerated, TCPIP has acquired an address).

#### *linkUp*

A BrainStem link to the host has been created.

#### *linkDown*

The BrainStem link has been disconnected.

#### *transportDown*

The transport is currently down (USB no upstream connection, TCPIP has lost IP address, or has been disconnected)

#### *mapDisable*

Actions to be executed before the map file is disabled.

## Map Enable

The map enable reflex is the primary entry point for reflex code when a map file is enabled on a module slot. The reflex machine within the module first adds each of the reflexes defined in the map file to their respective origins, and then executes the code inside the mapEnable reflex.

Example:

```
reflex mapEnable() {  
    // ...  
}
```

## Transport Up

Transport Up is called when the Module detects that its primary transport to the host is configured and ready for communication. On USB modules this is when the host has successfully enumerated the Module. In TCP/IP based modules, this event occurs when the stem detects that it has a valid IP address.

```
reflex transportUp() {  
    // ...  
}
```

## Link Up

Link up is called when the module detects that an active BrainStem link has been established.

```
reflex linkUp() {  
    // ...  
}
```

## Link Down

Link down is called when the module detects that an active BrainStem link has been disconnected.

```
reflex linkDown() {  
    // ...  
}
```

## Transport Down

Transport Down is called when the Module detects that its primary transport to the host has been disconnected.

```
reflex transportDown() {  
    // ...  
}
```

## Map Disable

Map Disable is called when the map file is disabled. The reflex machine within the module first executes this reflex origin, and then removes all allocated reflex origins for the map.

Example:

```
reflex mapDisable() {
    // ...
}
```

## 3.7.6 Keywords in the Reflex Language

The Reflex language has a couple of keywords with special meaning. The word reflex is used to declare reflexes. The pad keyword is used in the declaration of pad variables. Other than these two keywords, many of the language keywords share semantics with the same keyword in the ANSI C language.

Keywords		
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' ;
```

## 3.7.7 Operators and Precedence

The Reflex language shares many of its operators with other C-like languages. The following table presents the different operators, and organizes them by precedence. Operators higher in the table take precedence over operators at lower rows in the table.

Level	Operator	Description	Associativity
1	++ -- () [] .	Pre Increment Pre Decrement Function call Index Subscript command selection	Left-to-right

continues on next page

Table 5 – continued from previous page

Level	Operator	Description	Associativity
2	<code>++</code> <code>--</code> <code>+</code> <code>-</code> <code>!</code> <code>~</code> <code>( T )</code>	Post Increment Post Decrement Unary plus Unary minus Logical NOT Bitwise NOT Type cast	Right-to-left
3	<code>*</code> <code>/</code> <code>%</code>	Multiplication Divide Modulo	Left-to-right
4	<code>+</code> <code>-</code>	Addition Subtraction	Left-to-right
5	<code>&lt;&lt;</code> <code>&gt;&gt;</code>	Bitwise Left Shift Bitwise Right Shift	Left-to-right
6	<code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>&gt;=</code>	Less than Less than or equal Greater than Greater than or equal	Left-to-right
7	<code>==</code> <code>!=</code>	Equal to NOT Equal to	Left-to-right
8	<code>&amp;</code>	Bitwise AND	Left-to-right
9	<code>^</code>	Bitwise XOR	Left-to-right
10	<code> </code>	Bitwise OR	Left-to-right
11	<code>&amp;&amp;</code>	Logical AND	Left-to-right
12	<code>  </code>	Logical OR	Left-to-right

continues on next page

Table 5 – continued from previous page

Level	Operator	Description	Associativity
13	? :	Ternary Operator	Right-to-left
14	= += -= *= /= %= <<= >>=	Assignment 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

### 3.7.8 Types, Identifiers and Numbers

Reflex limits types to integer types and booleans. There is no floating point type included in the language, and new types cannot be declared. There are a couple of cases where certain constructs behave like types. For instance module definitions and declarations behave in some ways like types, but are not treated as such in the grammar or in the language definition. Identifiers follow the familiar C-like rules, Identifiers must start with a character which is not a digit, terminal character or '+' and identifiers cannot shadow keywords.

#### Legal Identifiers

```
__hello
value
v1234
$value
@value
aHappyVariable
a$Happy@Variable
another_happy_variable
```

#### Illegal Character

```
1badvar
+mybadvar
; anotherbadvar
```

## Integer Literals

Negative integer literals can be written by prefixing the integer with a - sign. Signed values are represented by two's complement values as in other C-like languages.

Integer literals can also be represented as hexidecimal values to write a hex value, prefix the literal with `0x`. So 127 can be represented as `0x7F` and -128 can be represented as `-0xF0`

## Signed vs Unsigned

A given type is considered a signed value unless it is prefixed with the `unsigned` keyword. The language does include the `signed` keyword and a value may be fully and explicitly declared as such, but in general use of `signed` is not necessary.

Type	Size in Bytes	Min	Max	unsigned Max
char (byte)	1	-128	127	255
short	2	-32768	32767	65535
int	4	-2147483648	2147483647	4294967295

## Identifiers, and Type Declarations EBNF

### Identifier

```

letter      ::=  'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J'
               | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T'
               | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z' | 'a' | 'b' | 'c' | 'd'
               | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n'
               | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x'
               | 'y' | 'z' ;
nonterminal ::=  '_' | '$' | '@' ;
ID          ::=  letter | nonterminal , { letter | digit | 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' ;
typeSpecifier ::=  TYPE | 'unsigned' , TYPE | 'signed' , TYPE ;
```

### 3.7.9 The Reflex Preprocessor

Reflex source files are preprocessed prior to compilation. There are a few useful preprocessor directives which are available within a reflex source file. These directives are borrowed from the C language;

Preprocessor keywords
define
ifdef/endif
include

## Examples

```
#define VAL 1234
#define DEBUG

#ifndef DEBUG
...
#endif

#include "myroutines.reflex"
#include <a40PinModule.reflex>
```

### 3.7.10 Variable Declaration

Reflex supports two types of global variable, and it supports a single scope of variables within reflex and routine declarations. Because reflexes are limited to a single variable scope. A declared variable is visible anywhere within the routine. 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_declaraction ::= 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_declaraction ::= 'pad' , "[" , INT , ":" , INT , "]" ,  
typeSpecifier , 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 to developers familiar with C-like languages. Variables may be initialized when they are declared but initialization at declaration is not required.

```
char value;
// or
char value = 12;
```

## Routine Variable Declaration EBNF

```
variable_declaraction ::= type_specifier , ID , [ "=" , INT ] , ";" ;
```

### 3.7.11 Statements

Statements in Reflex include control structures, assignment statements, and routine and reflex execution statements. A reflex or routine is made of a series of statements. In addition, a *variable\_declaration* is a special type of statement which must precede other types of statements within a statement list.

#### Control Statements

The largest class of statements are the control statements. These include branching statements like If-else and switch statements and looping control statements with include for and while loops. The reflex language syntax for control statements follows the familiar C-Like pattern. However the lack of sub scoping and the fact that variables are defined before other statements means that care must be given when introducing control statements 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 statements include:

Control Statements	
If/Else	Branching
Switch	Branching
While	Looping
For	Looping
Do while	Looping

## If/Else

If/Else statements follow the C syntax. For example;

```
if ( a == b ) {  
    ...  
} else {  
    ...  
}
```

## Switch

Switch statements follow the C syntax. For example;

```
switch ( a ) {  
    case 1:  
        ...  
    break;  
  
    case 2:  
        ...  
    break;  
  
    default:  
        ...  
    break;  
}
```

## While

While statements follow the C syntax. For example;

```
while ( a != b ) {  
    ...  
}
```

## For

For statements follow the C syntax. However the iteration variable must be declared with The rest of the variable declarations at the beginning of the reflex or routine compound statement. For example;

```
reflex mapEnable() {  
    int i;  
    for ( i = k; i > 0; i-- ) {  
        ...  
    }
```

## Do While

Do while statements follow the C syntax. For example;

```
do {
    ...
} while (a != b);
```

### 3.7.12 Reflex and Routine Definition

There are two types of top level definitions in the reflex language, reflex definitions and routine definitions. A reflex file consists of 1 or more reflex definitions, and zero or more routine definitions. The major syntactic difference between a routine definition and a reflex definition is that a reflex definition must be preceded by the keyword `reflex`, the fact that reflex definitions often include an entity index specification [ `index` ] as part of the reflex name declaration and reflexes do not return values.

Semantically the two types are very different. Routines are very similar to functions in other C-like languages. They exist to perform a function and are called within other code blocks. Reflexes on the other hand are tied to events which occur in the BrainStem module, such as an analog reading, timer expiration, or in the case of map enable and disable, the enabling of the compiled reflex file itself. In the reflex nomenclature we say that a reflex definition is tied to a reflex ‘origin’, multiple reflexes can be attached to a reflex origin through the enabling of multiple reflex files, and each reflex definition that is ‘mapped’ to an origin will execute when the condition triggering the origin occurs. In this way reflexes form a reactive system of behaviors on each BrainStem module.

---

**Note:** Reflexes cannot return values themselves. These are behaviors performed as the result of some system event, and as such have no place, like `main`, to return a value to. If a reflex needs to retain state at the end of its execution, it should place such state information into the Scratchpad.

---

#### Reflex Definition

```
reflex system.timer[0].expiration(void)
{
    ...
}
```

#### Routine Definition Example

```
short linearizeIRData(short data) {
    ...
    return linearizedData;
}
```

Routine definitions are syntactically much like function definitions in C. They declare a return type, and take zero or more parameters.

## Reflex Definition EBNF

```
reflex_definition ::= 'reflex' , entity_specifier , parameter_list ,
                    compound_statement ;
parameter_list ::= '(' [ parameter , {',' parameter} ] ')' ;
parameter      ::= typeSpecifier , ID ;
compound_statement ::= '{' statement_list '}' ;
statement_list ::= { variable_declaration } , { statement } ;
statement       ::= routine_call | assignment_statement | control_statement ;
```

## Routine Definition EBNF

```
routine_definition ::= typeSpecifier , ID , parameter_list ,
                     compound_statement ;
```

## Entity Specifier EBNF

The entity specifier fully describes a brainstem UEI. See the [UEI appendix](#) for more information. UEI's consist of an entity class, the entity index, and an option/command.

```
entity_specifier ::= entity_class , [ "[" , index , "]" ] , "." , class_option ;
index            ::= '0' | digit-not-zero , digit ;
entity_class     ::= "Module specific list of possible entities"
class_option     ::= "Entity specific list of possible options for the entity"
```

## 3.7.13 Appendix

## Appendix I: Example Reflex Module Definition file.

A reflex module definition file is used by the arc compiler to determine which entities and capabilities the BrainStem device supports. These files exist in the alinclude folder, and are included in a Reflex file that will use the module.

### Example Include Directive

```
#include <a40PinModule.reflex>
```

### Reflex Module Definition syntax

Module definition files are similar to C style headers, and support preprocessor directives like `#include` and `#define`.

The module definition begins with some includes for constants, and a module declaration `module a40PinModule`. Each line of the declaration block defines an individual entity or capability.

```
<cmdType> [Index] { <cmdOption>, type, <ueiRequestTypes (GET|SET)> }
```

```
#ifndef __a40PinModule_reflex__
#define __a40PinModule_reflex__

#include "aProtocoldefs.h"
#include "a40PinModuleDefs.h"

module a40PinModule
{
    cmdANALOG[0] { analogConfiguration, 0, ueiOPTION_GET }
    cmdANALOG[0] { analogVoltage, 0, ueiOPTION_GET }
    cmdANALOG[1] { analogConfiguration, 0, ueiOPTION_GET }
    cmdANALOG[1] { analogVoltage, 0, ueiOPTION_GET }
    cmdANALOG[2] { analogConfiguration, 0, ueiOPTION_GET }
    cmdANALOG[2] { analogVoltage, 0, ueiOPTION_GET }
    cmdANALOG[3] { analogConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
    cmdANALOG[3] { analogVoltage, 0, ueiOPTION_SET | ueiOPTION_GET }

    cmdAPP[0] { appExecute, 0, ueiOPTION_SET }
    cmdAPP[0] { appReturn, 0, ueiOPTION_SET }
    cmdAPP[1] { appExecute, 0, ueiOPTION_SET }
    cmdAPP[1] { appReturn, 0, ueiOPTION_SET }
    cmdAPP[2] { appExecute, 0, ueiOPTION_SET }
    cmdAPP[2] { appReturn, 0, ueiOPTION_SET }
    cmdAPP[3] { appExecute, 0, ueiOPTION_SET }
    cmdAPP[3] { appReturn, 0, ueiOPTION_SET }

    cmdCLOCK[0] { clockYear, 0, ueiOPTION_SET | ueiOPTION_GET }
    cmdCLOCK[0] { clockMonth, 0, ueiOPTION_SET | ueiOPTION_GET }
    cmdCLOCK[0] { clockDay, 0, ueiOPTION_SET | ueiOPTION_GET }
    cmdCLOCK[0] { clockHour, 0, ueiOPTION_SET | ueiOPTION_GET }
    cmdCLOCK[0] { clockMinute, 0, ueiOPTION_SET | ueiOPTION_GET }
    cmdCLOCK[0] { clockSecond, 0, ueiOPTION_SET | ueiOPTION_GET }

    cmdDIGITAL[0] { digitalConfiguration, 0, ueiOPTION_SET | ueiOPTION_GET }
}
```

(continues on next page)

(continued from previous page)

```

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 }
cmdPOINTER[2] { pointerTransferFromStore, 0, ueiOPTION_SET }

```

(continues on next page)

(continued from previous page)

```

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 }
cmdTIMER[5] { timerMode,                0, ueiOPTION_SET | ueiOPTION_GET }

```

(continues on next page)

(continued from previous page)

```
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 }
}

#ifndef __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
0x0006 pushls    0x0000      0x3C
                  0x04
                  0x00
                  0x00
0x0009 pushpc
0x000A popec    0x038460      0x39
                  0x74
                  0x03
                  0x84
                  0x60
0x000E popn     0x01      0x02
```

(continues on next page)

(continued from previous page)

0x0010	pushlc	0x01	0x01 0x03 0x01
0x0012	popec	0x4F8260	0x74 0x4F 0x82 0x60
0x0016	popn	0x01	0x02 0x01
0x0018	pushli	0x0007A120	0x05 0x20 0xA1 0x07 0x00
0x001D	popei	0x4F8160	0x76 0x4F 0x81 0x60
0x0021	popn	0x01	0x02 0x01
0x0023	exit		0x01
0x0024	pushls	0x0000	0x04 0x00 0x00
0x0027	pushpc		0x39
0x0028	pushlc	0x01	0x03 0x01
0x002A	eqc		0x54
0x002B	popn	0x01	0x02 0x01
0x002D	brz	0x0039	0x12 0x39 0x00
0x0030	pushlc	0x00	0x03 0x00
0x0032	pushls	0x0000	0x04 0x00 0x00
0x0035	poppc		0x3C
0x0036	br	0x003F	0x11 0x3F 0x00
0x0039	pushlc	0x01	0x03 0x01
0x003B	pushls	0x0000	0x04 0x00 0x00
0x003E	poppc		0x3C
0x003F	pushls	0x0000	0x04 0x00 0x00
0x0042	pushpc		0x39
0x0043	popec	0x038460	0x74 0x03 0x84 0x60

(continues on next page)

(continued from previous page)

0x0047	popn	0x01	0x02 0x01
0x0049	exit		0x01
0x004A	pushlc	0x00	0x03 0x00
0x004C	convci		0x1C
0x004D	popei	0x4F8160	0x76 0x4F 0x81 0x60
0x0051	popn	0x01	0x02 0x01
0x0053	pushlc	0x00	0x03 0x00
0x0055	popec	0x4F8260	0x74 0x4F 0x82 0x60
0x0059	popn	0x01	0x02 0x01
0x005B	pushlc	0x00	0x03 0x00
0x005D	popec	0x038460	0x74 0x03 0x84 0x60
0x0061	popn	0x01	0x02 0x01
0x0063	exit		0x01

## 4.1 BrainStem Platform

Acroname's BrainStem technology is a foundational tool for bridging the gap between hardware and software systems.

### 4.1.1 What is BrainStem?

What makes the BrainStem platform unique and exciting? Get an [overview](#).

### 4.1.2 Explore your module's capabilities With HubTool.

New Hardware? See what is included in the box, and how to quickly get connected to your new hardware. Find out about links and your hardware's I/O capabilities.

### 4.1.3 Are you up-to-date?

Is your hardware running the latest firmware? Learn how to take advantage of new features or make sure you don't get bitten by any nasty bugs. Also learn how to easily update your modules using aUpdater.

### 4.1.4 Hello World Reflex

Get started writing Reflex code for your BrainStem module. This tutorial will introduce you to the basics of the Reflex language and how to load and enable your reflex with aStemStoreTool.

## 4.1.5 Hello World C++

Now that you've seen the world of Reflex, Say hello to the C++ API. This tutorial will introduce you to the basics of writing a C++ application that communicates with your module to say hello.

## 4.1.6 Next Steps

### BrainStem Terminology

Read up about why we call it what we call it.

### Reflex Language Reference

Explore the reflex programming language, and the advantages of enabling basic behaviors on your BrainStem devices.

### C++ API Reference

Create rich host based applications using the C++ BrainStem API.

### Python API Reference

Create rich host based applications using the Python BrainStem API.

### C API Reference

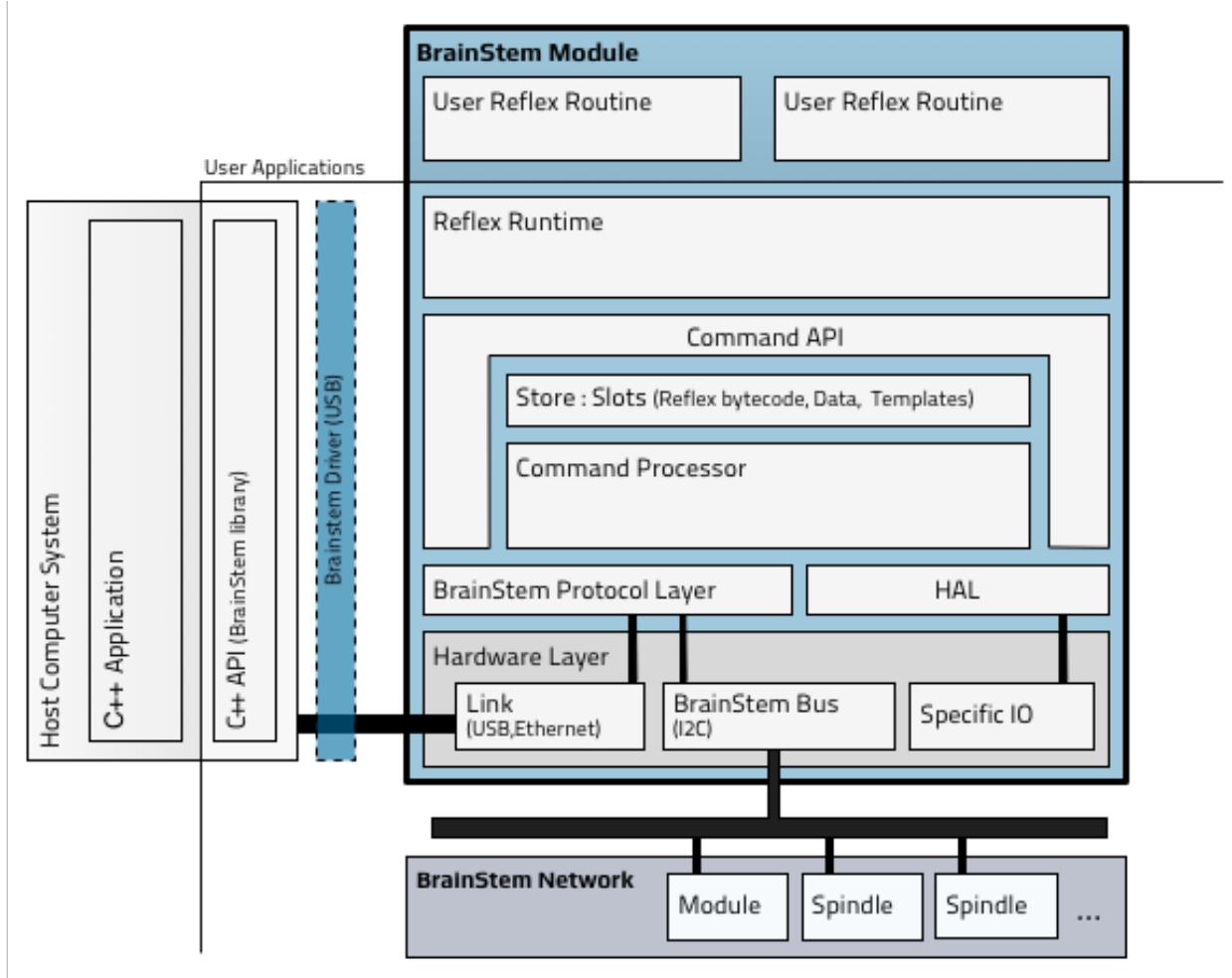
Dig into the lower level C API for more control, or to extend BrainStem to your favorite language.

## 4.2 What is BrainStem

Acroname's BrainStem technology is a foundational tool for bridging the gap between hardware and software systems. BrainStem controllers gather and relay peripheral information to other systems, which may be a host computer, additional microcontrollers or an array of sensors and testing equipment. The core BrainStem concept is metaphorically based on the human nervous system's data gathering, interpreting and transmitting processes.

### 4.2.1 Embedded With Reflex

A high-level C-like programming language called [Reflex](#), and the associated compiler, enable code to execute directly on-board the controller modules. The BrainStem Reflex system allows access to a soft real-time operating system. This type of system is ideal for those who have outgrown the "super-loop" type systems like Arduino.



However, the elegant simplicity and C-like familiarity of the Reflex language makes low level microcontroller programming approachable; no more digging through thousands of pages of datasheets or wiring up expensive JTAG debuggers. The Reflex language allows for simple code to execute complex functions when triggered by external events, without getting bogged down in low level interrupt calls and microcontroller specific limitations. By utilizing BrainStem reflexes, a system can be remotely deployed and react intelligently to its environment. The embedded reflex program can collect and store information for later retrieval on the BrainStem's internal persistent memory or an external micro-SD card.

## 4.2.2 Scalable

BrainStem modules are the ideal controllers for hierarchical control and autonomous systems. Using a robust networking protocol, BrainStem devices relay information across industry standardized interfaces such as serial, I2C, USB, Ethernet and BlueTooth. Each controller has an embedded virtual machine kernel that allows for rich embedded application execution, reflexive software creation and direct hardware access using a structured packet format. The BrainStem network can support more than 100 microcontrollers, and each microcontroller can have several subordinate networks of devices, sensors or interfaces.

### 4.2.3 Usable

The BrainStem platform and associated APIs provide powerful desktop computer access to sensors and actuators. Since these devices exist across such a wide application range, BrainStem modules are designed to be as generalized and adaptable as possible. This includes cross platform interface software, defining and publishing interconnect standards and protocols, selecting a common form factor and focusing on expandability.

### 4.2.4 Next Steps

The best place to get started with your new BrainStem hardware is by checking out the [Getting Started](#) section of our documentation.

## 4.3 Getting Started

Before you begin you'll need to collect the following items.

- A BrainStem Module with development board or an Acroname Hub/Switch.
- A Link Transport Cable (Ethernet or USB).
- The BrainStem Support software package.

BrainStem Devices and Development boards can be purchased from the [Acroname Products](#).

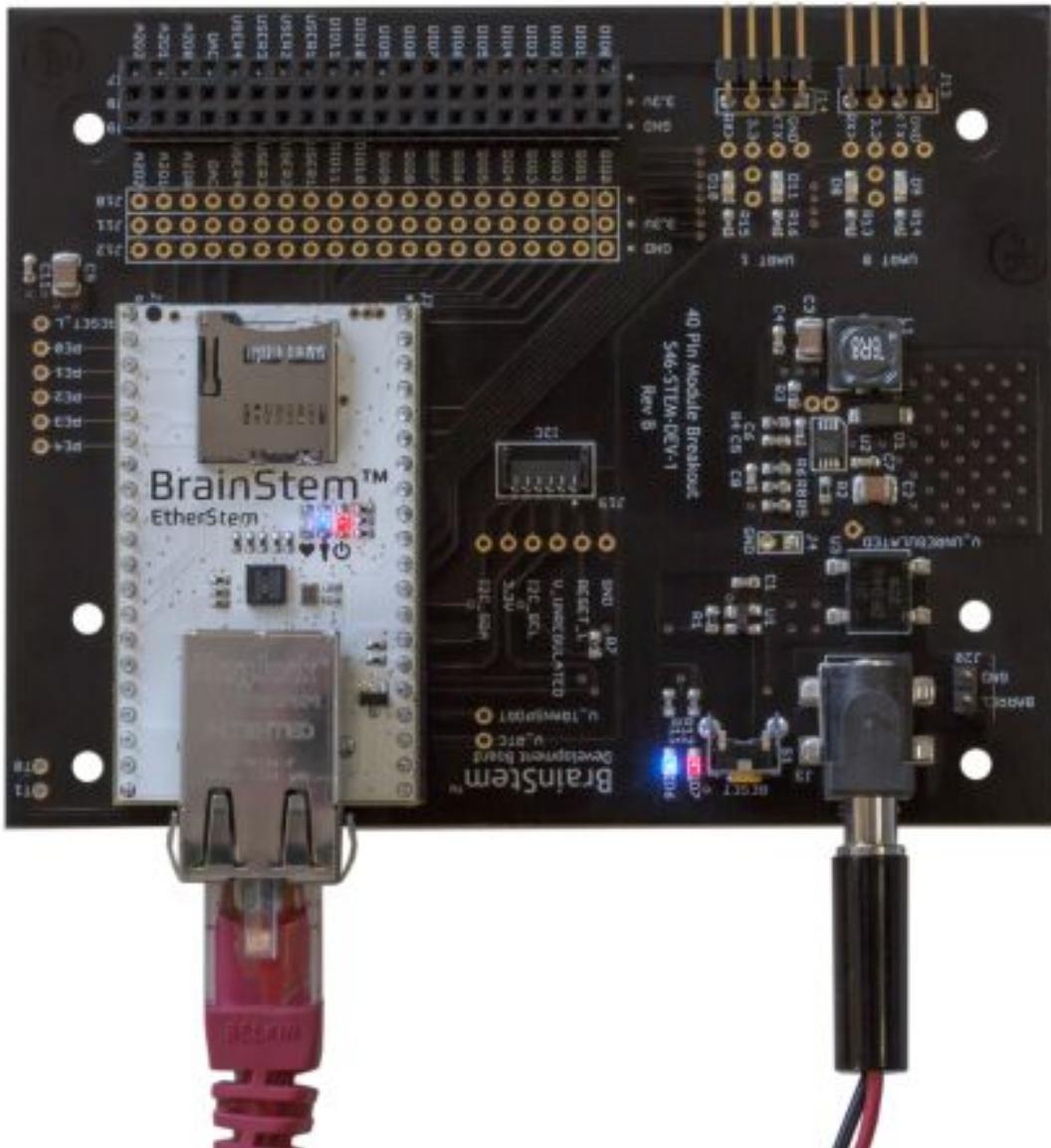
The latest version of the [BrainStem Development Kit or HubTool](#) can be downloaded from the Acroname Download Page. Extract the package to the location of your choice when the download has finished.

### 4.3.1 Do I need Drivers?

Acroname kernel drivers are no longer required for most modern operating systems; However, if you are running Windows 7 or Linux, please read the [BrainStem USB installation instructions](#) for your particular operating system.

### 4.3.2 Connecting to a BrainStem Device

With the exception of our Hubs/Switches most BrainStem devices must be plugged into the BrainStem Development Board to receive power. Once the device has been plugged into a development board it is safe to insert the link transport cable. The red power LED should be on and the green heartbeat LED should flicker rapidly when power is first applied. If you are using a Hub/Switch connect the supplied power adapter to the device.



### 4.3.3 Launch HubTool

For BrainStem devices and Hubs navigate to the bin folder of the BrainStem Development Kit to find HubTool.

### 4.3.4 Toggling the LED

HubTool will periodically search for connected devices and display them in the lower right corner. Selecting your device will cause a UI to be created for that device based on its capabilities. Click the LED button and observe the illumination of the user LED on the device.

#### Updating your module firmware with Updater.

We are constantly fixing bugs and improving our products. It's good practice to keep your modules up-to-date with the latest firmware. Please see the [BrainStem Firmware Management](#) to update your firmware.

#### Write and execute your first reflex.

Interested in Reflexes? See the [Reflex](#) section of this reference to get started.

#### Introduction to the C++ API.

Interested in communicating with your BrainStem module from a host computer via the [C++ API](#)? See the [Getting Started C++ guide](#)

#### Introduction to the Python API.

Want to work with Python? See the [Python API](#) section of this reference to get started.

## 4.4 Firmware Management

### 4.4.1 Firmware Upgrade Precautions

Reliable power should be used when updating firmware and the update process should be allowed to complete without interruption. An interrupted firmware upgrade process will leave the BrainStem unusable and will require using the firmware recovery process.

### 4.4.2 Brainstem Firmware

Each BrainStem module uses an Acroname developed firmware which contains a boot loader, the BrainStem OS, the reflex virtual machine, access to the BrainStem protocol, and access to functionality specific to each module. Each module's firmware is unique to the module type and the specific module serial number.

The firmware is continuously maintained for bug fixes and feature enhancements. Updates to the firmware can be downloaded from Acroname using the tools detailed below. For large volume customers, offline firmware files can be made available under license.

### 4.4.3 Firmware Update Tools

BrainStem modules can be updated via their link interface (e.g. USB, EtherNet, etc) by using the command line interface (CLI) “Updater” tool. The Updater tool is packaged in each BrainStem development and HubTool package. The extensive built-in help can be accessed with the “-H” option. Example update processes are shown in the following sections.

For legacy updaters such as: “aUpdater” and “UpdateTool” please see [Legacy Firmware Management](#).

### 4.4.4 Using Updater via CLI

BrainStem firmware can be modified using the Updater CLI utility. This method is often very useful in remote installations and large scale automation systems as it is easily invoked with a simple script. Using your computer’s operating systems terminal interface, navigate to the “Bin” folder located in the Brainstem Dev Kit download. Downloads for all of Acroname’s products can be found at [Download Center](#) under the Support tab.

Parameters and commands available for the Updater may be found by running the utility without any commands or arguments (When running Updater on a Mac or Linux you will need to use “./”. Windows does not require this).

Note: The following examples will be preformed from a Mac.

```
$> ./Updater
```

If you would like additional help information, including examples you can use the -H command.

```
$> ./Updater -H
```

```
Application:Updater
  Version:Version: 1.0 Dec 31 2015 11:00:34
  BrainStem 2.1.5
  Copyright (C) 1994-2015, Acroname Inc.

Updater [options]
  BrainStem Update and Maintenance Application

Parameters to commands:
  -b <build number>          - Build version of the firmware to transfer.
                                [Default: latest version available].
  -d <device>                - Device serial number of the device to access.
  -t <transport type>        - communications method [USB,TCP,RECOVER].
  -f <firmware file>          - Firmware file to be transferred to device.
  -r <router serial#>        - indirect connection through router device
  -s <serial port>            - name of serial port to use for recovery

  -l 'log message'           - message to place in device history log
  -a <mode>                  - set log file access mode [A, X].
                                Override the default append mode [A] with
                                overwrite mode [X].
  -k <filename>              - log messages written into given filename
                                Default:[APP.log]

Commands:
  -B                         - list the Builds available for given device.
```

(continues on next page)

(continued from previous page)

-D	- Discover devices that are currently connected.																																
-G	- Get build file from the Acroname server and store on local machine for download to device. Must specify a specific device ["-d" option]. Default is to obtain the latest build for the given device, override with "-b" option.																																
-H or -h	- Display extended usage information.																																
-I	- View stored device information. Must either use DISCOVER command ["-D"] or specify a specific device ["-d" option].																																
-L	- Log the session output into a log file. Use the "-k" option to override the default filename [APP.log].																																
-U	- Update the firmware on device specified. Must specify a specific device ["-d" option]. Default is to download the latest build for the given device, override with "-b" option or specify specific BIRD file using "-f" option.																																
-R	- Send RESET to device Must specify a specific device ["-d" option].																																
-V	- View stored history for device. Must either use DISCOVER command ["-D"] or specify a specific device ["-d" option].																																
-X <BIT_MASK>	<ul style="list-style-type: none"> <li>- Bitmask for Logging           <table> <tbody> <tr><td>DEBUG_Updater</td><td>0x00000001</td></tr> <tr><td>DEBUG_History</td><td>0x00000002</td></tr> <tr><td>DEBUG_Attributes</td><td>0x00000004</td></tr> <tr><td>DEBUG_BirdFile</td><td>0x00000008</td></tr> <tr><td>DEBUG_Network</td><td>0x00000010</td></tr> <tr><td>DEBUG_ServerComm</td><td>0x00000020</td></tr> <tr><td> </td><td> </td></tr> <tr><td>DEBUG_Network</td><td>0x00000100</td></tr> <tr><td>DEBUG_Network_SR</td><td>0x00000200</td></tr> <tr><td>DEBUG_Network_FIRM</td><td>0x00000400</td></tr> <tr><td> </td><td> </td></tr> <tr><td>DEBUG_NXPSerial</td><td>0x00001000</td></tr> <tr><td>DEBUG_NXPSerial_E</td><td>0x00002000</td></tr> <tr><td>DEBUG_NXPSerial_SR</td><td>0x00004000</td></tr> <tr><td> </td><td> </td></tr> <tr><td>DEBUG_APP</td><td>0x00010000</td></tr> </tbody> </table> </li> </ul>	DEBUG_Updater	0x00000001	DEBUG_History	0x00000002	DEBUG_Attributes	0x00000004	DEBUG_BirdFile	0x00000008	DEBUG_Network	0x00000010	DEBUG_ServerComm	0x00000020			DEBUG_Network	0x00000100	DEBUG_Network_SR	0x00000200	DEBUG_Network_FIRM	0x00000400			DEBUG_NXPSerial	0x00001000	DEBUG_NXPSerial_E	0x00002000	DEBUG_NXPSerial_SR	0x00004000			DEBUG_APP	0x00010000
DEBUG_Updater	0x00000001																																
DEBUG_History	0x00000002																																
DEBUG_Attributes	0x00000004																																
DEBUG_BirdFile	0x00000008																																
DEBUG_Network	0x00000010																																
DEBUG_ServerComm	0x00000020																																
DEBUG_Network	0x00000100																																
DEBUG_Network_SR	0x00000200																																
DEBUG_Network_FIRM	0x00000400																																
DEBUG_NXPSerial	0x00001000																																
DEBUG_NXPSerial_E	0x00002000																																
DEBUG_NXPSerial_SR	0x00004000																																
DEBUG_APP	0x00010000																																
-Y	- change router address of an network device to given router. Requires: -d <mod:device> and -r <device> options.																																
Sample usage:																																	
-D	Discover all devices connected by either USB or TCPIP.																																

(continues on next page)

(continued from previous page)

	Records the device information into the settings for that device.
-D -t USB	Discover all the devices connected by USB. Records the device information into the settings for that device.
-D -t USB -d 0x40F5849A	Discover settings of the specific device (serial number 0x40F5849A)
-G -d 0x40F5849A	Get the latest firmware for the given device. Downloads the firmware file into the updater specific device directory.
-U -d 0x40F5849A	Update the firmware on the specific device. Without other options, this command will try to locate the latest firmware version to be used for updating the device.
-G -d 0x40F5849A -b 99528558	Get the specified build's firmware for the given device. Downloads the firmware file into the updater's specific device directory with the build number as the filename.
-G -U -d 0x40F5849A	Get the latest firmware for the given device. Downloads the firmware file into the updater specific device directory. After download, update the firmware using the latest release.
-U -d 0x40F5849A -X 0x0010	Update the firmware as described above, but turns on the display of networking messages used in the transfer.
-U -t RECOVER -s /dev/serial	Update the firmware as described above, but uses the serial interface in communicating with the device to install the latest firmware.
Sample usage for network discovery and updates:	
-D -r 0xD272031D	Discover all devices connected to the I2C BrainStem network of the given routing device 0xD272031D.
-G -U -d 0x2181F0EE -r 0xD272031D	Get and Update to the latest firmware for device 0x2181F0EE indirectly through routing device 0xD272031D.
Updater [Version: 1.0 Dec 31 2015 11:00:34] [BrainStem Release:2.1.5] [Copyright (C) 1994-2015, Acroname Inc.]	
NO COMMANDS GIVEN	
Completed processing: Updater [Version: 1.0 Dec 31 2015 11:00:34] [BrainStem Release:2.1.5] [Copyright (C) 1994-2015, Acroname Inc.]	

As you can see there are a lot of options for customizing the Updater utility to meet your needs; however, there

are just a few basic command that will fit the needs of most users.

Next we will look at a few examples of how to use the updater.

#### 4.4.5 Example: Updating to the Latest Firmware

In this example we will go through the steps required to update our BrainStem module. We will be using a 40pin USBStem module throughout this exercise; however, the other modules work in a similar way.

Make sure your device is connected and has power. Refer to the *Getting Started* page for additional information.

##### Checking for connected devices:

This command will check for both USB and TCP/IP devices connected to your machine and network.

```
$> ./Updater -D
```

Looking at the output below you can see that two devices were discovered. One USBStem and one EtherStem. In this example we will be using the USBStem information. This information will be handy in the next step as we will need the serial number of our device in order to update it.

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-D]
Updater [Version 0.2 Nov 24 2015 09:34:27] [Copyright (C) 1994-2015, Acroname Inc.]

Discovering Devices [USB]:
    Device   Module   Router   Model           Firmware Version
    3797E6F8  02       02       04 [USBStem     ]  2.1.5

Discovering Devices [TCPIP]:
    Device   Module   Router   Model           Firmware Version
    856C1C03  06       06       05 [EtherStem   ]  2.1.1 [10.128.38.159]

Completed processing: Updater [Version 0.2 Nov 24 2015 09:34:27] [Copyright (C)
1994-2015, Acroname Inc.]
```

##### Getting the latest firmware from Acroname's servers:

Using the serial number for the USBStem above we will construct the following command. The “-G” will pull the most recent firmware from Acroname’s server for the given device serial number (“-d”).

```
$> ./Updater -G -d 0x3797E6F8
```

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-G]
    [-d]
    [0x3797E6F8]
Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]
```

(continues on next page)

(continued from previous page)

```
Current build for [3797E6F8][aUSBStem] ==> [130702287].
Get build for [3797E6F8][aUSBStem][130702287] ==> 72784 bytes [YmlyZAEBeJzsunk8lN/b].
GetBuild BIRD [/Users/Mitch/.acroname/updater/3797E6F8/130702287.bird] VERIFIED.
Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994–2015, Acroname Inc.]
```

### Loading the latest firmware:

Now that we have successfully pulled the most up to date firmware we now need to apply it to the device. The following code will apply the most up to date firmware to the given device.

```
$> ./Updater -U -d 0x3797E6F8
```

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-U]
    [-d]
    [0x3797E6F8]
Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994–2015, Acroname Inc.]
Update using firmware file [/Users/Mitch/.acroname/updater/3797E6F8/130702287.bird].
Transferring loader to device
Transferring loader block 0, 8852 bytes
Transferred 1 blocks, 8852 total bytes
Transferring firmware to device [/Users/Mitch/.acroname/updater/3797E6F8/130702287.
    ↵bird]
Transferring firmware block 0, 8852 bytes
Transferring firmware block 1, 772 bytes
Transferring firmware block 2, 49668 bytes
Transferred 3 blocks, 59292 total bytes
Completed updating firmware on device [3797E6F8]
Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994–2015, Acroname Inc.]
```

At this point the firmware has been downloaded to the device and the device has reset and is running with the new firmware.

### 4.4.6 Example: Reverting to a Previous Version

In our next example we are going to assume that we have just updated to the 2.1.5 firmware; however, for some reason we are not satisfied with how the device is behaving and we want to return back to 2.1.4. With the Updater utility this is easy to do.

Before we begin 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 6FCBBAA4
```

Below you will see what was printed on my machine<sup>20</sup>. All we need to do is take note of the specific build number that is associated with the version we would like to revert to.

```
2017:05:10:18:10:05 | Updater [Version: 1.1 May 1 2017 13:59:12]  
[BrainStem Release:2.3.12] [Copyright (C) 1994–2016, Acroname Inc.]
```

```
2017:05:10:18:10:05 |
```

```
Build List for device[6FCBBAA4]:  
2017:05:10:18:10:06 | Build Version  
2017:05:10:18:10:06 | 219607159 2.2.0  
2017:05:10:18:10:06 | 111486747 2.2.1  
2017:05:10:18:10:06 | 231932966 2.2.2  
2017:05:10:18:10:06 | 156729307 2.2.3  
2017:05:10:18:10:06 | 25628300 2.2.4  
2017:05:10:18:10:06 | 250406850 2.2.5  
2017:05:10:18:10:06 | 226315939 2.2.6  
2017:05:10:18:10:06 | 33051391 2.2.7  
2017:05:10:18:10:06 | 130872794 2.2.8  
2017:05:10:18:10:06 | 131823882 2.3.0  
2017:05:10:18:10:06 | 119187462 2.3.1  
2017:05:10:18:10:06 | 9473450 2.3.10  
2017:05:10:18:10:06 | 4719070 2.3.11  
2017:05:10:18:10:06 | 5329028 2.3.12  
2017:05:10:18:10:06 | 130782904 2.3.2  
2017:05:10:18:10:06 | 70185126 2.3.3  
2017:05:10:18:10:06 | 72872664 2.3.4  
2017:05:10:18:10:06 | 8157420 2.3.5  
2017:05:10:18:10:06 | 10682084 2.3.6  
2017:05:10:18:10:06 | 10941901 2.3.7  
2017:05:10:18:10:06 | 3196158 2.3.9
```

```
2017:05:10:18:10:06 | Completed processing: Updater [Version: 1.1 May 1 2017  
→13:59:12]  
[BrainStem Release:2.3.12] [Copyright (C) 1994–2016, Acroname Inc.]
```

<sup>20</sup> This output was spliced in to reflect the changes in Updaters output. Therefor, the build numbers will not align with the rest of the example.

## Applying a specific build to a device:

As previously explained we are hypothetically having issues with our device after updating and we want to revert to a previous version. Now that we have located the previous build number lets apply it by adding the “-b” parameter to the last command in the example before.

```
$> ./Updater -G -U -d 0x3797E6F8 -b 99528558
```

```
You should see something similar to the below output.

Home directory: [/Users/Mitch]

Application parameters: [Updater]
    [Updater]
    [-U]
    [-d]
    [0x3797E6F8]
    [-b]
    [99528558]

Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]
Update using firmware file [/Users/Mitch/.acroname/updater/3797E6F8/99528558.bird].
Transferring loader to device
Transferring loader block 0, 8296 bytes
Transferred 1 blocks, 8296 total bytes
Transferring firmware to device [/Users/Mitch/.acroname/updater/3797E6F8/99528558.
˓→bird]
Transferring firmware block 0, 8296 bytes
Transferring firmware block 1, 772 bytes
Transferring firmware block 2, 48776 bytes
Transferred 3 blocks, 57844 total bytes
Completed updating firmware on device [3797E6F8]
Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C)
1994-2015, Acroname Inc.]
```

**Confirm that you have successfully restored the old firmware:**

Just to be safe lets confirm that we have successfully restored the old firmware. This can be done by issuing the discover command.

```
$> ./Updater -D
```

As you can see the device is now running the 2.1.4 firmware.

```
Home directory:[/Users/Mitch]

Application parameters:[Updater]
    [Updater]
    [-D]
Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994-2015, Acroname Inc.]

Discovering Devices [USB]:
    Device      Module     Router   Model                  Firmware Version
        3797E6F8    02         02       04 [USBStem      ]    2.1.4

Discovering Devices [TCPIP]:
    Device      Module     Router   Model                  Firmware Version
```

(continues on next page)

(continued from previous page)

856C1C03 06 06 05 [EtherStem] 2.1.1 [10.128.38.159]

Completed processing: Updater [Version 0.2 Nov 25 2015 09:59:24] [Copyright (C) 1994–2015, Acroname Inc.]

#### 4.4.7 Example: Recovering a BrainStem Module

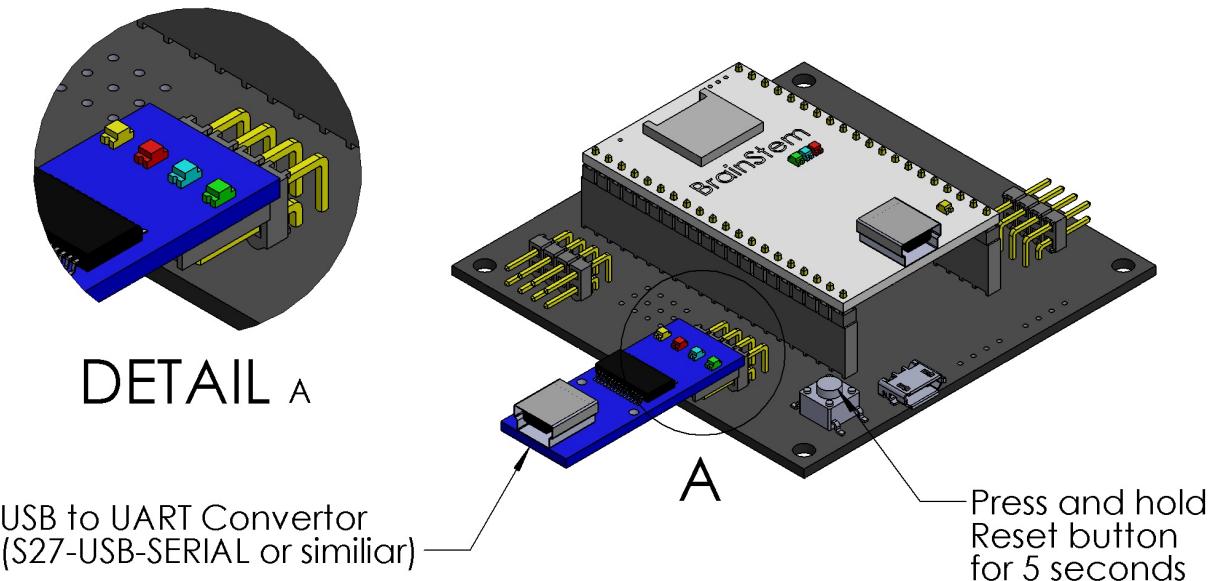
Lets take a look at how we can recover a BrainStem module after it has become unresponsive (aka: “Bricked”). This is also very helpful when dealing with old devices with a firmware version that might not be compatible with the new Updater utility.

In order to preform this recovery process you will need a [USB to Serial Module<sup>21</sup>](#). 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<sup>22</sup>](#).

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.



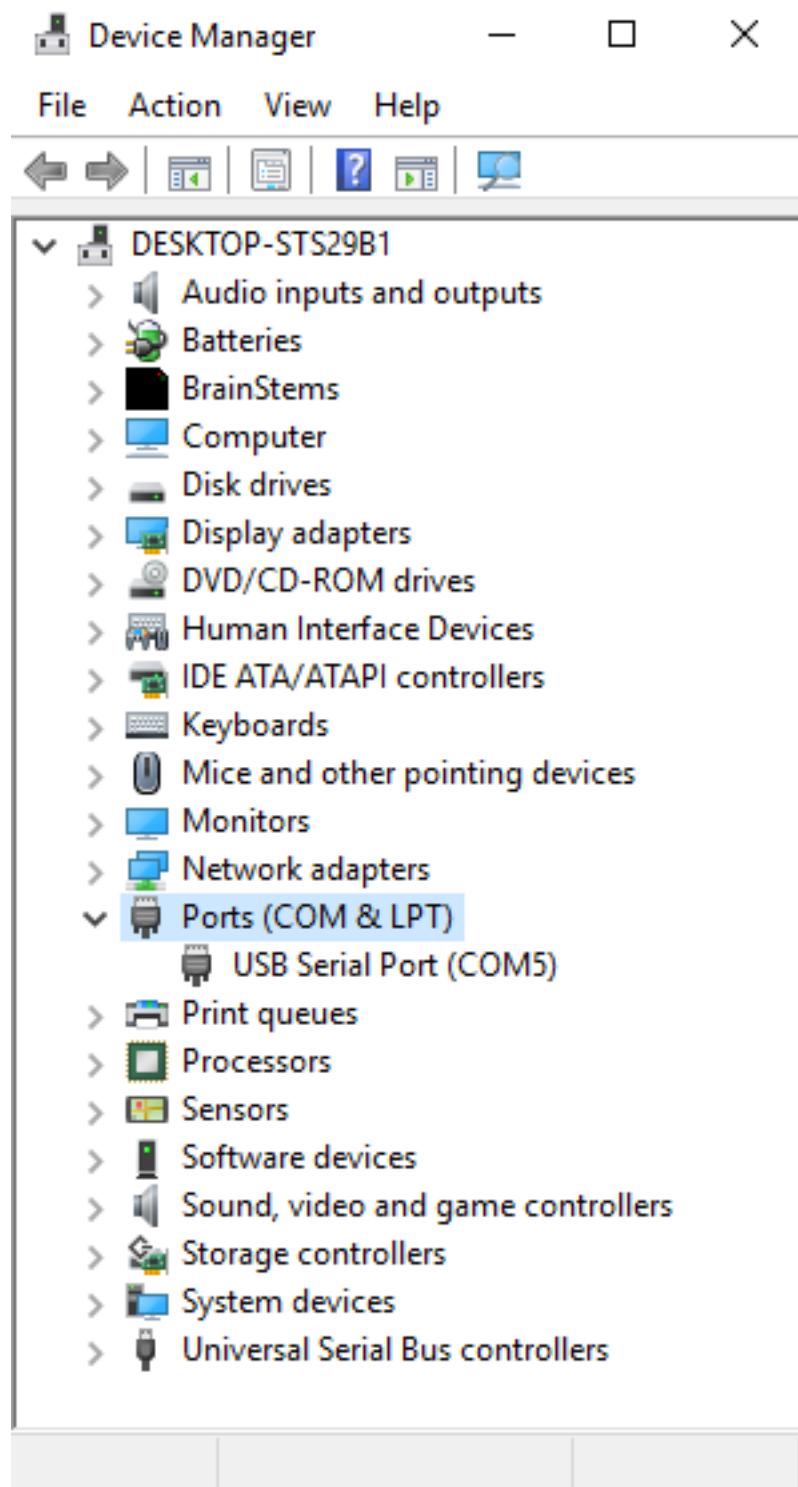
<sup>21</sup> <https://acroname.com/products/USB-SERIAL-CONVERTER?sku=S27-USB-SERIAL>

<sup>22</sup> <https://acroname.com/site-search/dev>

## **Finding your Communications Port**

### ***Windows***

The communications port (COM) on Windows can be found by navigating to the Device Manager and expanding the “Ports (COM & LPT)” section. If you do not immediately recognize your device you can open each and inspect the details or you can simply disconnect and reconnect the device and monitor which one disappears and then reappears (You may need to select Action > Scan for hardware changes between the disconnect and reconnect).



Once you have figured out which device is yours make note of the port number. In my case it would be "COM5"

## Mac/Linux

Open terminal and type in the following command (root access is required).

```
$> ls /dev/tty.*
```

This command will return all the serial devices connected to your machine. Locate the one you are wanting to work with. If you are not sure which serial device to use you can run the command twice. Once with the device connected and once without. The one that changes is the device you are interested in.

```
/dev/tty.Bluetooth-Incoming-Port
/dev/tty.Bluetooth-Modem
/dev/tty.usbserial-A601R8QJ
```

On my machine the device I am interested in is: “/dev/tty.usbserial-A601R8QJ”. Make note of your device as it will be needed later.

## Recovering your Device with Updater

Finally, we are ready to recover the device. Below you will see the command required to 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]
    [RECOVER]
    [-s]
    [/dev/tty.usbserial-A601R8QJ]
Updater [Version 0.2 Dec 9 2015 14:14:42] [Copyright (C) 1994-2015, Acroname Inc.]

Firmware_Recovery via Serial Interface:

Sync to Device:
```

(continues on next page)

(continued from previous page)

```
Get Info from Device:  
Device responded with version: 2.4  
Device responded with part number: 637615927  
Device ID: [ 05005006 AE061446 508B34C6 F5001E43 ]  
Device [05005006 AE061446 508B34C6 F5001E43] ==> Serial#:71E3928C, Build=130702287  
GetBuild BIRD [/Users/Mitch/.acroname/updater/71E3928C/130702287.bird] VERIFIED.  
  
Unlock Memory:  
Device ID: [ 05005006 AE061446 508B34C6 F5001E43 ]  
Transferring firmware to device from [/Users/Mitch/.acroname/updater/71E3928C/  
→130702287.bird]  
Transferring firmware block 0, 772 bytes  
Transferred 768 of 768 bytes (100%)  
Transferring firmware block 1, 49668 bytes  
Transferred 4096 of 49664 bytes (8%)  
Transferred 8192 of 49664 bytes (16%)  
Transferred 12288 of 49664 bytes (25%)  
Transferred 16384 of 49664 bytes (33%)  
Transferred 20480 of 49664 bytes (41%)  
Transferred 24576 of 49664 bytes (49%)  
Transferred 28672 of 49664 bytes (58%)  
Transferred 32768 of 49664 bytes (66%)  
Transferred 36864 of 49664 bytes (74%)  
Transferred 40960 of 49664 bytes (82%)  
Transferred 45056 of 49664 bytes (91%)  
Transferred 49152 of 49664 bytes (99%)  
Transferred 49664 of 49664 bytes (100%)  
Transferred 2 blocks, 50432 total bytes  
Rebooting Device  
  
END of Firmware_Recovery via Serial Interface  
  
Completed processing: Updater [Version 0.2 Dec 9 2015 14:14:42] [Copyright (C)  
1994-2015, Acroname Inc.]
```

#### 4.4.8 Example: Updating a Brainstem Module via the Brainstem Network.

The new updater utility also has the ability to update devices via the *BrainStem network*. This can be very handy when you have multiple BrainStem products connected to a single host computer.

## Transport vs Brainstem Network

Before digging in it is important to know the difference between a transport and the Brainstem network.

### **“Transport”**

When we refer to transport we are speaking of the interface between devices and more specifically we are referring to the hardware. Acroname currently offers three transports mediums; TCPIP, USB and I2C. While all are capable of trafficking the Brainstem network only TCPIP and USB are directly available to the user. Although I2C is 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](#)<sup>23</sup> where it will then be converted to the I2C transport and sent to a [MTM-PM1](#)<sup>24</sup> module.

Before we begin lets make sure your device is connected and has power. Refer to the [Getting Started](#) page for additional information.

## Discovery

Since we will be updating a device through another device we will need to know the serial number of the device we will be communicating through. To find the serial number we can simply use the ‘-D’ discover command.

```
$> ./Updater -D
```

```
Updater [Version 0.2 Dec 28 2015 13:54:37] [Copyright (C) 1994–2015, Acroname Inc.]
```

```
Discovering Devices [USB]:
```

Device	Module	Router	Model	Firmware Version
--------	--------	--------	-------	------------------

```
Discovering Devices [TCPIP]:
```

Device	Module	Router	Model	Firmware Version	[IP address]
D272031D	04	04	0F [MTMEtherStem]	2.2.0 (0)	[10.128.38.159]
856C1C03	02	02	05 [EtherStem ]	2.1.4 (99528558)	[10.128.38.122]

```
Completed processing: Updater [Version 0.2 Dec 28 2015 13:54:37] [Copyright (C) 1994–2015,  
Acroname Inc.]
```

<sup>23</sup> <https://acroname.com/products/MTM-ETHERSTEM-ETHERNET-MICROCONTROLLER-MODULE?sku=S67-MTM-ETHERSTEM>

<sup>24</sup> <https://acroname.com/products/ACRONAME-MTM-1-CHANNEL-POWER-MODULE?sku=S65-MTM-PM-1>

## Brainstem Network Discovery

From the discovery we found a MTM-EtherStem with serial number D272031D. We will need this number in order to perform an Brainstem network discovery. To perform the indirect discovery we will need to use the '-r' parameter. The '-r' is for router and this tells Updater to look for anything at that devices router level and below.

```
$> ./Updater -D -r 0xD272031D
```

```
Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem Release:2.1.5] [Copyright (C) 1994-2015, Acroname Inc.]
```

```
Discovering Network Devices from [D272031D] via [TCPIP]:
Device   Module   Router   Model           Firmware Version   [IP address]
D272031D 04       04       0F [MTMEtherStem]  2.2.0 (0)        [10.128.38.159]
2181F0EE 06       04       0E [MTMPM]         2.1.4 (239384838) [10.128.38.159]
CA6A1B05 08       04       0D [MTMIOSerial]   2.2.0 (0)        [10.128.38.159]
```

```
Completed processing: Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem Release:2.1.5]
[Copyright (C) 1994-2015, Acroname Inc.]
```

As you can see Updater returned 3 devices, One of which being the router device that we specified in the discovery. In other words Updater has returned all of the devices on the MTM-EtherStem's I2C *BrainStem network*.

## Updating via the Brainstem Network.

Now that we have the serial number of the Brainstem network device we can form our final command to update the device. The command is very similar to the previous one with the exception of swapping out '-D' (discovery) for '-GU' (get and update). Additionally, we will need to add the '-d' (device) parameter so that it knows which device to update.

```
$> ./Updater -G -U -r 0xD272031D -d 0x2181F0EE
```

```
Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem Release:2.1.5] [Copyright (C) 1994-2015, Acroname Inc.]

Latest firmware for [2181F0EE] [aMTMPM1] is build [264993366].
Getting firmware build [264993366] for [2181F0EE] [aMTMPM1].
Downloaded firmware into local file [/Users/Mitch/.acroname/updater/2181F0EE/264993366.bird] VERIFIED.
Network Update of device [2181F0EE] via [D272031D][00]
Discovering Module # of Network Device [2181F0EE] thru [D272031D]
Using Module #[06] for Network Device
Update using firmware file [/Users/Mitch/.acroname/updater/2181F0EE/264993366.bird].
Transferring loader to device
Transferring loader block 0, 8800 bytes
Transferred 1 blocks, 8800 total bytes
Transferring firmware to device [/Users/Mitch/.acroname/updater/2181F0EE/264993366.bird]
Transferring firmware block 0, 772 bytes

(4 of 772 bytes (0%) of firmware block transferred.
(772 of 772 bytes (100%) of firmware block transferred.
```

(continues on next page)

(continued from previous page)

```
Transferring firmware block 1, 33528 bytes
```

```
(4 of 33528 bytes (0%) of firmware block transferred.  

(1028 of 33528 bytes (3%) of firmware block transferred.  

(2052 of 33528 bytes (6%) of firmware block transferred.  

(3076 of 33528 bytes (9%) of firmware block transferred.  

(4100 of 33528 bytes (12%) of firmware block transferred.  

(5124 of 33528 bytes (15%) of firmware block transferred.  

(6148 of 33528 bytes (18%) of firmware block transferred.  

(7172 of 33528 bytes (21%) of firmware block transferred.  

(8196 of 33528 bytes (24%) of firmware block transferred.  

(9220 of 33528 bytes (27%) of firmware block transferred.  

(10244 of 33528 bytes (30%) of firmware block transferred.  

(11268 of 33528 bytes (33%) of firmware block transferred.  

(12292 of 33528 bytes (36%) of firmware block transferred.  

(13316 of 33528 bytes (39%) of firmware block transferred.  

(14340 of 33528 bytes (42%) of firmware block transferred.  

(15364 of 33528 bytes (45%) of firmware block transferred.  

(16388 of 33528 bytes (48%) of firmware block transferred.  

(17412 of 33528 bytes (51%) of firmware block transferred.  

(18436 of 33528 bytes (54%) of firmware block transferred.  

(19460 of 33528 bytes (58%) of firmware block transferred.  

(20484 of 33528 bytes (61%) of firmware block transferred.  

(21508 of 33528 bytes (64%) of firmware block transferred.  

(22532 of 33528 bytes (67%) of firmware block transferred.  

(23556 of 33528 bytes (70%) of firmware block transferred.  

(24580 of 33528 bytes (73%) of firmware block transferred.  

(25604 of 33528 bytes (76%) of firmware block transferred.  

(26628 of 33528 bytes (79%) of firmware block transferred.  

(27652 of 33528 bytes (82%) of firmware block transferred.  

(28676 of 33528 bytes (85%) of firmware block transferred.  

(29700 of 33528 bytes (88%) of firmware block transferred.  

(30724 of 33528 bytes (91%) of firmware block transferred.  

(31748 of 33528 bytes (94%) of firmware block transferred.  

(32772 of 33528 bytes (97%) of firmware block transferred.  

(33528 of 33528 bytes (100%) of firmware block transferred.
```

```
Transferred 2 blocks, 34300 total bytes
```

```
Resetting router number of device:[04:2181f0ee] with module# of router[D272031D]
```

```
Device [2181F0EE] may need a physical reset before router number can be reset.
```

```
Completed updating firmware on device [2181F0EE]
```

```
Sending Reset to device [2181F0EE]
```

```
Completed processing: Updater [Version: 1.0 Dec 30 2015 15:59:07] [BrainStem_
```

```
↳Release:2.1.5]
```

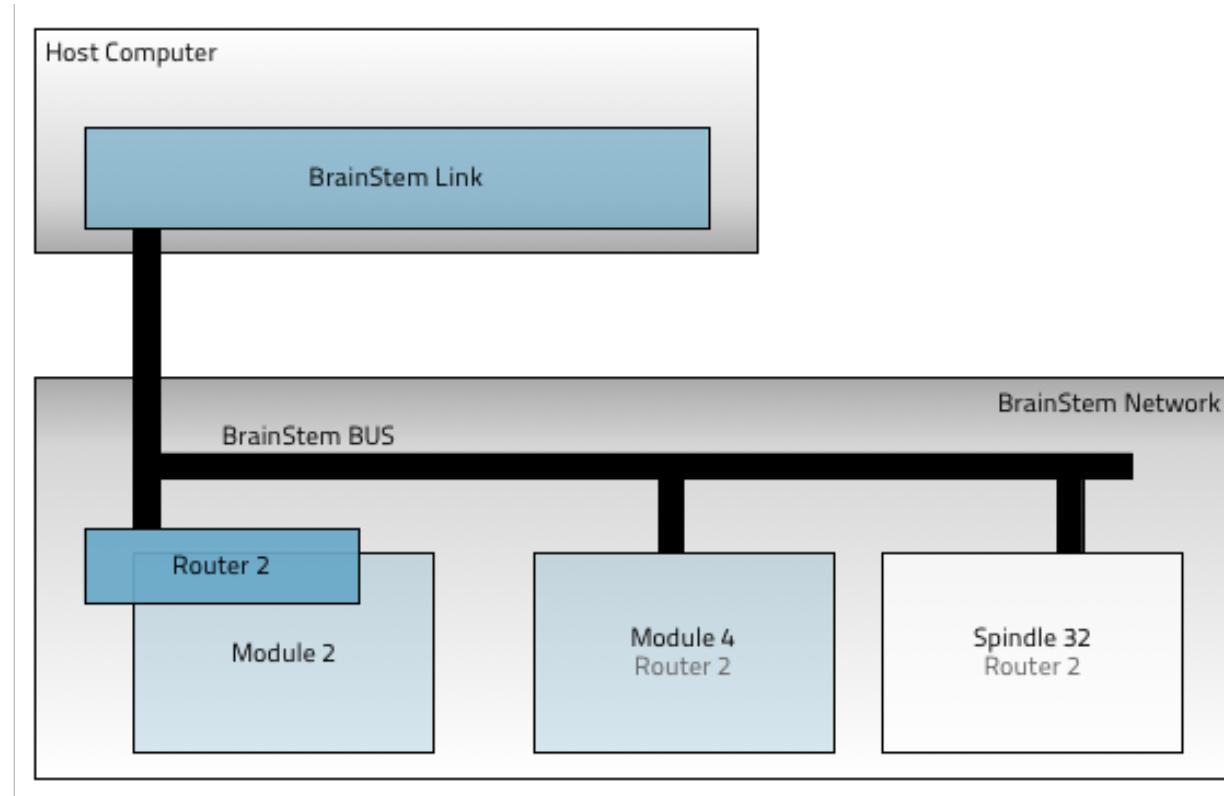
```
[Copyright (C) 1994-2015, Acroname Inc.]
```

**Please note that the Updater utility may appear to hang after it has transferred the firmware. This is because the device is waiting for the device to reset so that it can try and return all the previous BrainStem network settings (ie router and module addresses) for a seamless update process; however, with some updates a hard reset will be required. Simply press the reset button on the development board or power cycle the device.**

## 4.5 Terminology

### 4.5.1 BrainStem® Network

The BrainStem is a network of devices that offer rich I/O capabilities and comprehensive interactions. The hardware comes primarily in the form of modules and that can be combined to accomplish 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 I<sup>2</sup>C<sup>25</sup> as the hardware transport. The brainstem network is a multiple master I<sup>2</sup>C 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 I<sup>2</sup>C peripherals to the BrainStem Module's peripheral I<sup>2</sup>C ports.

---

<sup>25</sup> <http://i2c.info/i2c-bus-specification>

### 4.5.3 Routing

Each module in the BrainStem Network has a unique I<sup>2</sup>C 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 I<sup>2</sup>C bus.

Modules can interact with one another on the BrainStem Network as peers where each can manipulate one-another's I/O. From the I<sup>2</sup>C parlance, this means the BrainStem Network has multiple masters.

When a module needs to communicate back to the host (if present), the module can send the information to the router and the router will take care of relaying that information back across the link transport to the host.

### 4.5.4 Module

Modules are the heart of the BrainStem architecture. Each is a self-contained hardware solution that employs the BrainStem OS and can communicate with other modules over the BrainStem Network. Additionally, all modules have a link transport that enables them to talk with a host computer outside the BrainStem Network. Examples of link transports a module may use are TCP/IP, BlueTooth, USB, or other industry standards. Available BrainStem modules are listed in the [BrainStem Products webpage](#).

### 4.5.5 Host

The host is typically a larger computer or compute environment. These are most often desktop, mobile, or embedded processors running MacOS, Windows, or Linux operating systems. The protocols and transports are well documented so there is no practical restriction on what the host is running or how it works, it simply must be able to support the industry standard transport link mechanism. Many tools are provided for the above mentioned operating systems to allow control, configuration, and updating of the BrainStem modules across the host's link to the BrainStem Network.

### 4.5.6 Reflex

The Reflex programming language is a C-like language that runs on BrainStem controllers. Its simple interface allows a user to quickly implement application specific functionality on a BrainStem module to provide process control, data acquisition, filtering, and other custom behavior on the hardware. Reflex provides a flexible event driven architecture of embedded code, and was born out of the hierarchical control model used in many robotic systems. Reflexes running on BrainStem are ideally suited to reactive first line behaviors just above the metal. Further information about the reflex Language can be found within in the [Reflex Language](#) section.

### 4.5.7 Entity

An Entity provides a way to interact with a type of Hardware I/O within a BrainStem module. Entities include Digital inputs and outputs, analog inputs and outputs, I<sup>2</sup>C bus', Serial UARTS, system components, and other specialized hardware classes. Entities are the fundamental building blocks of interaction between BrainStem 'clients' and the hardware that BrainStems interact with. See the reference section on [Entities](#) for more in-depth discussion.

## 4.5.8 Discovery

The BrainStem API provides a mechanism for discovering the devices that are currently connected to the Host computer. This is part of the [C API](#) on the C/C++ library, and part of the discovery module within the [Python package](#). The Discovery API provides methods to find connection details for a specific module, as well as methods to list all connected modules. The Discovery methods return Spec objects which represent the required connection details for the device, as well as the Device's *model number*. The list of model numbers is provided on the [C API](#) page and the [Python API](#) page.

# 4.6 USB Drivers

Acroname has removed the need for kernel drivers for BrainStem devices across all of the platforms that we support. There is no longer a need to install drivers. However, both Linux and Windows 7 have steps that are required to allow BrainStem devices to work properly.

## 4.6.1 Mac OS X

On Mac OSX there are no installation procedures required.

## 4.6.2 Linux Ubuntu

Acroname is currently building and testing with Ubuntu 14.04LTS, on these systems users must run a script to properly set ownership and permissions for BrainStem devices. The script is located within the brainstem\_linux\_driverless folder, and is called udev.sh.

```
$> cd /path/to/brainstem_linux_driverless  
$> ./udev.sh
```

---

**Note:** Executing the commands within udev.sh requires sudo privileges. You will be prompted for your login password when you execute the script. Once you execute the script, you may have to log out and back in.

---

## 4.6.3 Windows 7 USB Driverless Installation

On the Windows 7 OS an installation is required to allow BrainStem devices to be recognized by the system. BrainStem devices use the Microsoft provided WinUSB device drivers to communicate with the brainstem. On Windows 7 operating systems the WinUSB driver is not installed automatically. On more modern versions of Windows newer than 7 this process is automatic and BrainStem devices need no install.

There is a windows\_driver\_installation.pdf within the Drivers folder of the Brainstem development kit, and HubTool downloads that describes the process for installing the WinUSB Driver on Windows 7 OS.

## 4.7 Appendix

## 4.7.1 Appendix I: BrainStem Universal Entity Interface (UEI)

Most of the BrainStem 2.0 functionality is represented by abstract entities. These entities are things like battery voltage, the module address, or an analog voltage. These entities are accessed in a common command interface called a UEI. These UEIs allow various clients to access module entities both locally and over the BrainStem's network. Clients include the Host, and Reflex code running on the module or on another module in the network.

### How UEI's Work

UEI's allow the setting and getting of entity information. This information can be in empty, byte, 2-byte, or 4-byte data sizes and the UEI's allow a common mechanism for either reading or writing these entity values. Some entities are limited to strictly reading (getting) or writing (setting), based on the underlying entity properties. For instance, and 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.

### 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.

### 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.
- Option - The specific entity within the class of UEI's
- Index - The array index of the class. This allows for multiple groups of classes like servos, motion channels, etc.
- Operation - the operation of the UEI which is GET in the case where the possible operations are GET, VAL, SET, and ACK
- Reply - Identifies the requestor so the response returns to that requestor where the possible options include I2C, Host, Reflex Machine, etc.
- ReplyID (optional) - For reply values that require more information, this information is included such as the Reflex Machine thread identifier or the remote module's address 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.

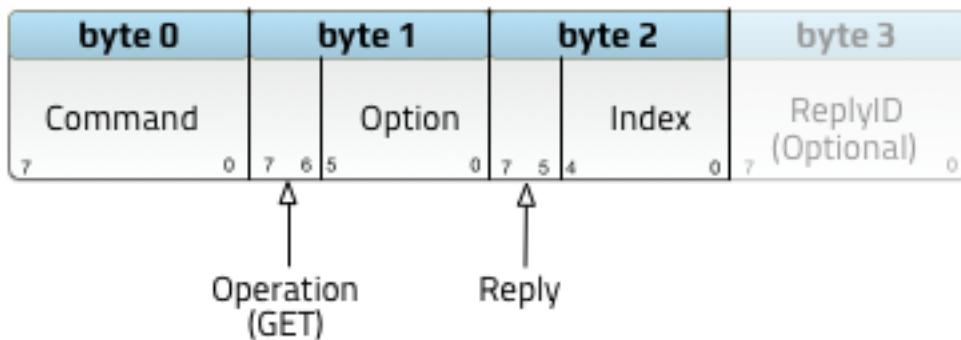


Fig. 1: Packed Byte GET UEI Structure

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, or 4-byte value, depending on the entity's native size. There are then three 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. Finally, the data (1-4 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.
- Option - The specific entity within the class of UEI's.
- Index - The array index of the class. This allows for multiple groups of classes like servos, motion channels, etc.
- Operation - The operation of the UIE which is VAL in this case where the possible operations are GET, VAL, SET, and ACK.
- 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.
- Data - 1, 2, or 4 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.
- 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.

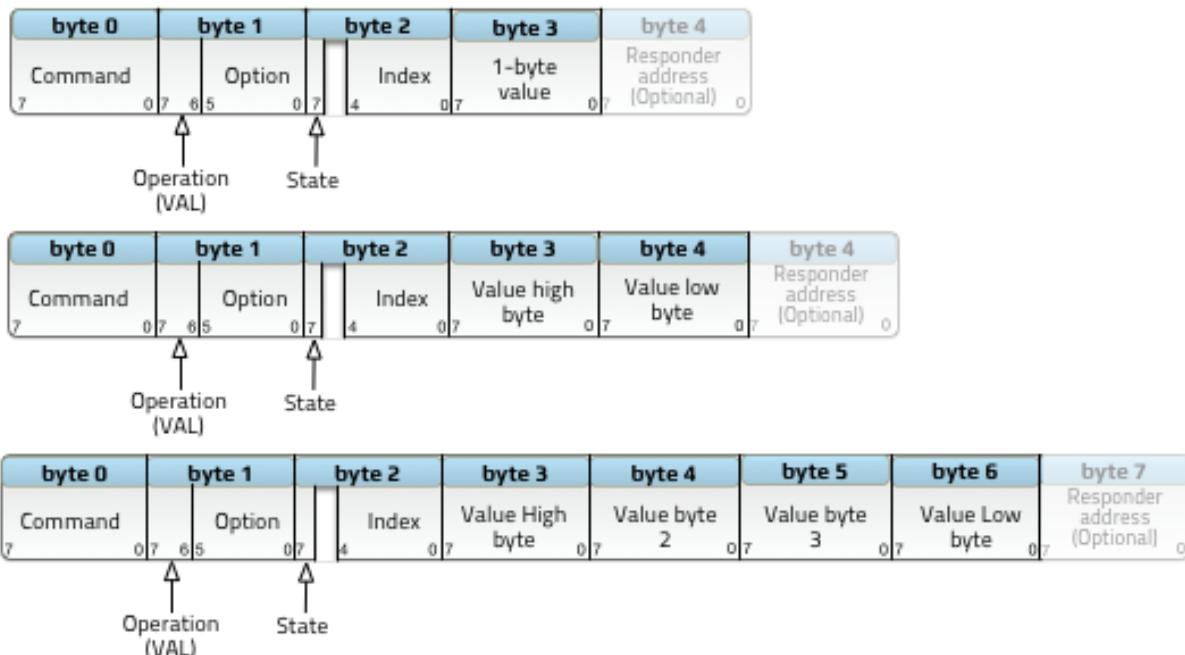


Fig. 2: Packed Byte VAL UEI Structure 1,2, and 4 Byte Values

## 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.

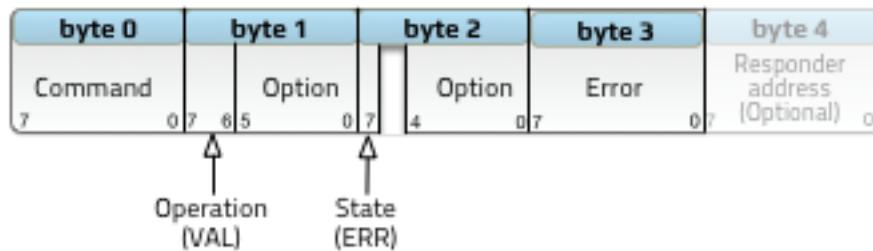


Fig. 3: Packed Byte VAL UEI Structure 1,2, and 4 Byte Values

## 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 acknowledgement 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.

### UEI SET Request

- Command - The command that groups the class of UEI's in which the entity being written is included.
- Option - The specific entity within the class of UEI's.
- Index - The array index of the class. This allows for multiple groups of classes like servos, motion channels, etc.
- Operation - The operation of the UIE which is SET in this case where the possible operations are GET, VAL, SET, and ACK.
- Reply - Identifies the requestor so the acknowledgement returns to that requestor where the possible options include none (no acknowledgement), I2C, Host, Reflex Machine, etc.
- Data - empty, 1, 2, or 4 bytes for the entity value or trigger being written.
- ReplyID (optional) - For acknowledgements that require more information, this information is included such as the Reflex Machine thread identifier or the remote module's address where the acknowledgement response will be sent. When acting as a requestor, the host does not have any additional replyID information.

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.

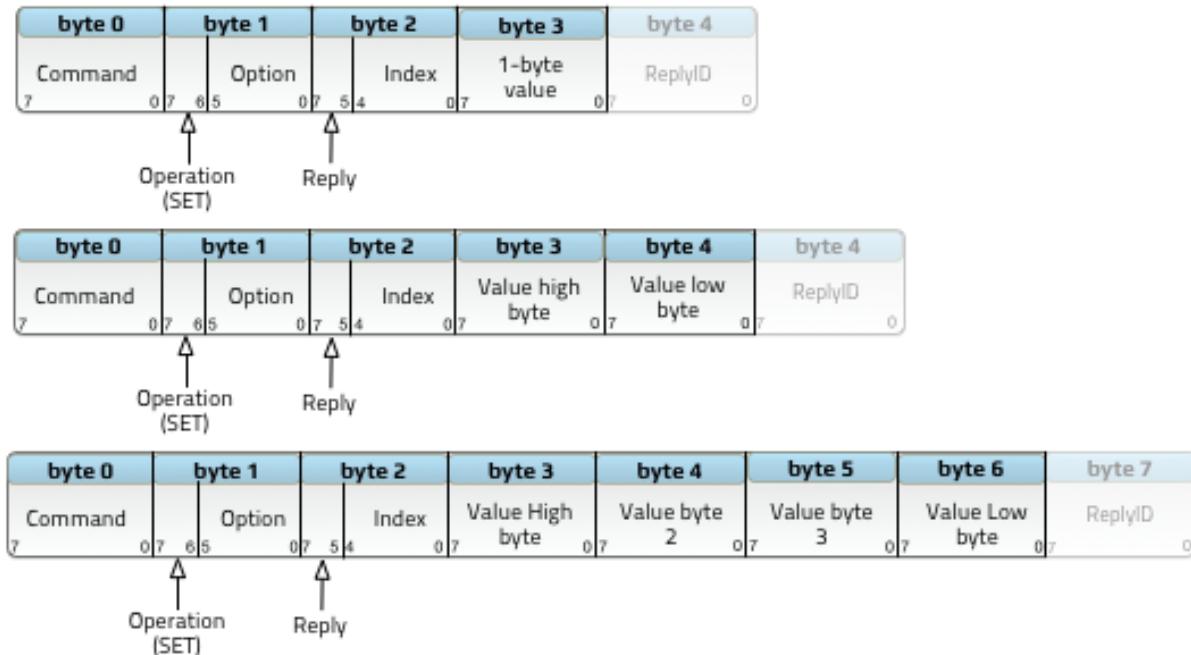


Fig. 4: Packed Byte SET UEI Structure 1,2, and 4 Byte Values

### UEI ACK Reply

- Command - The command that groups the class of UEI's in which the entity is included.
- Option - The specific entity within the class of UEI's.
- Index - The array index of the class. This allows for multiple groups of classes like servos, motion channels, etc.
- Operation - The operation of the UIE which is ACK in this case where the possible operations are GET, VAL, SET, and ACK.
- 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.

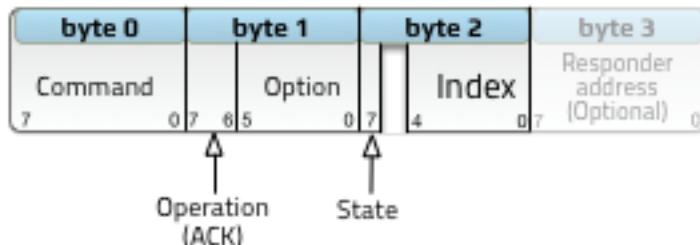


Fig. 5: Packed Byte ACK UEI Structure

### 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.

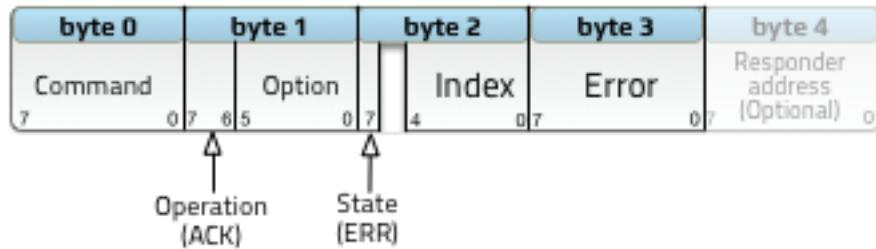


Fig. 6: Packed Byte ACK UEI Error Structure

## 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 three 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.

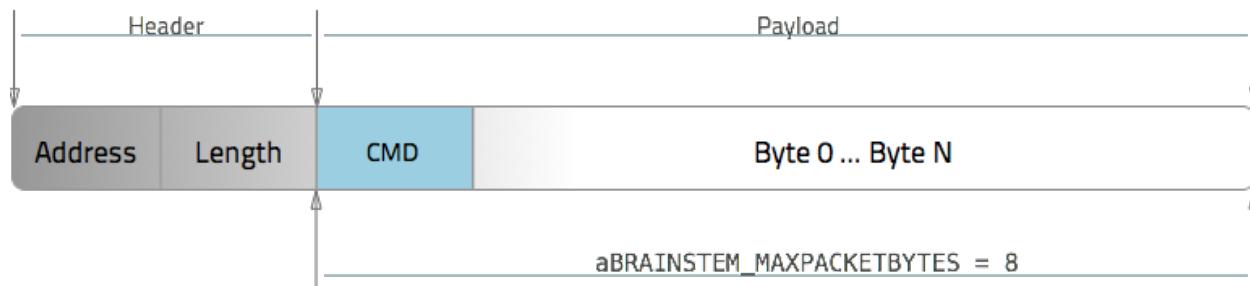


Fig. 7: Typical BrainStem Command Structure

### 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` (8 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 littleEndian byte order.

## Command Interaction

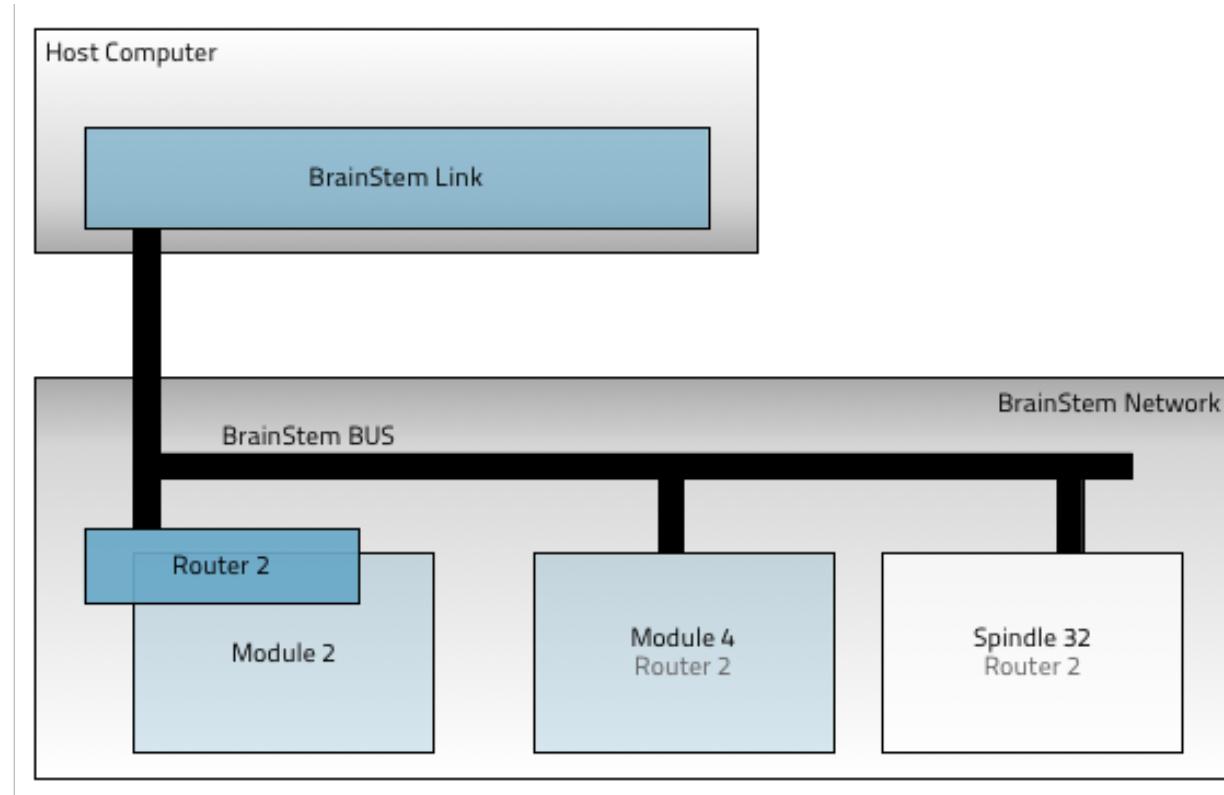
*UEI*'s impose a request response behavior on top of the BrainStem protocol, but the protocol does not itself define a need for a response. Delivery of data is best effort. Some Link transports (TCP/IP) make guarantees about data delivery, this is not part of the BrainStem protocol. Please see specific command documentation to determine whether to expect a response packet.

There are currently two cases when a BrainStem client may receive an error message unrelated to a request. The first case occurs when a module address is given where no such module exists, in this case a cmdMSG packet will be received with a no I2C ack payload. The second case occurs when a Reflex VM exits unexpectedly, a client may receive a cmdMSG packet with a vm exit payload. Please see the Command reference section for more information.

### 4.7.3 Appendix III: BrainStem Networking

The BrainStem bus is the network backbone of the BrainStem network. Most BrainStem modules, including all MTM modules, use an I<sub>2</sub>C<sup>26</sup> as the hardware transport. The brainstem network is a multiple master I<sub>2</sub>C 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 I<sub>2</sub>C peripherals to a BrainStem module's peripheral I<sub>2</sub>C ports.

BrainStem networks closely mirror standard I<sub>2</sub>C networks, but aren't necessarily always on an I<sub>2</sub>C physical network. For example, BrainStem network as described below may use a CAN bus physical network.



<sup>26</sup> <http://i2c.info/i2c-bus-specification>

## Module Addresses

BrainStem devices rely on having a unique module address on the bus that following I2C conventions. The Brainstem module address is a single unsigned-byte, and can take even (non-odd) values from 2 to 254. Each class of BrainStem module has a specific default base address, listed in the table below. A software offset to this address can be set with the BrainStem API, and MTM modules include a set of hardware offset pins which can be used to modify module addresses with external pin connections.

BrainStem Model	Default Base Address
40Pin BrainStems EtherStem USBStem	2
MTM-EtherStem MTM-USBStem	4
MTM-PM-1	6
USBHub2x4 USBHub3+	6
MTM-IO-Serial	8
MTM-DAQ-2	10
MTM-Relay	12

## Hardware Offsets

Hardware offset pins are useful when more than one of the same type of module (i.e. modules with the same base address) are installed on a single BrainStem network. Applying a different hardware offset to each module of the same type the modules to seamlessly and automatically be configured on the network for inter-module communication. Further, modules can be simply swapped in and out of the network without needing to pre-configure a module's address before being added to a network. Finally, when a system has more than one of the same type of module in a network, the module's hardware offset can be used to determine the module's physical location and thus its interconnection and intended function.

Each hardware offset pin can be left floating or pulled to ground with a  $1\text{k}\Omega$  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 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

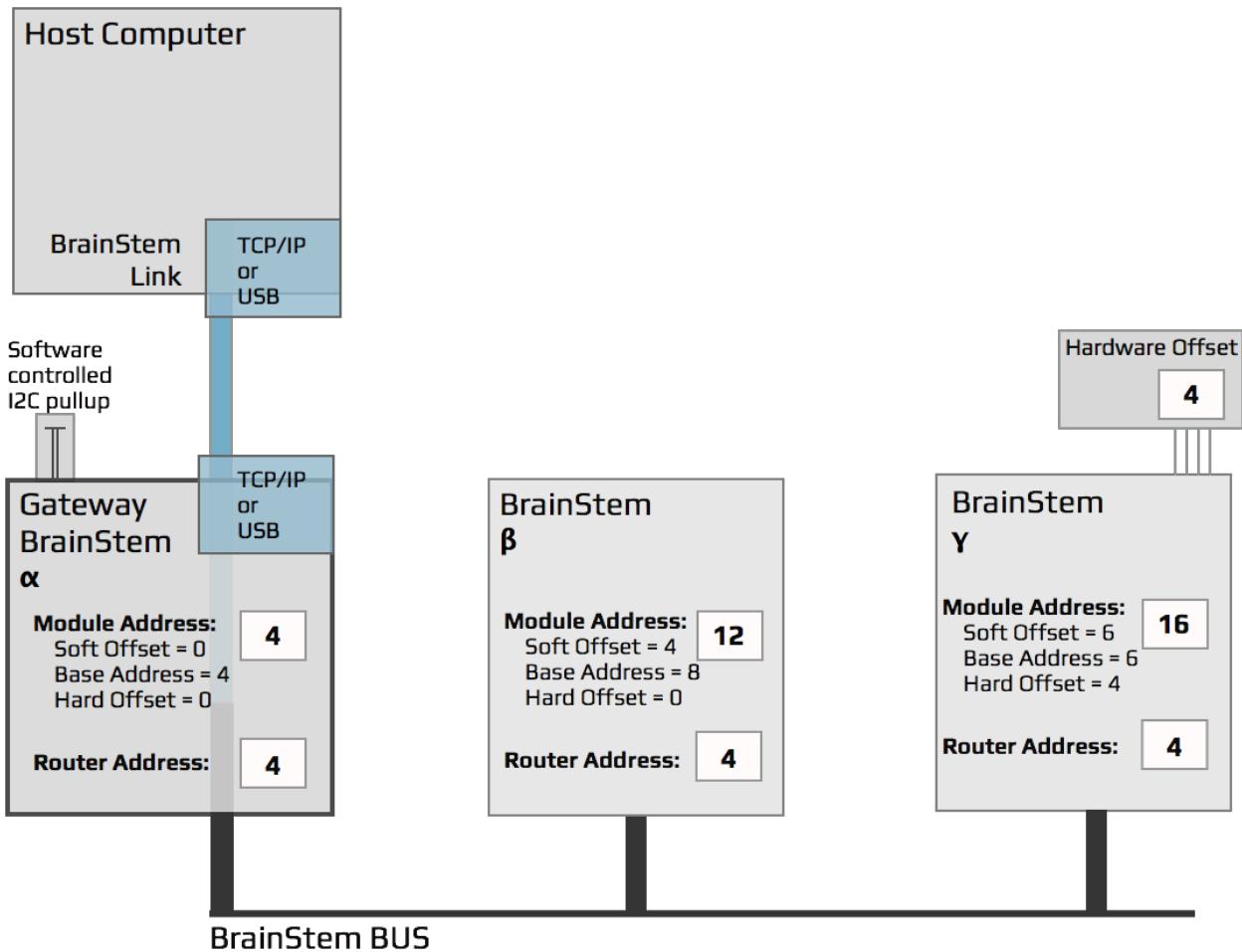
## Router Addresses

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 its module address, it will route any communication intended for the host computer to its router's address on the BrainStem network.

By default, each BrainStem's router address is set to its default base address plus any offset. In this way, each module will communicate over its transport link out of the box. In order to have multiple BrainStem modules communicate across a BrainStem network over one transport link, each module needs have its router address configured. The BrainStem API provides a simple mechanism to quickly configure the router address of all modules on the same BrainStem network: `routeToMe`.

## Setting up a BrainStem Network

This section of the appendix will walk you through setting up the network shown in the figure below. This is fairly typical network containing three BrainStem devices since it represents a fully populated MTM development board.



## Out of the Box

In the example above the Routing or Gateway BrainStem ( $\alpha$ ) is set to route through itself; i.e. its module and router addresses are equal. This is the setup that comes with the BrainStem out of the box. To complete the example, two more BrainStem devices are needed and they will have their router and address software offsets changed (described below).

## I2C Pull-ups

Most BrainStem use an I2C physical network. I2C relies on bus pull-ups resistors. BrainStem modules have built-in  $330\Omega$  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  $\alpha$  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  $\alpha$  module:

```
>>> alpha.system.routeToMe(1)
```

After this, all modules in the network will start receiving and sending heartbeat traffic, and can be connected from the host:

```
>>> beta = brainstem.stem.MTMIOSerial()  
>>> beta.connectThroughLinkModule(alpha)
```

Similarly, the  $\gamma$  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 to the host. This powerful networking allows large networks of modules to be controlled from a single host link.

## Setting and saving address offsets and router: the hard(er) way

If it is desirable to configure modules to change their module address or router setting even through power cycles or resets, the BrainStem API provide an interface for directly setting and saving the address offset and the router of each module. This method is more complicated than the `routeToMe` interface, but is available to provide flexibility for complex network setups. Never worry, if a device's router address is saved to a non-default value simply creating a link directly to that module (e.g. via USB) will temporarily re-configure its route to itself so the link can be functional.

The *system entity* contains the options for getting and setting the module offsets and router address of the `brainstem` module. Module offsets and the router address are applied after a system save and reset. The python interpreter code below sets the module and router addresses for the two modules ( $\beta$  and  $\gamma$ ). The same exercise can be done in C++ with almost the same code. For this example, we will assume that both  $\beta$  (beta) and  $\gamma$  (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  $\beta$  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  $\alpha$ , the gateway BrainStem. As such, all host-bound communication will be routed to address 4 on the BrainStem network, instead of going through the module's transport link connector.

The following code sets the module software offset and the router settings on  $\gamma$  to match the diagram. Connecting to  $\gamma$  is done in same way as shown above for  $\beta$ , 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  $\alpha$  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:

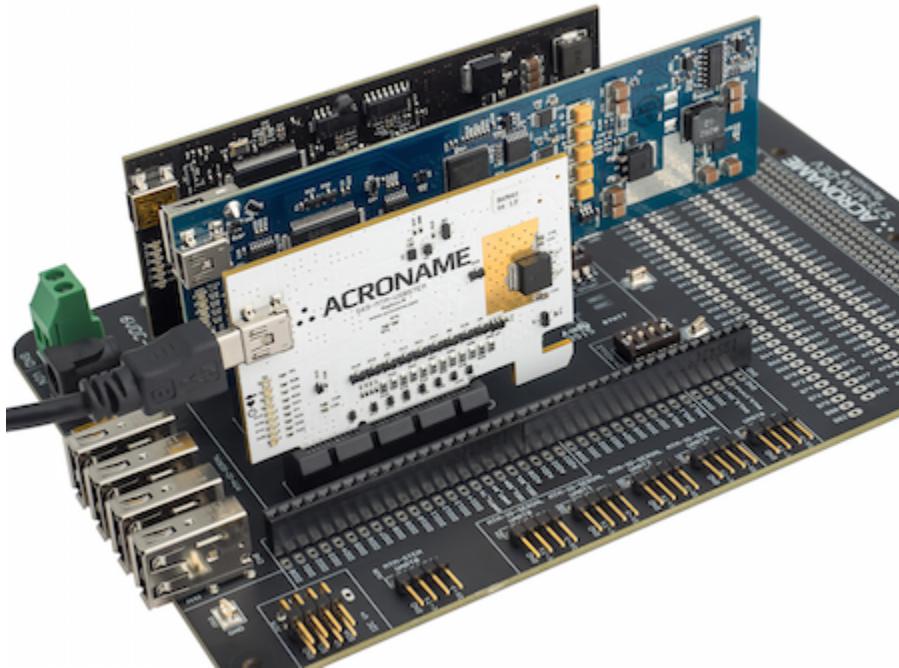
```
>>> beta.setModuleAddress(12) # Set the Module object's address so that we
#communicate correctly when we reconnect.
>>> gamma.setModuleAddress(16)
```

Finally we connect directly to the Gateway Brainstem  $\alpha$ , and then connect  $\beta$  and  $\gamma$  through  $\alpha$ 's link. We should now be able to successfully send commands and receive responses on all three brainstems through the single link connection to  $\alpha$ .

```
# Connect to the gateway BrainStem. All three stem heartbeats should be active.
>>> alpha.connect(discover.findFirstModule(link.Spec.USB))
>>> beta.connectThroughLinkModule(alpha) # Now reconnect the beta and gamma through
#the gateway module.
>>> gamma.connectThroughLinkModule(alpha)
```

## In Practice

The above example is intentionally complex in order to show the interaction of hardware offsets and software offsets within a BrainStem network. In most applications and with the MTM development board, usually the user will prefer to make minimal changes in order to form the brainstem network.



In a real life use case of an MTM-USBStem acting as the Gateway, and an MTM-IO-Serial and MTM-PM-1 boards acting as the other two devices, the only changes that need to be made are to set the router address of the MTM-IO-Serial, and the MTM-PM-1 to match the MTM-USBStem base address of 4. Each of the modules have unique base addresses so there are no software or hardware offsets to apply. Also, the default object instantiation can be used without having to set the module address.

```
>>> beta.setRouter(4)
>>> gamma.setRouter(4)
>>> beta.save()
>>> gamma.save()
>>> beta.reset()
>>> gamma.reset()
>>> beta.disconnect()
>>> gamma.disconnect()
>>> beta.connectThroughLinkModule(alpha)
>>> gamma.connectThroughLinkModule(alpha)
```

## 4.7.4 Appendix IV: Updater File Structure

As we discussed in the “[BrainStem Firmware Management](#)” section, Updater is our new tool updating and recovering your Brainstem modules. After playing around with it a bit you might have noticed that it keeps a history of all the devices it interacts with. In the following appendix I will be explaining the file structure Updater creates on your machine 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 currently implemented).

The remaining four items are all devices (serial numbers) that the Updater utility has connected to 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 communicate with the brainstem module. Although you have read/write access to this file it is recommended that you do not make any changes to this file. Lets take a look at what type of information is stored in the “.settings” file.

```
$> vi .history
```

```
FIRMWARE= 2.1.5
LAST_TRANSFER_DATE=Fri Dec 4 10:58:05 2015
LAST_TRANSFER_VERSION=/Users/Mitch/.acroname/updater/66F4859B/130702287.bird
LINK_TYPE=USB
MODEL_NUMBER=4
MODULE_NAME=USBStem
MODULE_NUMBER=2
SERIAL_NUMBER=0x66F4859B
```

### **“.bird” Files**

The remaining 4 items from the *device files* listed above are called “.bird” files. This is the file type in which we store our firmware. For this particular device I have updated the firmware 3 times and thus have 3 “.bird” files stored under this device. This can be very handy if you would like to return to a previous firmware. See “*Example: Reverting to a Previous Version*” in the “*BrainStem Firmware Management*” section.



# A

aBaudRate (*C++ enum*), 433  
aBaudRate::aBAUD\_115200 (*C++ enumerator*), 433  
aBaudRate::aBAUD\_19200 (*C++ enumerator*), 433  
aBaudRate::aBAUD\_230400 (*C++ enumerator*), 433  
aBaudRate::aBAUD\_2400 (*C++ enumerator*), 433  
aBaudRate::aBAUD\_38400 (*C++ enumerator*), 433  
aBaudRate::aBAUD\_4800 (*C++ enumerator*), 433  
aBaudRate::aBAUD\_57600 (*C++ enumerator*), 433  
aBaudRate::aBAUD\_9600 (*C++ enumerator*), 433  
aBRAINSTEM\_MAXPACKETBYTES (*C macro*), 404  
Acroname::BrainStem2CLI::aErr (*C++ enum*), 452  
Acroname::BrainStem2CLI::aErr::aErrAsyncReturn (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrBusy (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrCancel (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrConfiguration (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrConnection (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrDuplicate (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrEOF (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrFileNameLength (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrIndexRange (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrInitialization (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrInvalidEntity (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrInvalidOption (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrIO (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrMedia (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrMemory (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrMode (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrNone (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrNotFound (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrNotReady (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrOverrun (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrPacket (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrParam (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::aErr::aErrParse (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrPermission (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrRange (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrRead (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrResource (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrShortCommand (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrSize (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrTimeout (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrUnimplemented (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrUnknown (*C++ enumerator*), 454  
Acroname::BrainStem2CLI::aErr::aErrVersion (*C++ enumerator*), 453  
Acroname::BrainStem2CLI::aErr::aErrWrite (*C++ enumerator*), 452  
Acroname::BrainStem2CLI::AnalogClass (*C++ class*), 454

Acroname::BrainStem2CLI::AnalogClass::~AnalogClass (*C++ function*), 455  
Acroname::BrainStem2CLI::AnalogClass::AnalogClass (*C++ function*), 455  
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureNumberOfSamples (*C++ function*), 457  
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureSampleRate (*C++ function*), 457  
Acroname::BrainStem2CLI::AnalogClass::getBulkCaptureState (*C++ function*), 457  
Acroname::BrainStem2CLI::AnalogClass::getConfiguration (*C++ function*), 456  
Acroname::BrainStem2CLI::AnalogClass::getEnable (*C++ function*), 456  
Acroname::BrainStem2CLI::AnalogClass::getRange (*C++ function*), 456  
Acroname::BrainStem2CLI::AnalogClass::getValue (*C++ function*), 455  
Acroname::BrainStem2CLI::AnalogClass::getVoltage (*C++ function*), 455  
Acroname::BrainStem2CLI::AnalogClass::initiateBulkCapture (*C++ function*), 457  
Acroname::BrainStem2CLI::AnalogClass::setBulkCaptureNumberOfSamples (*C++ function*), 457  
Acroname::BrainStem2CLI::AnalogClass::setBulkCaptureSampleRate (*C++ function*), 457  
Acroname::BrainStem2CLI::AnalogClass::setConfiguration (*C++ function*), 456  
Acroname::BrainStem2CLI::AnalogClass::setEnable (*C++ function*), 456  
Acroname::BrainStem2CLI::AnalogClass::setRange (*C++ function*), 456  
Acroname::BrainStem2CLI::AnalogClass::setValue (*C++ function*), 455  
Acroname::BrainStem2CLI::AnalogClass::setVoltage (*C++ function*), 455  
Acroname::BrainStem2CLI::AppClass (*C++ class*), 458  
Acroname::BrainStem2CLI::AppClass::~AppClass (*C++ function*), 458  
Acroname::BrainStem2CLI::AppClass::AppClass (*C++ function*), 458  
Acroname::BrainStem2CLI::AppClass::execute (*C++ function*), 458  
Acroname::BrainStem2CLI::ClockClass (*C++ class*), 459  
Acroname::BrainStem2CLI::ClockClass::~ClockClass (*C++ function*), 459  
Acroname::BrainStem2CLI::ClockClass::ClockClass (*C++ function*), 459  
Acroname::BrainStem2CLI::ClockClass::getDay (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::getHour (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::getMinute (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::getMonth (*C++ function*), 459  
Acroname::BrainStem2CLI::ClockClass::getSecond (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::getYear (*C++ function*), 459  
Acroname::BrainStem2CLI::ClockClass::setDay (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::setHour (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::setMinute (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::setMonth (*C++ function*), 460  
Acroname::BrainStem2CLI::ClockClass::setSecond (*C++ function*), 461  
Acroname::BrainStem2CLI::ClockClass::setYear (*C++ function*), 459  
Acroname::BrainStem2CLI::DigitalClass (*C++ class*), 461  
Acroname::BrainStem2CLI::DigitalClass::~DigitalClass (*C++ function*), 461  
Acroname::BrainStem2CLI::DigitalClass::DigitalClass (*C++ function*), 461  
Acroname::BrainStem2CLI::DigitalClass::getConfiguration (*C++ function*), 462  
Acroname::BrainStem2CLI::DigitalClass::getState (*C++ function*), 461  
Acroname::BrainStem2CLI::DigitalClass::getStateAll (*C++ function*), 462  
Acroname::BrainStem2CLI::DigitalClass::setConfiguration (*C++ function*), 461  
Acroname::BrainStem2CLI::DigitalClass::setState (*C++ function*), 461  
Acroname::BrainStem2CLI::DigitalClass::setStateAll (*C++ function*), 462  
Acroname::BrainStem2CLI::EqualizerClass (*C++ class*), 462  
Acroname::BrainStem2CLI::EqualizerClass::~EqualizerClass (*C++ function*), 463  
Acroname::BrainStem2CLI::EqualizerClass::EqualizerClass (*C++ function*), 463  
Acroname::BrainStem2CLI::EqualizerClass::getReceiverConfig (*C++ function*), 463  
Acroname::BrainStem2CLI::EqualizerClass::getTransmitterConfig (*C++ function*), 463  
Acroname::BrainStem2CLI::EqualizerClass::setReceiverConfig (*C++ function*), 463  
Acroname::BrainStem2CLI::EqualizerClass::setTransmitterConfig (*C++ function*), 463  
Acroname::BrainStem2CLI::I2CClass (*C++ class*), 463  
Acroname::BrainStem2CLI::I2CClass::~I2CClass (*C++ function*), 464  
Acroname::BrainStem2CLI::I2CClass::getSpeed (*C++ function*), 464  
Acroname::BrainStem2CLI::I2CClass::I2CClass (*C++ function*), 464  
Acroname::BrainStem2CLI::I2CClass::read (*C++ function*), 464  
Acroname::BrainStem2CLI::I2CClass::setPullup (*C++ function*), 464  
Acroname::BrainStem2CLI::I2CClass::setSpeed (*C++ function*), 464  
Acroname::BrainStem2CLI::I2CClass::write (*C++ function*), 464  
Acroname::BrainStem2CLI::ModuleClass (*C++ class*), 465  
Acroname::BrainStem2CLI::ModuleClass::~ModuleClass (*C++ function*), 465  
Acroname::BrainStem2CLI::ModuleClass::connect (*C++ function*), 466  
Acroname::BrainStem2CLI::ModuleClass::connectFromSpec (*C++ function*), 466  
Acroname::BrainStem2CLI::ModuleClass::connectThroughLinkModule (*C++ function*), 466  
Acroname::BrainStem2CLI::ModuleClass::disconnect (*C++ function*), 467  
Acroname::BrainStem2CLI::ModuleClass::discoverAndConnect (*C++ function*), 465

Acroname::BrainStem2CLI::ModuleClass::findFirstModule (*C++ function*), 468  
 Acroname::BrainStem2CLI::ModuleClass::findModule (*C++ function*), 468  
 Acroname::BrainStem2CLI::ModuleClass::getModuleAddress (*C++ function*), 467  
 Acroname::BrainStem2CLI::ModuleClass::isConnected (*C++ function*), 467  
 Acroname::BrainStem2CLI::ModuleClass::ModuleClass (*C++ function*), 465  
 Acroname::BrainStem2CLI::ModuleClass::reconnect (*C++ function*), 467  
 Acroname::BrainStem2CLI::ModuleClass::sDiscover (*C++ function*), 468  
 Acroname::BrainStem2CLI::ModuleClass::setModuleAddress (*C++ function*), 467  
 Acroname::BrainStem2CLI::ModuleClass::setNetworkingMode (*C++ function*), 467  
 Acroname::BrainStem2CLI::MuxClass (*C++ class*), 468  
 Acroname::BrainStem2CLI::MuxClass::~MuxClass (*C++ function*), 469  
 Acroname::BrainStem2CLI::MuxClass::getChannel (*C++ function*), 469  
 Acroname::BrainStem2CLI::MuxClass::getChannelVoltage (*C++ function*), 469  
 Acroname::BrainStem2CLI::MuxClass::getConfiguration (*C++ function*), 470  
 Acroname::BrainStem2CLI::MuxClass::getEnable (*C++ function*), 469  
 Acroname::BrainStem2CLI::MuxClass::getSplitMode (*C++ function*), 470  
 Acroname::BrainStem2CLI::MuxClass::MuxClass (*C++ function*), 469  
 Acroname::BrainStem2CLI::MuxClass::setChannel (*C++ function*), 469  
 Acroname::BrainStem2CLI::MuxClass::setConfiguration (*C++ function*), 470  
 Acroname::BrainStem2CLI::MuxClass::setEnabled (*C++ function*), 469  
 Acroname::BrainStem2CLI::MuxClass::setSplitMode (*C++ function*), 470  
 Acroname::BrainStem2CLI::PointerClass (*C++ class*), 470  
 Acroname::BrainStem2CLI::PointerClass::~PointerClass (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::getChar (*C++ function*), 472  
 Acroname::BrainStem2CLI::PointerClass::getInt (*C++ function*), 472  
 Acroname::BrainStem2CLI::PointerClass::getMode (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::getOffset (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::getShort (*C++ function*), 472  
 Acroname::BrainStem2CLI::PointerClass::getTransferStore (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::initiateTransferFromStore (*C++ function*), 472  
 Acroname::BrainStem2CLI::PointerClass::initiateTransferToStore (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::PointerClass (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::setChar (*C++ function*), 472  
 Acroname::BrainStem2CLI::PointerClass::setInt (*C++ function*), 472  
 Acroname::BrainStem2CLI::PointerClass::setMode (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::setOffset (*C++ function*), 471  
 Acroname::BrainStem2CLI::PointerClass::setShort (*C++ function*), 472  
 Acroname::BrainStem2CLI::PointerClass::setTransferStore (*C++ function*), 471  
 Acroname::BrainStem2CLI::PortClass (*C++ class*), 473  
 Acroname::BrainStem2CLI::PortClass::~PortClass (*C++ function*), 473  
 Acroname::BrainStem2CLI::PortClass::getAllocatedPower (*C++ function*), 480  
 Acroname::BrainStem2CLI::PortClass::getAvailablePower (*C++ function*), 479  
 Acroname::BrainStem2CLI::PortClass::getCC1Enabled (*C++ function*), 477  
 Acroname::BrainStem2CLI::PortClass::getCC2Enabled (*C++ function*), 478  
 Acroname::BrainStem2CLI::PortClass::getCCEnabled (*C++ function*), 477  
 Acroname::BrainStem2CLI::PortClass::getCurrentLimit (*C++ function*), 479  
 Acroname::BrainStem2CLI::PortClass::getCurrentLimitMode (*C++ function*), 479  
 Acroname::BrainStem2CLI::PortClass::getDataEnabled (*C++ function*), 474  
 Acroname::BrainStem2CLI::PortClass::getDataHS1Enabled (*C++ function*), 475  
 Acroname::BrainStem2CLI::PortClass::getDataHS2Enabled (*C++ function*), 475  
 Acroname::BrainStem2CLI::PortClass::getDataHSEnabled (*C++ function*), 474  
 Acroname::BrainStem2CLI::PortClass::getDataHSRoutingBehavior (*C++ function*), 481  
 Acroname::BrainStem2CLI::PortClass::getDataRole (*C++ function*), 476  
 Acroname::BrainStem2CLI::PortClass::getDataSpeed (*C++ function*), 478  
 Acroname::BrainStem2CLI::PortClass::getDataSS1Enabled (*C++ function*), 476  
 Acroname::BrainStem2CLI::PortClass::getDataSS2Enabled (*C++ function*), 476  
 Acroname::BrainStem2CLI::PortClass::getDataSSEnabled (*C++ function*), 475  
 Acroname::BrainStem2CLI::PortClass::getDataSSRoutingBehavior (*C++ function*), 481  
 Acroname::BrainStem2CLI::PortClass::getEnabled (*C++ function*), 474  
 Acroname::BrainStem2CLI::PortClass::getErrors (*C++ function*), 479  
 Acroname::BrainStem2CLI::PortClass::getHSBoost (*C++ function*), 482  
 Acroname::BrainStem2CLI::PortClass::getMode (*C++ function*), 478  
 Acroname::BrainStem2CLI::PortClass::getName (*C++ function*), 480  
 Acroname::BrainStem2CLI::PortClass::getPowerEnabled (*C++ function*), 476  
 Acroname::BrainStem2CLI::PortClass::getPowerLimit (*C++ function*), 480  
 Acroname::BrainStem2CLI::PortClass::getPowerLimitMode (*C++ function*), 480  
 Acroname::BrainStem2CLI::PortClass::getPowerMode (*C++ function*), 474  
 Acroname::BrainStem2CLI::PortClass::getState (*C++ function*), 478

Acroname::BrainStem2CLI::PortClass::getVbusAccumulatedPower (*C++ function*), 481  
Acroname::BrainStem2CLI::PortClass::getVbusCurrent (*C++ function*), 473  
Acroname::BrainStem2CLI::PortClass::getVbusVoltage (*C++ function*), 473  
Acroname::BrainStem2CLI::PortClass::getVconn1Enabled (*C++ function*), 477  
Acroname::BrainStem2CLI::PortClass::getVconn2Enabled (*C++ function*), 477  
Acroname::BrainStem2CLI::PortClass::getVconnAccumulatedPower (*C++ function*), 482  
Acroname::BrainStem2CLI::PortClass::getVconnCurrent (*C++ function*), 473  
Acroname::BrainStem2CLI::PortClass::getVconnEnabled (*C++ function*), 476  
Acroname::BrainStem2CLI::PortClass::getVconnVoltage (*C++ function*), 473  
Acroname::BrainStem2CLI::PortClass::getVoltageSetpoint (*C++ function*), 478  
Acroname::BrainStem2CLI::PortClass (*C++ function*), 473  
Acroname::BrainStem2CLI::PortClass::resetEntityToFactoryDefaults (*C++ function*), 482  
Acroname::BrainStem2CLI::PortClass::resetVbusAccumulatedPower (*C++ function*), 482  
Acroname::BrainStem2CLI::PortClass::resetVconnAccumulatedPower (*C++ function*), 482  
Acroname::BrainStem2CLI::PortClass::setCC1Enabled (*C++ function*), 478  
Acroname::BrainStem2CLI::PortClass::setCC2Enabled (*C++ function*), 478  
Acroname::BrainStem2CLI::PortClass::setCCEnabled (*C++ function*), 477  
Acroname::BrainStem2CLI::PortClass::setCurrentLimit (*C++ function*), 479  
Acroname::BrainStem2CLI::PortClass::setCurrentLimitMode (*C++ function*), 479  
Acroname::BrainStem2CLI::PortClass::setDataEnabled (*C++ function*), 474  
Acroname::BrainStem2CLI::PortClass::setDataHS1Enabled (*C++ function*), 475  
Acroname::BrainStem2CLI::PortClass::setDataHS2Enabled (*C++ function*), 475  
Acroname::BrainStem2CLI::PortClass::setDataHSEnabled (*C++ function*), 475  
Acroname::BrainStem2CLI::PortClass::setDataHSRoutingBehavior (*C++ function*), 481  
Acroname::BrainStem2CLI::PortClass::setDataSS1Enabled (*C++ function*), 476  
Acroname::BrainStem2CLI::PortClass::setDataSS2Enabled (*C++ function*), 476  
Acroname::BrainStem2CLI::PortClass::setDataSSEnabled (*C++ function*), 475  
Acroname::BrainStem2CLI::PortClass::setDataSSRoutingBehavior (*C++ function*), 481  
Acroname::BrainStem2CLI::PortClass::setEnabled (*C++ function*), 474  
Acroname::BrainStem2CLI::PortClass::setHSBoost (*C++ function*), 482  
Acroname::BrainStem2CLI::PortClass::setMode (*C++ function*), 479  
Acroname::BrainStem2CLI::PortClass::setName (*C++ function*), 481  
Acroname::BrainStem2CLI::PortClass::setPowerEnabled (*C++ function*), 476  
Acroname::BrainStem2CLI::PortClass::setPowerLimit (*C++ function*), 480  
Acroname::BrainStem2CLI::PortClass::setPowerLimitMode (*C++ function*), 480  
Acroname::BrainStem2CLI::PortClass::setPowerMode (*C++ function*), 474  
Acroname::BrainStem2CLI::PortClass::setVconn1Enabled (*C++ function*), 477  
Acroname::BrainStem2CLI::PortClass::setVconn2Enabled (*C++ function*), 477  
Acroname::BrainStem2CLI::PortClass::setVconnEnabled (*C++ function*), 477  
Acroname::BrainStem2CLI::PortClass::setVoltageSetpoint (*C++ function*), 478  
Acroname::BrainStem2CLI::PowerDeliveryClass (*C++ class*), 482  
Acroname::BrainStem2CLI::PowerDeliveryClass::~PowerDeliveryClass (*C++ function*), 483  
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableCurrentMax (*C++ function*), 486  
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableOrientation (*C++ function*), 487  
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableSpeedMax (*C++ function*), 487  
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableType (*C++ function*), 487  
Acroname::BrainStem2CLI::PowerDeliveryClass::getCableVoltageMax (*C++ function*), 486  
Acroname::BrainStem2CLI::PowerDeliveryClass::getConnectionState (*C++ function*), 483  
Acroname::BrainStem2CLI::PowerDeliveryClass::getFastRoleSwapCurrent (*C++ function*), 489  
Acroname::BrainStem2CLI::PowerDeliveryClass::getFlagMode (*C++ function*), 488  
Acroname::BrainStem2CLI::PowerDeliveryClass::getNumberOfPowerDataObjects (*C++ function*), 483  
Acroname::BrainStem2CLI::PowerDeliveryClass::getOverride (*C++ function*), 488  
Acroname::BrainStem2CLI::PowerDeliveryClass::getPeakCurrentConfiguration (*C++ function*), 489  
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObject (*C++ function*), 483  
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectEnabled (*C++ function*), 484  
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectEnabledList (*C++ function*), 485  
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerDataObjectList (*C++ function*), 484  
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerRole (*C++ function*), 485  
Acroname::BrainStem2CLI::PowerDeliveryClass::getPowerRolePreferred (*C++ function*), 486  
Acroname::BrainStem2CLI::PowerDeliveryClass::getRequestDataObject (*C++ function*), 485  
Acroname::BrainStem2CLI::PowerDeliveryClass::PowerDeliveryClass (*C++ function*), 483  
Acroname::BrainStem2CLI::PowerDeliveryClass::request (*C++ function*), 487  
Acroname::BrainStem2CLI::PowerDeliveryClass::requestStatus (*C++ function*), 488  
Acroname::BrainStem2CLI::PowerDeliveryClass::resetEntityToFactoryDefaults (*C++ function*), 488  
Acroname::BrainStem2CLI::PowerDeliveryClass::resetPowerDataObjectToDefault (*C++ function*), 484  
Acroname::BrainStem2CLI::PowerDeliveryClass::setFastRoleSwapCurrent (*C++ function*), 489  
Acroname::BrainStem2CLI::PowerDeliveryClass::setFlagMode (*C++ function*), 488  
Acroname::BrainStem2CLI::PowerDeliveryClass::setOverride (*C++ function*), 488

Acroname::BrainStem2CLI::PowerDeliveryClass::setPeakCurrentConfiguration (*C++ function*), 489  
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerDataObject (*C++ function*), 483  
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerDataObjectEnabled (*C++ function*), 484  
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerRole (*C++ function*), 486  
 Acroname::BrainStem2CLI::PowerDeliveryClass::setPowerRolePreferred (*C++ function*), 486  
 Acroname::BrainStem2CLI::PowerDeliveryClass::setRequestDataObject (*C++ function*), 485  
 Acroname::BrainStem2CLI::RailClass (*C++ class*), 490  
 Acroname::BrainStem2CLI::RailClass::~RailClass (*C++ function*), 490  
 Acroname::BrainStem2CLI::RailClass::clearFaults (*C++ function*), 495  
 Acroname::BrainStem2CLI::RailClass::getCurrent (*C++ function*), 490  
 Acroname::BrainStem2CLI::RailClass::getCurrentLimit (*C++ function*), 490  
 Acroname::BrainStem2CLI::RailClass::getCurrentSetpoint (*C++ function*), 490  
 Acroname::BrainStem2CLI::RailClass::getEnable (*C++ function*), 491  
 Acroname::BrainStem2CLI::RailClass::getKelvinSensingEnable (*C++ function*), 494  
 Acroname::BrainStem2CLI::RailClass::getKelvinSensingState (*C++ function*), 494  
 Acroname::BrainStem2CLI::RailClass::getOperationalMode (*C++ function*), 494  
 Acroname::BrainStem2CLI::RailClass::getOperationalState (*C++ function*), 494  
 Acroname::BrainStem2CLI::RailClass::getPower (*C++ function*), 492  
 Acroname::BrainStem2CLI::RailClass::getPowerLimit (*C++ function*), 493  
 Acroname::BrainStem2CLI::RailClass::getPowerSetpoint (*C++ function*), 493  
 Acroname::BrainStem2CLI::RailClass::getResistance (*C++ function*), 493  
 Acroname::BrainStem2CLI::RailClass::getResistanceSetpoint (*C++ function*), 493  
 Acroname::BrainStem2CLI::RailClass::getTemperature (*C++ function*), 491  
 Acroname::BrainStem2CLI::RailClass::getVoltage (*C++ function*), 491  
 Acroname::BrainStem2CLI::RailClass::getVoltageMaxLimit (*C++ function*), 492  
 Acroname::BrainStem2CLI::RailClass::getVoltageMinLimit (*C++ function*), 492  
 Acroname::BrainStem2CLI::RailClass::getVoltageSetpoint (*C++ function*), 491  
 Acroname::BrainStem2CLI::RailClass::RailClass (*C++ function*), 490  
 Acroname::BrainStem2CLI::RailClass::setCurrentLimit (*C++ function*), 490  
 Acroname::BrainStem2CLI::RailClass::setCurrentSetpoint (*C++ function*), 490  
 Acroname::BrainStem2CLI::RailClass::setEnable (*C++ function*), 491  
 Acroname::BrainStem2CLI::RailClass::setKelvinSensingEnable (*C++ function*), 494  
 Acroname::BrainStem2CLI::RailClass::setOperationalMode (*C++ function*), 494  
 Acroname::BrainStem2CLI::RailClass::setPowerLimit (*C++ function*), 493  
 Acroname::BrainStem2CLI::RailClass::setPowerSetpoint (*C++ function*), 492  
 Acroname::BrainStem2CLI::RailClass::setResistanceSetpoint (*C++ function*), 493  
 Acroname::BrainStem2CLI::RailClass::setVoltageMaxLimit (*C++ function*), 492  
 Acroname::BrainStem2CLI::RailClass::setVoltageMinLimit (*C++ function*), 492  
 Acroname::BrainStem2CLI::RailClass::setVoltageSetpoint (*C++ function*), 491  
 Acroname::BrainStem2CLI::RCServoClass (*C++ class*), 495  
 Acroname::BrainStem2CLI::RCServoClass::~RCServoClass (*C++ function*), 495  
 Acroname::BrainStem2CLI::RCServoClass::getEnable (*C++ function*), 495  
 Acroname::BrainStem2CLI::RCServoClass::getPosition (*C++ function*), 495  
 Acroname::BrainStem2CLI::RCServoClass::getReverse (*C++ function*), 496  
 Acroname::BrainStem2CLI::RCServoClass::RCServoClass (*C++ function*), 495  
 Acroname::BrainStem2CLI::RCServoClass::setEnabled (*C++ function*), 495  
 Acroname::BrainStem2CLI::RCServoClass::setPosition (*C++ function*), 495  
 Acroname::BrainStem2CLI::RCServoClass::setReverse (*C++ function*), 496  
 Acroname::BrainStem2CLI::RelayClass (*C++ class*), 496  
 Acroname::BrainStem2CLI::RelayClass::~RelayClass (*C++ function*), 496  
 Acroname::BrainStem2CLI::RelayClass::getEnable (*C++ function*), 496  
 Acroname::BrainStem2CLI::RelayClass::getVoltage (*C++ function*), 497  
 Acroname::BrainStem2CLI::RelayClass::RelayClass (*C++ function*), 496  
 Acroname::BrainStem2CLI::RelayClass::setEnabled (*C++ function*), 496  
 Acroname::BrainStem2CLI::SignalClass (*C++ class*), 497  
 Acroname::BrainStem2CLI::SignalClass::~SignalClass (*C++ function*), 497  
 Acroname::BrainStem2CLI::SignalClass::getEnable (*C++ function*), 497  
 Acroname::BrainStem2CLI::SignalClass::getInvert (*C++ function*), 498  
 Acroname::BrainStem2CLI::SignalClass::getT2Time (*C++ function*), 498  
 Acroname::BrainStem2CLI::SignalClass::getT3Time (*C++ function*), 498  
 Acroname::BrainStem2CLI::SignalClass::setEnabled (*C++ function*), 497  
 Acroname::BrainStem2CLI::SignalClass::setInvert (*C++ function*), 498  
 Acroname::BrainStem2CLI::SignalClass::setT2Time (*C++ function*), 498  
 Acroname::BrainStem2CLI::SignalClass::setT3Time (*C++ function*), 498  
 Acroname::BrainStem2CLI::SignalClass::SignalClass (*C++ function*), 497  
 Acroname::BrainStem2CLI::StoreClass (*C++ class*), 499  
 Acroname::BrainStem2CLI::StoreClass::~StoreClass (*C++ function*), 499  
 Acroname::BrainStem2CLI::StoreClass::getSlotCapacity (*C++ function*), 500

Acroname::BrainStem2CLI::StoreClass::getSlotSize (*C++ function*), 500  
Acroname::BrainStem2CLI::StoreClass::getSlotState (*C++ function*), 499  
Acroname::BrainStem2CLI::StoreClass::loadSlot (*C++ function*), 499  
Acroname::BrainStem2CLI::StoreClass::slotDisable (*C++ function*), 500  
Acroname::BrainStem2CLI::StoreClass::slotEnable (*C++ function*), 500  
Acroname::BrainStem2CLI::StoreClass::StoreClass (*C++ function*), 499  
Acroname::BrainStem2CLI::StoreClass::unloadSlot (*C++ function*), 499  
Acroname::BrainStem2CLI::SystemClass (*C++ class*), 500  
Acroname::BrainStem2CLI::SystemClass::~SystemClass (*C++ function*), 501  
Acroname::BrainStem2CLI::SystemClass::getBootSlot (*C++ function*), 502  
Acroname::BrainStem2CLI::SystemClass::getErrors (*C++ function*), 505  
Acroname::BrainStem2CLI::SystemClass::getHardwareVersion (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::getHBInterval (*C++ function*), 502  
Acroname::BrainStem2CLI::SystemClass::getInputCurrent (*C++ function*), 504  
Acroname::BrainStem2CLI::SystemClass::getInputVoltage (*C++ function*), 504  
Acroname::BrainStem2CLI::SystemClass::getLED (*C++ function*), 502  
Acroname::BrainStem2CLI::SystemClass::getMaximumTemperature (*C++ function*), 504  
Acroname::BrainStem2CLI::SystemClass::getMinimumTemperature (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::getModel (*C++ function*), 502  
Acroname::BrainStem2CLI::SystemClass::getModule (*C++ function*), 501  
Acroname::BrainStem2CLI::SystemClass::getModuleBaseAddress (*C++ function*), 501  
Acroname::BrainStem2CLI::SystemClass::getModuleHardwareOffset (*C++ function*), 504  
Acroname::BrainStem2CLI::SystemClass::getModuleSoftwareOffset (*C++ function*), 504  
Acroname::BrainStem2CLI::SystemClass::getRouter (*C++ function*), 501  
Acroname::BrainStem2CLI::SystemClass::getRouterAddressSetting (*C++ function*), 504  
Acroname::BrainStem2CLI::SystemClass::getSerialNumber (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::getTemperature (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::getUptime (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::getVersion (*C++ function*), 502  
Acroname::BrainStem2CLI::SystemClass::logEvents (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::reset (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::routeToMe (*C++ function*), 505  
Acroname::BrainStem2CLI::SystemClass::save (*C++ function*), 503  
Acroname::BrainStem2CLI::SystemClass::setBootSlot (*C++ function*), 502  
Acroname::BrainStem2CLI::SystemClass::setHBInterval (*C++ function*), 501  
Acroname::BrainStem2CLI::SystemClass::setLED (*C++ function*), 502  
Acroname::BrainStem2CLI::SystemClass::setModuleSoftwareOffset (*C++ function*), 504  
Acroname::BrainStem2CLI::SystemClass::setRouter (*C++ function*), 501  
Acroname::BrainStem2CLI::SystemClass::SystemClass (*C++ function*), 501  
Acroname::BrainStem2CLI::TemperatureClass (*C++ class*), 505  
Acroname::BrainStem2CLI::TemperatureClass::~TemperatureClass (*C++ function*), 505  
Acroname::BrainStem2CLI::TemperatureClass::getValue (*C++ function*), 505  
Acroname::BrainStem2CLI::TemperatureClass::getValueMax (*C++ function*), 506  
Acroname::BrainStem2CLI::TemperatureClass::getValueMin (*C++ function*), 505  
Acroname::BrainStem2CLI::TemperatureClass::TemperatureClass (*C++ function*), 505  
Acroname::BrainStem2CLI::TimerClass (*C++ class*), 506  
Acroname::BrainStem2CLI::TimerClass::~TimerClass (*C++ function*), 506  
Acroname::BrainStem2CLI::TimerClass::getExpiration (*C++ function*), 506  
Acroname::BrainStem2CLI::TimerClass::getMode (*C++ function*), 506  
Acroname::BrainStem2CLI::TimerClass::setExpiration (*C++ function*), 506  
Acroname::BrainStem2CLI::TimerClass::setMode (*C++ function*), 507  
Acroname::BrainStem2CLI::TimerClass::TimerClass (*C++ function*), 506  
Acroname::BrainStem2CLI::UARTClass (*C++ class*), 507  
Acroname::BrainStem2CLI::UARTClass::~UARTClass (*C++ function*), 507  
Acroname::BrainStem2CLI::UARTClass::getEnable (*C++ function*), 507  
Acroname::BrainStem2CLI::UARTClass::setEnable (*C++ function*), 507  
Acroname::BrainStem2CLI::UARTClass::UARTClass (*C++ function*), 507  
Acroname::BrainStem2CLI::UARTClass::getBaudRate (*C++ function*), 508  
Acroname::BrainStem2CLI::UARTClass::getProtocol (*C++ function*), 508  
Acroname::BrainStem2CLI::UARTClass::setBaudRate (*C++ function*), 507  
Acroname::BrainStem2CLI::UARTClass::setProtocol (*C++ function*), 508  
Acroname::BrainStem2CLI::USBClass (*C++ class*), 508  
Acroname::BrainStem2CLI::USBClass::~USBClass (*C++ function*), 508  
Acroname::BrainStem2CLI::USBClass::clearPortErrorStatus (*C++ function*), 510  
Acroname::BrainStem2CLI::USBClass::getAltModeConfig (*C++ function*), 516  
Acroname::BrainStem2CLI::USBClass::getCableFlip (*C++ function*), 515  
Acroname::BrainStem2CLI::USBClass::getCC1Current (*C++ function*), 514  
Acroname::BrainStem2CLI::USBClass::getCC1Enable (*C++ function*), 514

Acroname:::BrainStem2CLI:::USBClass:::getCC1Voltage (*C++ function*), 515  
 Acroname:::BrainStem2CLI:::USBClass:::getCC2Current (*C++ function*), 514  
 Acroname:::BrainStem2CLI:::USBClass:::getCC2Enable (*C++ function*), 514  
 Acroname:::BrainStem2CLI:::USBClass:::getCC2Voltage (*C++ function*), 515  
 Acroname:::BrainStem2CLI:::USBClass:::getConnectMode (*C++ function*), 513  
 Acroname:::BrainStem2CLI:::USBClass:::getDownstreamBoostMode (*C++ function*), 513  
 Acroname:::BrainStem2CLI:::USBClass:::getDownstreamDataSpeed (*C++ function*), 513  
 Acroname:::BrainStem2CLI:::USBClass:::getEnumerationDelay (*C++ function*), 511  
 Acroname:::BrainStem2CLI:::USBClass:::getHubMode (*C++ function*), 510  
 Acroname:::BrainStem2CLI:::USBClass:::getPortCurrent (*C++ function*), 510  
 Acroname:::BrainStem2CLI:::USBClass:::getPortCurrentLimit (*C++ function*), 511  
 Acroname:::BrainStem2CLI:::USBClass:::getPortError (*C++ function*), 512  
 Acroname:::BrainStem2CLI:::USBClass:::getPortMode (*C++ function*), 512  
 Acroname:::BrainStem2CLI:::USBClass:::getPortState (*C++ function*), 512  
 Acroname:::BrainStem2CLI:::USBClass:::getPortVoltage (*C++ function*), 510  
 Acroname:::BrainStem2CLI:::USBClass:::getSBU1Voltage (*C++ function*), 516  
 Acroname:::BrainStem2CLI:::USBClass:::getSBU2Voltage (*C++ function*), 516  
 Acroname:::BrainStem2CLI:::USBClass:::getSBUEnable (*C++ function*), 515  
 Acroname:::BrainStem2CLI:::USBClass:::getUpstreamBoostMode (*C++ function*), 513  
 Acroname:::BrainStem2CLI:::USBClass:::getUpstreamMode (*C++ function*), 510  
 Acroname:::BrainStem2CLI:::USBClass:::getUpstreamState (*C++ function*), 511  
 Acroname:::BrainStem2CLI:::USBClass:::setAltModeConfig (*C++ function*), 516  
 Acroname:::BrainStem2CLI:::USBClass:::setCableFlip (*C++ function*), 515  
 Acroname:::BrainStem2CLI:::USBClass:::setCC1Enable (*C++ function*), 514  
 Acroname:::BrainStem2CLI:::USBClass:::setCC2Enable (*C++ function*), 514  
 Acroname:::BrainStem2CLI:::USBClass:::setConnectMode (*C++ function*), 513  
 Acroname:::BrainStem2CLI:::USBClass:::setDataDisable (*C++ function*), 509  
 Acroname:::BrainStem2CLI:::USBClass:::setDataEnable (*C++ function*), 509  
 Acroname:::BrainStem2CLI:::USBClass:::setDownstreamBoostMode (*C++ function*), 512  
 Acroname:::BrainStem2CLI:::USBClass:::setEnumerationDelay (*C++ function*), 511  
 Acroname:::BrainStem2CLI:::USBClass:::setHiSpeedDataDisable (*C++ function*), 509  
 Acroname:::BrainStem2CLI:::USBClass:::setHiSpeedDataEnable (*C++ function*), 509  
 Acroname:::BrainStem2CLI:::USBClass:::setHubMode (*C++ function*), 510  
 Acroname:::BrainStem2CLI:::USBClass:::setPortCurrentLimit (*C++ function*), 511  
 Acroname:::BrainStem2CLI:::USBClass:::setPortDisable (*C++ function*), 508  
 Acroname:::BrainStem2CLI:::USBClass:::setPortEnable (*C++ function*), 508  
 Acroname:::BrainStem2CLI:::USBClass:::setPortMode (*C++ function*), 511  
 Acroname:::BrainStem2CLI:::USBClass:::setPowerDisable (*C++ function*), 510  
 Acroname:::BrainStem2CLI:::USBClass:::setPowerEnable (*C++ function*), 509  
 Acroname:::BrainStem2CLI:::USBClass:::setSBUEnable (*C++ function*), 515  
 Acroname:::BrainStem2CLI:::USBClass:::setSuperSpeedDataDisable (*C++ function*), 509  
 Acroname:::BrainStem2CLI:::USBClass:::setSuperSpeedDataEnable (*C++ function*), 509  
 Acroname:::BrainStem2CLI:::USBClass:::setUpstreamBoostMode (*C++ function*), 512  
 Acroname:::BrainStem2CLI:::USBClass:::setUpstreamMode (*C++ function*), 511  
 Acroname:::BrainStem2CLI:::USBClass (*C++ function*), 508  
 Acroname:::BrainStem2CLI:::USBSystemClass (*C++ class*), 516  
 Acroname:::BrainStem2CLI:::USBSystemClass:::~USBSystemClass (*C++ function*), 516  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getDataRoleBehavior (*C++ function*), 519  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getDataRoleBehaviorConfig (*C++ function*), 519  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getDataRoleList (*C++ function*), 517  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getEnabledList (*C++ function*), 517  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getEnumerationDelay (*C++ function*), 517  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getModeList (*C++ function*), 517  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getPowerBehavior (*C++ function*), 518  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getPowerBehaviorConfig (*C++ function*), 518  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getStateList (*C++ function*), 518  
 Acroname:::BrainStem2CLI:::USBSystemClass:::getUpstream (*C++ function*), 516  
 Acroname:::BrainStem2CLI:::USBSystemClass:::resetEntityToFactoryDefaults (*C++ function*), 519  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setDataRoleBehavior (*C++ function*), 519  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setDataRoleBehaviorConfig (*C++ function*), 519  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setEnabledList (*C++ function*), 517  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setEnumerationDelay (*C++ function*), 517  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setModeList (*C++ function*), 518  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setPowerBehavior (*C++ function*), 518  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setPowerBehaviorConfig (*C++ function*), 519  
 Acroname:::BrainStem2CLI:::USBSystemClass:::setUpstream (*C++ function*), 517  
 Acroname:::BrainStem2CLI:::USBSystemClass:::USBSysClass (*C++ function*), 516  
 Acroname:::BrainStem:::AnalogClass (*C++ class*), 298

Acroname::BrainStem::AnalogClass::~AnalogClass (*C++ function*), 298  
Acroname::BrainStem::AnalogClass::AnalogClass (*C++ function*), 298  
Acroname::BrainStem::AnalogClass::getBulkCaptureNumberOfSamples (*C++ function*), 300  
Acroname::BrainStem::AnalogClass::getBulkCaptureSampleRate (*C++ function*), 300  
Acroname::BrainStem::AnalogClass::getBulkCaptureState (*C++ function*), 301  
Acroname::BrainStem::AnalogClass::getConfiguration (*C++ function*), 300  
Acroname::BrainStem::AnalogClass::getEnable (*C++ function*), 299  
Acroname::BrainStem::AnalogClass::getRange (*C++ function*), 299  
Acroname::BrainStem::AnalogClass::getValue (*C++ function*), 298  
Acroname::BrainStem::AnalogClass::getVoltage (*C++ function*), 298  
Acroname::BrainStem::AnalogClass::init (*C++ function*), 298  
Acroname::BrainStem::AnalogClass::initiateBulkCapture (*C++ function*), 300  
Acroname::BrainStem::AnalogClass::setBulkCaptureNumberOfSamples (*C++ function*), 300  
Acroname::BrainStem::AnalogClass::setBulkCaptureSampleRate (*C++ function*), 300  
Acroname::BrainStem::AnalogClass::setConfiguration (*C++ function*), 300  
Acroname::BrainStem::AnalogClass::setEnable (*C++ function*), 299  
Acroname::BrainStem::AnalogClass::setRange (*C++ function*), 299  
Acroname::BrainStem::AnalogClass::setValue (*C++ function*), 299  
Acroname::BrainStem::AnalogClass::setVoltage (*C++ function*), 299  
Acroname::BrainStem::AppClass (*C++ class*), 301  
Acroname::BrainStem::AppClass::~AppClass (*C++ function*), 301  
Acroname::BrainStem::AppClass::AppClass (*C++ function*), 301  
Acroname::BrainStem::AppClass::execute (*C++ function*), 301, 302  
Acroname::BrainStem::AppClass::init (*C++ function*), 301  
Acroname::BrainStem::ClockClass (*C++ class*), 302  
Acroname::BrainStem::ClockClass::~ClockClass (*C++ function*), 302  
Acroname::BrainStem::ClockClass::ClockClass (*C++ function*), 302  
Acroname::BrainStem::ClockClass::getDay (*C++ function*), 303  
Acroname::BrainStem::ClockClass::getHour (*C++ function*), 303  
Acroname::BrainStem::ClockClass::getMinute (*C++ function*), 303  
Acroname::BrainStem::ClockClass::getMonth (*C++ function*), 303  
Acroname::BrainStem::ClockClass::getSecond (*C++ function*), 304  
Acroname::BrainStem::ClockClass::getYear (*C++ function*), 302  
Acroname::BrainStem::ClockClass::init (*C++ function*), 302  
Acroname::BrainStem::ClockClass::setDay (*C++ function*), 303  
Acroname::BrainStem::ClockClass::setHour (*C++ function*), 303  
Acroname::BrainStem::ClockClass::setMinute (*C++ function*), 303  
Acroname::BrainStem::ClockClass::setMonth (*C++ function*), 303  
Acroname::BrainStem::ClockClass::setSecond (*C++ function*), 304  
Acroname::BrainStem::ClockClass::setYear (*C++ function*), 302  
Acroname::BrainStem::DigitalClass (*C++ class*), 304  
Acroname::BrainStem::DigitalClass::~DigitalClass (*C++ function*), 304  
Acroname::BrainStem::DigitalClass::DigitalClass (*C++ function*), 304  
Acroname::BrainStem::DigitalClass::getConfiguration (*C++ function*), 305  
Acroname::BrainStem::DigitalClass::getState (*C++ function*), 305  
Acroname::BrainStem::DigitalClass::getStateAll (*C++ function*), 305  
Acroname::BrainStem::DigitalClass::init (*C++ function*), 304  
Acroname::BrainStem::DigitalClass::setConfiguration (*C++ function*), 304  
Acroname::BrainStem::DigitalClass::setState (*C++ function*), 305  
Acroname::BrainStem::DigitalClass::setStateAll (*C++ function*), 305  
Acroname::BrainStem::EntityClass (*C++ class*), 306  
Acroname::BrainStem::EntityClass::~EntityClass (*C++ function*), 306  
Acroname::BrainStem::EntityClass::callUEI (*C++ function*), 306  
Acroname::BrainStem::EntityClass::drainUEI (*C++ function*), 308  
Acroname::BrainStem::EntityClass::EntityClass (*C++ function*), 306  
Acroname::BrainStem::EntityClass::getIndex (*C++ function*), 308  
Acroname::BrainStem::EntityClass::getStreamStatus (*C++ function*), 309  
Acroname::BrainStem::EntityClass::getUEI16 (*C++ function*), 307  
Acroname::BrainStem::EntityClass::getUEI32 (*C++ function*), 308  
Acroname::BrainStem::EntityClass::getUEI8 (*C++ function*), 307  
Acroname::BrainStem::EntityClass::getUEIBytes32 (*C++ function*), 308  
Acroname::BrainStem::EntityClass::getUEIBytes8 (*C++ function*), 308  
Acroname::BrainStem::EntityClass::getUEIBytesContinue (*C++ function*), 309  
Acroname::BrainStem::EntityClass::getUEIBytesSequence (*C++ function*), 309  
Acroname::BrainStem::EntityClass::init (*C++ function*), 306  
Acroname::BrainStem::EntityClass::registerOptionCallback (*C++ function*), 309  
Acroname::BrainStem::EntityClass::setStreamEnabled (*C++ function*), 308  
Acroname::BrainStem::EntityClass::setUEI16 (*C++ function*), 307

Acroname::BrainStem::EntityClass::setUEI32 (*C++ function*), 307  
 Acroname::BrainStem::EntityClass::setUEI8 (*C++ function*), 306  
 Acroname::BrainStem::EntityClass::sUEIBytesFilter (*C++ function*), 309  
 Acroname::BrainStem::EqualizerClass (*C++ class*), 310  
 Acroname::BrainStem::EqualizerClass::~EqualizerClass (*C++ function*), 310  
 Acroname::BrainStem::EqualizerClass::EqualizerClass (*C++ function*), 310  
 Acroname::BrainStem::EqualizerClass::getReceiverConfig (*C++ function*), 310  
 Acroname::BrainStem::EqualizerClass::getTransmitterConfig (*C++ function*), 310  
 Acroname::BrainStem::EqualizerClass::init (*C++ function*), 310  
 Acroname::BrainStem::EqualizerClass::setReceiverConfig (*C++ function*), 310  
 Acroname::BrainStem::EqualizerClass::setTransmitterConfig (*C++ function*), 310  
 Acroname::BrainStem::I2CClass (*C++ class*), 311  
 Acroname::BrainStem::I2CClass::~I2CClass (*C++ function*), 311  
 Acroname::BrainStem::I2CClass::getSpeed (*C++ function*), 312  
 Acroname::BrainStem::I2CClass::I2CClass (*C++ function*), 311  
 Acroname::BrainStem::I2CClass::init (*C++ function*), 311  
 Acroname::BrainStem::I2CClass::read (*C++ function*), 311  
 Acroname::BrainStem::I2CClass::setPullup (*C++ function*), 311  
 Acroname::BrainStem::I2CClass::setSpeed (*C++ function*), 312  
 Acroname::BrainStem::I2CClass::write (*C++ function*), 311  
 Acroname::BrainStem::Link (*C++ class*), 312  
 Acroname::BrainStem::Link::~Link (*C++ function*), 314  
 Acroname::BrainStem::Link::connect (*C++ function*), 314  
 Acroname::BrainStem::Link::disablePacketLog (*C++ function*), 321  
 Acroname::BrainStem::Link::disconnect (*C++ function*), 315  
 Acroname::BrainStem::Link::discoverAndConnect (*C++ function*), 314  
 Acroname::BrainStem::Link::dropMatchingUEIPackets (*C++ function*), 317  
 Acroname::BrainStem::Link::enablePacketLog (*C++ function*), 321  
 Acroname::BrainStem::Link::enableStream (*C++ function*), 320  
 Acroname::BrainStem::Link::filterActiveStreamKeys (*C++ function*), 321  
 Acroname::BrainStem::Link::getFactoryData (*C++ function*), 321  
 Acroname::BrainStem::Link::getLinkSpecifier (*C++ function*), 315  
 Acroname::BrainStem::Link::getModuleAddress (*C++ function*), 315  
 Acroname::BrainStem::Link::getName (*C++ function*), 315  
 Acroname::BrainStem::Link::getStatus (*C++ function*), 315  
 Acroname::BrainStem::Link::getStreamKeyElement (*C++ function*), 323  
 Acroname::BrainStem::Link::getStreamPacketType (*C++ function*), 322  
 Acroname::BrainStem::Link::getStreamSample (*C++ function*), 323  
 Acroname::BrainStem::Link::getStreamStatus (*C++ function*), 321  
 Acroname::BrainStem::Link::getStreamValue (*C++ function*), 320  
 Acroname::BrainStem::Link::getTimestampParts (*C++ function*), 323  
 Acroname::BrainStem::Link::isConnected (*C++ function*), 315  
 Acroname::BrainStem::Link::isLinkStreaming (*C++ function*), 320  
 Acroname::BrainStem::Link::isStreamPacket (*C++ function*), 323  
 Acroname::BrainStem::Link::isStreamSample (*C++ function*), 323  
 Acroname::BrainStem::Link::isSubindexType (*C++ function*), 323  
 Acroname::BrainStem::Link::Link (*C++ function*), 314  
 Acroname::BrainStem::Link::linkStreamFilter (*C++ function*), 324  
 Acroname::BrainStem::Link::loadStoreSlot (*C++ function*), 318  
 Acroname::BrainStem::Link::receivePacket (*C++ function*), 317  
 Acroname::BrainStem::Link::receiveUEI (*C++ function*), 316  
 Acroname::BrainStem::Link::registerStreamCallback (*C++ function*), 320  
 Acroname::BrainStem::Link::reset (*C++ function*), 315  
 Acroname::BrainStem::Link::sDiscover (*C++ function*), 322  
 Acroname::BrainStem::Link::sendPacket (*C++ function*), 317  
 Acroname::BrainStem::Link::sendUEI (*C++ function*), 315, 316  
 Acroname::BrainStem::Link::setFactoryData (*C++ function*), 321  
 Acroname::BrainStem::Link::setLinkSpecifier (*C++ function*), 315  
 Acroname::BrainStem::Link::sFindAll (*C++ function*), 322  
 Acroname::BrainStem::Link::storeSlotCapacity (*C++ function*), 319  
 Acroname::BrainStem::Link::storeSlotSize (*C++ function*), 319  
 Acroname::BrainStem::Link::STREAM\_KEY (*C++ enum*), 313  
 Acroname::BrainStem::Link::STREAM\_KEY::STREAM\_KEY\_CMD (*C++ enumerator*), 313  
 Acroname::BrainStem::Link::STREAM\_KEY::STREAM\_KEY\_INDEX (*C++ enumerator*), 313  
 Acroname::BrainStem::Link::STREAM\_KEY::STREAM\_KEY\_MODULE\_ADDRESS (*C++ enumerator*), 313  
 Acroname::BrainStem::Link::STREAM\_KEY::STREAM\_KEY\_OPTION (*C++ enumerator*), 313  
 Acroname::BrainStem::Link::STREAM\_KEY::STREAM\_KEY\_SUBINDEX (*C++ enumerator*), 313  
 Acroname::BrainStem::Link::STREAM\_KEY\_t (*C++ type*), 313

Acroname::BrainStem::Link::STREAM\_PACKET (*C++ enum*), 312  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_BYTES (*C++ enumerator*), 313  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_LAST (*C++ enumerator*), 313  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_SUBINDEX\_U16 (*C++ enumerator*), 313  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_SUBINDEX\_U32 (*C++ enumerator*), 313  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_SUBINDEX\_U8 (*C++ enumerator*), 313  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_U16 (*C++ enumerator*), 312  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_U32 (*C++ enumerator*), 312  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_U8 (*C++ enumerator*), 312  
Acroname::BrainStem::Link::STREAM\_PACKET::kSTREAM\_PACKET\_UNKNOWN (*C++ enumerator*), 312  
Acroname::BrainStem::Link::STREAM\_PACKET\_t (*C++ type*), 313  
Acroname::BrainStem::Link::streamCallback\_t (*C++ type*), 313  
Acroname::BrainStem::Link::unloadStoreSlot (*C++ function*), 318  
Acroname::BrainStem::Module (*C++ class*), 324  
Acroname::BrainStem::Module::~Module (*C++ function*), 324  
Acroname::BrainStem::Module::classQuantity (*C++ function*), 327  
Acroname::BrainStem::Module::connect (*C++ function*), 324  
Acroname::BrainStem::Module::connectFromSpec (*C++ function*), 325  
Acroname::BrainStem::Module::connectThroughLinkModule (*C++ function*), 325  
Acroname::BrainStem::Module::debug (*C++ function*), 328  
Acroname::BrainStem::Module::disconnect (*C++ function*), 326  
Acroname::BrainStem::Module::discoverAndConnect (*C++ function*), 325  
Acroname::BrainStem::Module::entityGroup (*C++ function*), 327  
Acroname::BrainStem::Module::getLink (*C++ function*), 326  
Acroname::BrainStem::Module::getLinkSpecifier (*C++ function*), 326  
Acroname::BrainStem::Module::getModuleAddress (*C++ function*), 326  
Acroname::BrainStem::Module::getStatus (*C++ function*), 325  
Acroname::BrainStem::Module::hasUEI (*C++ function*), 326  
Acroname::BrainStem::Module::isConnected (*C++ function*), 325  
Acroname::BrainStem::Module (*C++ function*), 324  
Acroname::BrainStem::Module::reconnect (*C++ function*), 326  
Acroname::BrainStem::Module::setModuleAddress (*C++ function*), 326  
Acroname::BrainStem::Module::setNetworkingMode (*C++ function*), 328  
Acroname::BrainStem::Module::subClassQuantity (*C++ function*), 327  
Acroname::BrainStem::MuxClass (*C++ class*), 328  
Acroname::BrainStem::~MuxClass (*C++ function*), 329  
Acroname::BrainStem::MuxClass::getChannel (*C++ function*), 329  
Acroname::BrainStem::MuxClass::getChannelVoltage (*C++ function*), 329  
Acroname::BrainStem::MuxClass::getConfiguration (*C++ function*), 329  
Acroname::BrainStem::MuxClass::getEnable (*C++ function*), 329  
Acroname::BrainStem::MuxClass::getSplitMode (*C++ function*), 330  
Acroname::BrainStem::MuxClass::init (*C++ function*), 329  
Acroname::BrainStem::MuxClass::MuxClass (*C++ function*), 329  
Acroname::BrainStem::MuxClass::setChannel (*C++ function*), 329  
Acroname::BrainStem::MuxClass::setConfiguration (*C++ function*), 330  
Acroname::BrainStem::MuxClass::setEnable (*C++ function*), 329  
Acroname::BrainStem::MuxClass::setSplitMode (*C++ function*), 330  
Acroname::BrainStem::PointerClass (*C++ class*), 330  
Acroname::BrainStem::PointerClass::~PointerClass (*C++ function*), 331  
Acroname::BrainStem::PointerClass::getChar (*C++ function*), 332  
Acroname::BrainStem::PointerClass::getInt (*C++ function*), 332  
Acroname::BrainStem::PointerClass::getMode (*C++ function*), 331  
Acroname::BrainStem::PointerClass::getOffset (*C++ function*), 331  
Acroname::BrainStem::PointerClass::getShort (*C++ function*), 332  
Acroname::BrainStem::PointerClass::getTransferStore (*C++ function*), 331  
Acroname::BrainStem::PointerClass::init (*C++ function*), 331  
Acroname::BrainStem::PointerClass::initiateTransferFromStore (*C++ function*), 332  
Acroname::BrainStem::PointerClass::initiateTransferToStore (*C++ function*), 332  
Acroname::BrainStem::PointerClass::PointerClass (*C++ function*), 331  
Acroname::BrainStem::PointerClass::setChar (*C++ function*), 332  
Acroname::BrainStem::PointerClass::setInt (*C++ function*), 332  
Acroname::BrainStem::PointerClass::setMode (*C++ function*), 331  
Acroname::BrainStem::PointerClass::setOffset (*C++ function*), 331  
Acroname::BrainStem::PointerClass::setShort (*C++ function*), 332  
Acroname::BrainStem::PointerClass::setTransferStore (*C++ function*), 331  
Acroname::BrainStem::PortClass (*C++ class*), 333  
Acroname::BrainStem::~PortClass (*C++ function*), 333  
Acroname::BrainStem::PortClass::getAllocatedPower (*C++ function*), 340

Acroname::BrainStem::PortClass::getAvailablePower (*C++ function*), 339  
 Acroname::BrainStem::PortClass::getCC1Enabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::getCC2Enabled (*C++ function*), 338  
 Acroname::BrainStem::PortClass::getCCEnabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::getCurrentLimit (*C++ function*), 339  
 Acroname::BrainStem::PortClass::getCurrentLimitMode (*C++ function*), 339  
 Acroname::BrainStem::PortClass::getDataEnabled (*C++ function*), 334  
 Acroname::BrainStem::PortClass::getDataHS1Enabled (*C++ function*), 335  
 Acroname::BrainStem::PortClass::getDataHS2Enabled (*C++ function*), 335  
 Acroname::BrainStem::PortClass::getDataHSEnabled (*C++ function*), 334  
 Acroname::BrainStem::PortClass::getDataHSRoutingBehavior (*C++ function*), 341  
 Acroname::BrainStem::PortClass::getDataRole (*C++ function*), 336  
 Acroname::BrainStem::PortClass::getDataSpeed (*C++ function*), 338  
 Acroname::BrainStem::PortClass::getDataSS1Enabled (*C++ function*), 335  
 Acroname::BrainStem::PortClass::getDataSS2Enabled (*C++ function*), 336  
 Acroname::BrainStem::PortClass::getDataSSEnabled (*C++ function*), 335  
 Acroname::BrainStem::PortClass::getDataSSRoutingBehavior (*C++ function*), 341  
 Acroname::BrainStem::PortClass::getEnabled (*C++ function*), 334  
 Acroname::BrainStem::PortClass::getErrors (*C++ function*), 339  
 Acroname::BrainStem::PortClass::getHSBoost (*C++ function*), 342  
 Acroname::BrainStem::PortClass::getMode (*C++ function*), 338  
 Acroname::BrainStem::PortClass::getName (*C++ function*), 340  
 Acroname::BrainStem::PortClass::getPowerEnabled (*C++ function*), 336  
 Acroname::BrainStem::PortClass::getPowerLimit (*C++ function*), 340  
 Acroname::BrainStem::PortClass::getPowerLimitMode (*C++ function*), 340  
 Acroname::BrainStem::PortClass::getPowerMode (*C++ function*), 334  
 Acroname::BrainStem::PortClass::getState (*C++ function*), 338  
 Acroname::BrainStem::PortClass::getVbusAccumulatedPower (*C++ function*), 341  
 Acroname::BrainStem::PortClass::getVbusCurrent (*C++ function*), 333  
 Acroname::BrainStem::PortClass::getVbusVoltage (*C++ function*), 333  
 Acroname::BrainStem::PortClass::getVconn1Enabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::getVconn2Enabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::getVconnAccumulatedPower (*C++ function*), 342  
 Acroname::BrainStem::PortClass::getVconnCurrent (*C++ function*), 333  
 Acroname::BrainStem::PortClass::getVconnEnabled (*C++ function*), 336  
 Acroname::BrainStem::PortClass::getVconnVoltage (*C++ function*), 333  
 Acroname::BrainStem::PortClass::getVoltageSetpoint (*C++ function*), 338  
 Acroname::BrainStem::PortClass (*C++ function*), 333  
 Acroname::BrainStem::PortClass::resetEntityToFactoryDefaults (*C++ function*), 342  
 Acroname::BrainStem::PortClass::resetVbusAccumulatedPower (*C++ function*), 342  
 Acroname::BrainStem::PortClass::resetVconnAccumulatedPower (*C++ function*), 342  
 Acroname::BrainStem::PortClass::setCC1Enabled (*C++ function*), 338  
 Acroname::BrainStem::PortClass::setCC2Enabled (*C++ function*), 338  
 Acroname::BrainStem::PortClass::setCCEnabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::setCurrentLimit (*C++ function*), 339  
 Acroname::BrainStem::PortClass::setCurrentLimitMode (*C++ function*), 339  
 Acroname::BrainStem::PortClass::setDataEnabled (*C++ function*), 334  
 Acroname::BrainStem::PortClass::setDataHS1Enabled (*C++ function*), 335  
 Acroname::BrainStem::PortClass::setDataHS2Enabled (*C++ function*), 335  
 Acroname::BrainStem::PortClass::setDataHSEnabled (*C++ function*), 334  
 Acroname::BrainStem::PortClass::setDataHSRoutingBehavior (*C++ function*), 341  
 Acroname::BrainStem::PortClass::setDataSS1Enabled (*C++ function*), 336  
 Acroname::BrainStem::PortClass::setDataSS2Enabled (*C++ function*), 336  
 Acroname::BrainStem::PortClass::setDataSSEnabled (*C++ function*), 335  
 Acroname::BrainStem::PortClass::setDataSSRoutingBehavior (*C++ function*), 341  
 Acroname::BrainStem::PortClass::setEnabled (*C++ function*), 334  
 Acroname::BrainStem::PortClass::setHSBoost (*C++ function*), 342  
 Acroname::BrainStem::PortClass::setMode (*C++ function*), 339  
 Acroname::BrainStem::PortClass::setName (*C++ function*), 341  
 Acroname::BrainStem::PortClass::setPowerEnabled (*C++ function*), 336  
 Acroname::BrainStem::PortClass::setPowerLimit (*C++ function*), 340  
 Acroname::BrainStem::PortClass::setPowerLimitMode (*C++ function*), 340  
 Acroname::BrainStem::PortClass::setPowerMode (*C++ function*), 334  
 Acroname::BrainStem::PortClass::setVconn1Enabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::setVconn2Enabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::setVconnEnabled (*C++ function*), 337  
 Acroname::BrainStem::PortClass::setVoltageSetpoint (*C++ function*), 338  
 Acroname::BrainStem::PowerDeliveryClass (*C++ class*), 342

Acroname::BrainStem::PowerDeliveryClass::~PowerDeliveryClass (*C++ function*), 343  
Acroname::BrainStem::PowerDeliveryClass::get Cable Current Max (*C++ function*), 346  
Acroname::BrainStem::PowerDeliveryClass::get Cable Orientation (*C++ function*), 347  
Acroname::BrainStem::PowerDeliveryClass::get Cable Speed Max (*C++ function*), 346  
Acroname::BrainStem::PowerDeliveryClass::get Cable Type (*C++ function*), 347  
Acroname::BrainStem::PowerDeliveryClass::get Cable Voltage Max (*C++ function*), 346  
Acroname::BrainStem::PowerDeliveryClass::get Connection State (*C++ function*), 343  
Acroname::BrainStem::PowerDeliveryClass::get Fast Role Swap Current (*C++ function*), 349  
Acroname::BrainStem::PowerDeliveryClass::get Flag Mode (*C++ function*), 348  
Acroname::BrainStem::PowerDeliveryClass::get Number Of Power Data Objects (*C++ function*), 343  
Acroname::BrainStem::PowerDeliveryClass::get Override (*C++ function*), 348  
Acroname::BrainStem::PowerDeliveryClass::get Peak Current Configuration (*C++ function*), 348  
Acroname::BrainStem::PowerDeliveryClass::get Power Data Object (*C++ function*), 343  
Acroname::BrainStem::PowerDeliveryClass::get Power Data Object Enabled (*C++ function*), 344  
Acroname::BrainStem::PowerDeliveryClass::get Power Data Object Enabled List (*C++ function*), 345  
Acroname::BrainStem::PowerDeliveryClass::get Power Data Object List (*C++ function*), 344  
Acroname::BrainStem::PowerDeliveryClass::get Power Role (*C++ function*), 345  
Acroname::BrainStem::PowerDeliveryClass::get Power Role Preferred (*C++ function*), 346  
Acroname::BrainStem::PowerDeliveryClass::get Request Data Object (*C++ function*), 345  
Acroname::BrainStem::PowerDeliveryClass::pack Data Object Attributes (*C++ function*), 349  
Acroname::BrainStem::PowerDeliveryClass::PowerDeliveryClass (*C++ function*), 343  
Acroname::BrainStem::PowerDeliveryClass::request (*C++ function*), 347  
Acroname::BrainStem::PowerDeliveryClass::request Status (*C++ function*), 347  
Acroname::BrainStem::PowerDeliveryClass::reset Entity To Factory Defaults (*C++ function*), 348  
Acroname::BrainStem::PowerDeliveryClass::reset Power Data Object To Default (*C++ function*), 344  
Acroname::BrainStem::PowerDeliveryClass::set Fast Role Swap Current (*C++ function*), 349  
Acroname::BrainStem::PowerDeliveryClass::set Flag Mode (*C++ function*), 348  
Acroname::BrainStem::PowerDeliveryClass::set Override (*C++ function*), 348  
Acroname::BrainStem::PowerDeliveryClass::set Peak Current Configuration (*C++ function*), 349  
Acroname::BrainStem::PowerDeliveryClass::set Power Data Object (*C++ function*), 343  
Acroname::BrainStem::PowerDeliveryClass::set Power Data Object Enabled (*C++ function*), 344  
Acroname::BrainStem::PowerDeliveryClass::set Power Role (*C++ function*), 345  
Acroname::BrainStem::PowerDeliveryClass::set Power Role Preferred (*C++ function*), 346  
Acroname::BrainStem::PowerDeliveryClass::set Request Data Object (*C++ function*), 345  
Acroname::BrainStem::PowerDeliveryClass::unpack Data Object Attributes (*C++ function*), 350  
Acroname::BrainStem::Rail Class (*C++ class*), 350  
Acroname::BrainStem::Rail Class::~Rail Class (*C++ function*), 350  
Acroname::BrainStem::Rail Class::clear Faults (*C++ function*), 355  
Acroname::BrainStem::Rail Class::get Current (*C++ function*), 350  
Acroname::BrainStem::Rail Class::get Current Limit (*C++ function*), 351  
Acroname::BrainStem::Rail Class::get Current Setpoint (*C++ function*), 351  
Acroname::BrainStem::Rail Class::get Enable (*C++ function*), 351  
Acroname::BrainStem::Rail Class::get Kelvin Sensing Enable (*C++ function*), 354  
Acroname::BrainStem::Rail Class::get Kelvin Sensing State (*C++ function*), 354  
Acroname::BrainStem::Rail Class::get Operational Mode (*C++ function*), 355  
Acroname::BrainStem::Rail Class::get Operational State (*C++ function*), 355  
Acroname::BrainStem::Rail Class::get Power (*C++ function*), 353  
Acroname::BrainStem::Rail Class::get Power Limit (*C++ function*), 353  
Acroname::BrainStem::Rail Class::get Power Setpoint (*C++ function*), 353  
Acroname::BrainStem::Rail Class::get Resistance (*C++ function*), 354  
Acroname::BrainStem::Rail Class::get Resistance Setpoint (*C++ function*), 354  
Acroname::BrainStem::Rail Class::get Temperature (*C++ function*), 351  
Acroname::BrainStem::Rail Class::get Voltage (*C++ function*), 352  
Acroname::BrainStem::Rail Class::get Voltage Max Limit (*C++ function*), 353  
Acroname::BrainStem::Rail Class::get Voltage Min Limit (*C++ function*), 352  
Acroname::BrainStem::Rail Class::get Voltage Setpoint (*C++ function*), 352  
Acroname::BrainStem::Rail Class::init (*C++ function*), 350  
Acroname::BrainStem::Rail Class::Rail Class (*C++ function*), 350  
Acroname::BrainStem::Rail Class::set Current Limit (*C++ function*), 351  
Acroname::BrainStem::Rail Class::set Current Setpoint (*C++ function*), 351  
Acroname::BrainStem::Rail Class::set Enable (*C++ function*), 351  
Acroname::BrainStem::Rail Class::set Kelvin Sensing Enable (*C++ function*), 354  
Acroname::BrainStem::Rail Class::set Operational Mode (*C++ function*), 355  
Acroname::BrainStem::Rail Class::set Power Limit (*C++ function*), 353  
Acroname::BrainStem::Rail Class::set Power Setpoint (*C++ function*), 353  
Acroname::BrainStem::Rail Class::set Resistance Setpoint (*C++ function*), 354  
Acroname::BrainStem::Rail Class::set Voltage Max Limit (*C++ function*), 352  
Acroname::BrainStem::Rail Class::set Voltage Min Limit (*C++ function*), 352

Acroname::BrainStem::RailClass::setVoltageSetpoint (*C++ function*), 352  
Acroname::BrainStem::RCServoClass (*C++ class*), 355  
Acroname::BrainStem::RCServoClass::~RCServoClass (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::getEnable (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::getPosition (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::getReverse (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::init (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::RCServoClass (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::setEnabled (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::setPosition (*C++ function*), 356  
Acroname::BrainStem::RCServoClass::setReverse (*C++ function*), 356  
Acroname::BrainStem::RelayClass (*C++ class*), 357  
Acroname::BrainStem::RelayClass::~RelayClass (*C++ function*), 357  
Acroname::BrainStem::RelayClass::getEnable (*C++ function*), 357  
Acroname::BrainStem::RelayClass::getVoltage (*C++ function*), 357  
Acroname::BrainStem::RelayClass::init (*C++ function*), 357  
Acroname::BrainStem::RelayClass::RelayClass (*C++ function*), 357  
Acroname::BrainStem::RelayClass::setEnabled (*C++ function*), 357  
Acroname::BrainStem::SignalClass (*C++ class*), 358  
Acroname::BrainStem::SignalClass::~SignalClass (*C++ function*), 358  
Acroname::BrainStem::SignalClass::getEnable (*C++ function*), 358  
Acroname::BrainStem::SignalClass::getInvert (*C++ function*), 358  
Acroname::BrainStem::SignalClass::getT2Time (*C++ function*), 359  
Acroname::BrainStem::SignalClass::getT3Time (*C++ function*), 359  
Acroname::BrainStem::SignalClass::init (*C++ function*), 358  
Acroname::BrainStem::SignalClass::setEnabled (*C++ function*), 358  
Acroname::BrainStem::SignalClass::setInvert (*C++ function*), 358  
Acroname::BrainStem::SignalClass::setT2Time (*C++ function*), 359  
Acroname::BrainStem::SignalClass::setT3Time (*C++ function*), 359  
Acroname::BrainStem::SignalClass::SignalClass (*C++ function*), 358  
Acroname::BrainStem::StoreClass (*C++ class*), 359  
Acroname::BrainStem::StoreClass::~StoreClass (*C++ function*), 360  
Acroname::BrainStem::StoreClass::getSlotCapacity (*C++ function*), 361  
Acroname::BrainStem::StoreClass::getSlotLocked (*C++ function*), 361  
Acroname::BrainStem::StoreClass::getSlotSize (*C++ function*), 361  
Acroname::BrainStem::StoreClass::getSlotState (*C++ function*), 360  
Acroname::BrainStem::StoreClass::init (*C++ function*), 360  
Acroname::BrainStem::StoreClass::loadSlot (*C++ function*), 360  
Acroname::BrainStem::StoreClass::setSlotLocked (*C++ function*), 361  
Acroname::BrainStem::StoreClass::slotDisable (*C++ function*), 360  
Acroname::BrainStem::StoreClass::slotEnable (*C++ function*), 360  
Acroname::BrainStem::StoreClass::StoreClass (*C++ function*), 360  
Acroname::BrainStem::StoreClass::unloadSlot (*C++ function*), 360  
Acroname::BrainStem::SystemClass (*C++ class*), 361  
Acroname::BrainStem::SystemClass::~SystemClass (*C++ function*), 362  
Acroname::BrainStem::SystemClass::getBootSlot (*C++ function*), 363  
Acroname::BrainStem::SystemClass::getErrors (*C++ function*), 368  
Acroname::BrainStem::SystemClass::getHardwareVersion (*C++ function*), 364  
Acroname::BrainStem::SystemClass::getHBInterval (*C++ function*), 363  
Acroname::BrainStem::SystemClass::getInputCurrent (*C++ function*), 365  
Acroname::BrainStem::SystemClass::getInputPowerBehavior (*C++ function*), 367  
Acroname::BrainStem::SystemClass::getInputPowerBehaviorConfig (*C++ function*), 367  
Acroname::BrainStem::SystemClass::getInputPowerSource (*C++ function*), 367  
Acroname::BrainStem::SystemClass::getInputVoltage (*C++ function*), 365  
Acroname::BrainStem::SystemClass::getLED (*C++ function*), 363  
Acroname::BrainStem::SystemClass::getLinkInterface (*C++ function*), 368  
Acroname::BrainStem::SystemClass::getMaximumTemperature (*C++ function*), 365  
Acroname::BrainStem::SystemClass::getMinimumTemperature (*C++ function*), 364  
Acroname::BrainStem::SystemClass::getModel (*C++ function*), 363  
Acroname::BrainStem::SystemClass::getModule (*C++ function*), 362  
Acroname::BrainStem::SystemClass::getModuleBaseAddress (*C++ function*), 362  
Acroname::BrainStem::SystemClass::getModuleHardwareOffset (*C++ function*), 365  
Acroname::BrainStem::SystemClass::getModuleSoftwareOffset (*C++ function*), 365  
Acroname::BrainStem::SystemClass::getName (*C++ function*), 367  
Acroname::BrainStem::SystemClass::getPowerLimit (*C++ function*), 366  
Acroname::BrainStem::SystemClass::getPowerLimitMax (*C++ function*), 366  
Acroname::BrainStem::SystemClass::getPowerLimitState (*C++ function*), 366  
Acroname::BrainStem::SystemClass::getRouter (*C++ function*), 362

Acroname::BrainStem::SystemClass::getRouterAddressSetting (*C++ function*), 365  
Acroname::BrainStem::SystemClass::getSerialNumber (*C++ function*), 364  
Acroname::BrainStem::SystemClass::getTemperature (*C++ function*), 364  
Acroname::BrainStem::SystemClass::getUnregulatedCurrent (*C++ function*), 367  
Acroname::BrainStem::SystemClass::getUnregulatedVoltage (*C++ function*), 366  
Acroname::BrainStem::SystemClass::getUptime (*C++ function*), 364  
Acroname::BrainStem::SystemClass::getVersion (*C++ function*), 363  
Acroname::BrainStem::SystemClass::init (*C++ function*), 362  
Acroname::BrainStem::SystemClass::logEvents (*C++ function*), 364  
Acroname::BrainStem::SystemClass::reset (*C++ function*), 364  
Acroname::BrainStem::SystemClass::resetDeviceToFactoryDefaults (*C++ function*), 368  
Acroname::BrainStem::SystemClass::resetEntityToFactoryDefaults (*C++ function*), 368  
Acroname::BrainStem::SystemClass::routeToMe (*C++ function*), 366  
Acroname::BrainStem::SystemClass::save (*C++ function*), 364  
Acroname::BrainStem::SystemClass::setBootSlot (*C++ function*), 363  
Acroname::BrainStem::SystemClass::setHBInterval (*C++ function*), 362  
Acroname::BrainStem::SystemClass::setInputPowerBehavior (*C++ function*), 367  
Acroname::BrainStem::SystemClass::setInputPowerBehaviorConfig (*C++ function*), 367  
Acroname::BrainStem::SystemClass::setLED (*C++ function*), 363  
Acroname::BrainStem::SystemClass::setLinkInterface (*C++ function*), 368  
Acroname::BrainStem::SystemClass::setModuleSoftwareOffset (*C++ function*), 365  
Acroname::BrainStem::SystemClass::setName (*C++ function*), 368  
Acroname::BrainStem::SystemClass::setPowerLimitMax (*C++ function*), 366  
Acroname::BrainStem::SystemClass::setRouter (*C++ function*), 362  
Acroname::BrainStem::SystemClass::SystemClass (*C++ function*), 362  
Acroname::BrainStem::TemperatureClass (*C++ class*), 369  
Acroname::BrainStem::TemperatureClass::~TemperatureClass (*C++ function*), 369  
Acroname::BrainStem::TemperatureClass::getValue (*C++ function*), 369  
Acroname::BrainStem::TemperatureClass::getValueMax (*C++ function*), 369  
Acroname::BrainStem::TemperatureClass::getValueMin (*C++ function*), 369  
Acroname::BrainStem::TemperatureClass::init (*C++ function*), 369  
Acroname::BrainStem::TemperatureClass::resetEntityToFactoryDefaults (*C++ function*), 369  
Acroname::BrainStem::TemperatureClass::TemperatureClass (*C++ function*), 369  
Acroname::BrainStem::TimerClass (*C++ class*), 370  
Acroname::BrainStem::TimerClass::~TimerClass (*C++ function*), 370  
Acroname::BrainStem::TimerClass::getExpiration (*C++ function*), 370  
Acroname::BrainStem::TimerClass::getMode (*C++ function*), 370  
Acroname::BrainStem::TimerClass::init (*C++ function*), 370  
Acroname::BrainStem::TimerClass::setExpiration (*C++ function*), 370  
Acroname::BrainStem::TimerClass::setMode (*C++ function*), 370  
Acroname::BrainStem::TimerClass::TimerClass (*C++ function*), 370  
Acroname::BrainStem::UARTClass (*C++ class*), 371  
Acroname::BrainStem::UARTClass::~UARTClass (*C++ function*), 371  
Acroname::BrainStem::UARTClass::getBaudRate (*C++ function*), 371  
Acroname::BrainStem::UARTClass::getEnable (*C++ function*), 371  
Acroname::BrainStem::UARTClass::getProtocol (*C++ function*), 372  
Acroname::BrainStem::UARTClass::init (*C++ function*), 371  
Acroname::BrainStem::UARTClass::setBaudRate (*C++ function*), 371  
Acroname::BrainStem::UARTClass::setEnable (*C++ function*), 371  
Acroname::BrainStem::UARTClass::setProtocol (*C++ function*), 372  
Acroname::BrainStem::UARTClass::UARTClass (*C++ function*), 371  
Acroname::BrainStem::UARTClass::UARTClass (*C++ class*), 372  
Acroname::BrainStem::USBClass (*C++ class*), 372  
Acroname::BrainStem::USBClass::~USBClass (*C++ function*), 372  
Acroname::BrainStem::USBClass::clearPortErrorStatus (*C++ function*), 374  
Acroname::BrainStem::USBClass::getAltModeConfig (*C++ function*), 379  
Acroname::BrainStem::USBClass::getCableFlip (*C++ function*), 379  
Acroname::BrainStem::USBClass::getCC1Current (*C++ function*), 378  
Acroname::BrainStem::USBClass::getCC1Enable (*C++ function*), 378  
Acroname::BrainStem::USBClass::getCC1Voltage (*C++ function*), 378  
Acroname::BrainStem::USBClass::getCC2Current (*C++ function*), 378  
Acroname::BrainStem::USBClass::getCC2Enable (*C++ function*), 378  
Acroname::BrainStem::USBClass::getCC2Voltage (*C++ function*), 379  
Acroname::BrainStem::USBClass::getConnectMode (*C++ function*), 377  
Acroname::BrainStem::USBClass::getDownstreamBoostMode (*C++ function*), 377  
Acroname::BrainStem::USBClass::getDownstreamDataSpeed (*C++ function*), 377  
Acroname::BrainStem::USBClass::getEnumerationDelay (*C++ function*), 375  
Acroname::BrainStem::USBClass::getHubMode (*C++ function*), 374  
Acroname::BrainStem::USBClass::getPortCurrent (*C++ function*), 374

Acroname::BrainStem::USBClass::getPortCurrentLimit (*C++ function*), 375  
 Acroname::BrainStem::USBClass::getPortError (*C++ function*), 376  
 Acroname::BrainStem::USBClass::getPortMode (*C++ function*), 375  
 Acroname::BrainStem::USBClass::getPortState (*C++ function*), 376  
 Acroname::BrainStem::USBClass::getPortVoltage (*C++ function*), 374  
 Acroname::BrainStem::USBClass::getSBU1Voltage (*C++ function*), 380  
 Acroname::BrainStem::USBClass::getSBU2Voltage (*C++ function*), 380  
 Acroname::BrainStem::USBClass::getSBUEnable (*C++ function*), 379  
 Acroname::BrainStem::USBClass::getUpstreamBoostMode (*C++ function*), 376  
 Acroname::BrainStem::USBClass::getUpstreamMode (*C++ function*), 374  
 Acroname::BrainStem::USBClass::getUpstreamState (*C++ function*), 375  
 Acroname::BrainStem::USBClass::init (*C++ function*), 372  
 Acroname::BrainStem::USBClass::setAltModeConfig (*C++ function*), 379  
 Acroname::BrainStem::USBClass::setCableFlip (*C++ function*), 379  
 Acroname::BrainStem::USBClass::setCC1Enable (*C++ function*), 377  
 Acroname::BrainStem::USBClass::setCC2Enable (*C++ function*), 378  
 Acroname::BrainStem::USBClass::setConnectMode (*C++ function*), 377  
 Acroname::BrainStem::USBClass::setDataDisable (*C++ function*), 373  
 Acroname::BrainStem::USBClass::setDataEnable (*C++ function*), 372  
 Acroname::BrainStem::USBClass::setDownstreamBoostMode (*C++ function*), 376  
 Acroname::BrainStem::USBClass::setEnumerationDelay (*C++ function*), 375  
 Acroname::BrainStem::USBClass::setHiSpeedDataDisable (*C++ function*), 373  
 Acroname::BrainStem::USBClass::setHiSpeedDataEnable (*C++ function*), 373  
 Acroname::BrainStem::USBClass::setHubMode (*C++ function*), 374  
 Acroname::BrainStem::USBClass::setPortCurrentLimit (*C++ function*), 375  
 Acroname::BrainStem::USBClass::setPortDisable (*C++ function*), 372  
 Acroname::BrainStem::USBClass::setPortEnable (*C++ function*), 372  
 Acroname::BrainStem::USBClass::setPortMode (*C++ function*), 375  
 Acroname::BrainStem::USBClass::setPowerDisable (*C++ function*), 373  
 Acroname::BrainStem::USBClass::setPowerEnable (*C++ function*), 373  
 Acroname::BrainStem::USBClass::setSBUEnable (*C++ function*), 379  
 Acroname::BrainStem::USBClass::setSuperSpeedDataDisable (*C++ function*), 373  
 Acroname::BrainStem::USBClass::setSuperSpeeddataEnable (*C++ function*), 373  
 Acroname::BrainStem::USBClass::setUpstreamBoostMode (*C++ function*), 376  
 Acroname::BrainStem::USBClass::setUpstreamMode (*C++ function*), 374  
 Acroname::BrainStem::USBClass::USBClass (*C++ function*), 372  
 Acroname::BrainStem::USBSystemClass (*C++ class*), 380  
 Acroname::BrainStem::USBSystemClass::~USBSystemClass (*C++ function*), 380  
 Acroname::BrainStem::USBSystemClass::getDataRoleBehavior (*C++ function*), 383  
 Acroname::BrainStem::USBSystemClass::getDataRoleBehaviorConfig (*C++ function*), 383  
 Acroname::BrainStem::USBSystemClass::getDataRoleList (*C++ function*), 381  
 Acroname::BrainStem::USBSystemClass::getEnabledList (*C++ function*), 381  
 Acroname::BrainStem::USBSystemClass::getEnumerationDelay (*C++ function*), 381  
 Acroname::BrainStem::USBSystemClass::getModeList (*C++ function*), 381  
 Acroname::BrainStem::USBSystemClass::getPowerBehavior (*C++ function*), 382  
 Acroname::BrainStem::USBSystemClass::getPowerBehaviorConfig (*C++ function*), 382  
 Acroname::BrainStem::USBSystemClass::getSelectorMode (*C++ function*), 383  
 Acroname::BrainStem::USBSystemClass::getStateList (*C++ function*), 382  
 Acroname::BrainStem::USBSystemClass::getUpstream (*C++ function*), 380  
 Acroname::BrainStem::USBSystemClass::init (*C++ function*), 380  
 Acroname::BrainStem::USBSystemClass::resetEntityToFactoryDefaults (*C++ function*), 384  
 Acroname::BrainStem::USBSystemClass::setDataRoleBehavior (*C++ function*), 383  
 Acroname::BrainStem::USBSystemClass::setDataRoleBehaviorConfig (*C++ function*), 383  
 Acroname::BrainStem::USBSystemClass::setEnabledList (*C++ function*), 381  
 Acroname::BrainStem::USBSystemClass::setEnumerationDelay (*C++ function*), 381  
 Acroname::BrainStem::USBSystemClass::setModeList (*C++ function*), 382  
 Acroname::BrainStem::USBSystemClass::setPowerBehavior (*C++ function*), 382  
 Acroname::BrainStem::USBSystemClass::setPowerBehaviorConfig (*C++ function*), 382  
 Acroname::BrainStem::USBSystemClass::setSelectorMode (*C++ function*), 383  
 Acroname::BrainStem::USBSystemClass::setUpstream (*C++ function*), 380  
 Acroname::BrainStem::USBSystemClass::USBSysClass (*C++ function*), 380  
 address (*brainstem.module.Module* property), 199  
 aDefs\_GetmodelName (*C++ function*), 388  
 aDiscovery\_EnumerateModules (*C++ function*), 390  
 aDiscovery\_FindFirstModule (*C++ function*), 390  
 aDiscovery\_FindModule (*C++ function*), 390  
 aDiscoveryModuleFoundProc (*C++ type*), 389  
 aErr (*C++ enum*), 391

aErr::aErrAsyncReturn (*C++ enumerator*), 393  
aErr::aErrBusy (*C++ enumerator*), 391  
aErr::aErrCancel (*C++ enumerator*), 393  
aErr::aErrConfiguration (*C++ enumerator*), 392  
aErr::aErrConnection (*C++ enumerator*), 393  
aErr::aErrDuplicate (*C++ enumerator*), 393  
aErr::aErrEOF (*C++ enumerator*), 392  
aErr::aErrFileNameLength (*C++ enumerator*), 391  
aErr::aErrIndexRange (*C++ enumerator*), 393  
aErr::aErrInitialization (*C++ enumerator*), 392  
aErr::aErrInvalidEntity (*C++ enumerator*), 393  
aErr::aErrInvalidOption (*C++ enumerator*), 393  
aErr::aErrIO (*C++ enumerator*), 392  
aErr::aErrMedia (*C++ enumerator*), 393  
aErr::aErrMemory (*C++ enumerator*), 391  
aErr::aErrMode (*C++ enumerator*), 392  
aErr::aErrNone (*C++ enumerator*), 391  
aErr::aErrNotFound (*C++ enumerator*), 391  
aErr::aErrNotReady (*C++ enumerator*), 392  
aErr::aErrOverrun (*C++ enumerator*), 392  
aErr::aErrPacket (*C++ enumerator*), 393  
aErr::aErrParam (*C++ enumerator*), 391  
aErr::aErrParse (*C++ enumerator*), 392  
aErr::aErrPermission (*C++ enumerator*), 392  
aErr::aErrRange (*C++ enumerator*), 392  
aErr::aErrRead (*C++ enumerator*), 392  
aErr::aErrResource (*C++ enumerator*), 393  
aErr::aErrShortCommand (*C++ enumerator*), 393  
aErr::aErrSize (*C++ enumerator*), 392  
aErr::aErrStreamStale (*C++ enumerator*), 393  
aErr::aErrTimeout (*C++ enumerator*), 392  
aErr::aErrUnimplemented (*C++ enumerator*), 393  
aErr::aErrUnknown (*C++ enumerator*), 393  
aErr::aErrVersion (*C++ enumerator*), 392  
aErr::aErrWrite (*C++ enumerator*), 392  
aError\_GetErrorText (*C++ function*), 393  
aFile\_Close (*C++ function*), 395  
aFile\_Delete (*C++ function*), 396  
aFile\_Exists (*C++ function*), 394  
aFile\_GetSize (*C++ function*), 396  
aFile\_Open (*C++ function*), 395  
aFile\_Read (*C++ function*), 395  
aFile\_Seek (*C++ function*), 396  
aFile\_Write (*C++ function*), 395  
a FileMode (*C++ enum*), 394  
a FileMode::a FileModeAppend (*C++ enumerator*), 394  
a FileMode::a FileModeReadOnly (*C++ enumerator*), 394  
a FileMode::a FileModeUnknown (*C++ enumerator*), 394  
a FileMode::a FileModeWriteOnly (*C++ enumerator*), 394  
a FileRef (*C++ type*), 394  
a FileSeekMode (*C++ enum*), 394  
a FileSeekMode::a SeekCurrent (*C++ enumerator*), 394  
a FileSeekMode::a SeekEnd (*C++ enumerator*), 394  
a FileSeekMode::a SeekStart (*C++ enumerator*), 394  
a LIBEXPORT (*C macro*), 387  
a Link\_AwaitFirst (*C++ function*), 400  
a Link\_AwaitPacket (*C++ function*), 399  
a Link\_CreateTCPIP (*C++ function*), 398  
a Link\_CreateUSB (*C++ function*), 398  
a Link\_Destroy (*C++ function*), 398  
a Link\_DrainPackets (*C++ function*), 400  
a Link\_GetFirst (*C++ function*), 399  
a Link\_GetPacket (*C++ function*), 399  
a Link\_GetStatus (*C++ function*), 399  
a Link\_PutPacket (*C++ function*), 400  
a Link\_Reset (*C++ function*), 399  
a LinkRef (*C++ type*), 397  
a LinkSpec\_Create (*C++ function*), 391

aLinkSpec\_Destroy (*C++ function*), 391  
 aMemPtr (*C macro*), 388  
 aMTM\_ETHERSTEM\_BULK\_CAPTURE\_MAX\_HZ (*C macro*), 283  
 aMTM\_ETHERSTEM\_BULK\_CAPTURE\_MIN\_HZ (*C macro*), 283  
 aMTM\_ETHERSTEM\_MODULE\_BASE\_ADDRESS (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_A2D (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_APPS (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_CLOCK (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_DIG (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_I2C (*C macro*), 284  
 aMTM\_ETHERSTEM\_NUM\_INPUT\_SIGNALS (*C macro*), 284  
 aMTM\_ETHERSTEM\_NUM\_INTERNAL\_SLOTS (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_OUTPUT\_SIGNALS (*C macro*), 284  
 aMTM\_ETHERSTEM\_NUM\_POINTERS (*C macro*), 284  
 aMTM\_ETHERSTEM\_NUM\_RAM\_SLOTS (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_SD\_SLOTS (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_SERVOS (*C macro*), 284  
 aMTM\_ETHERSTEM\_NUM\_SIGNALS (*C macro*), 284  
 aMTM\_ETHERSTEM\_NUM\_STORES (*C macro*), 283  
 aMTM\_ETHERSTEM\_NUM\_TIMERS (*C macro*), 284  
 aMTM\_STEM\_BULK\_CAPTURE\_MAX\_HZ (*C macro*), 297  
 aMTM\_STEM\_BULK\_CAPTURE\_MIN\_HZ (*C macro*), 297  
 aMTM\_STEM\_MODULE\_BASE\_ADDRESS (*C macro*), 296  
 aMTM\_STEM\_NUM\_A2D (*C macro*), 296  
 aMTM\_STEM\_NUM\_APPS (*C macro*), 296  
 aMTM\_STEM\_NUM\_CLOCK (*C macro*), 297  
 aMTM\_STEM\_NUM\_DIG (*C macro*), 297  
 aMTM\_STEM\_NUM\_I2C (*C macro*), 297  
 aMTM\_STEM\_NUM\_INPUT\_SIGNALS (*C macro*), 297  
 aMTM\_STEM\_NUM\_INTERNAL\_SLOTS (*C macro*), 297  
 aMTM\_STEM\_NUM\_OUTPUT\_SIGNALS (*C macro*), 297  
 aMTM\_STEM\_NUM\_POINTERS (*C macro*), 297  
 aMTM\_STEM\_NUM\_RAM\_SLOTS (*C macro*), 297  
 aMTM\_STEM\_NUM\_SD\_SLOTS (*C macro*), 297  
 aMTM\_STEM\_NUM\_SERVOS (*C macro*), 297  
 aMTM\_STEM\_NUM\_SIGNALS (*C macro*), 297  
 aMTM\_STEM\_NUM\_STORES (*C macro*), 297  
 aMTM\_STEM\_NUM\_TIMERS (*C macro*), 297  
 aMTM\_USBSTEM\_BULK\_CAPTURE\_MAX\_HZ (*C macro*), 294  
 aMTM\_USBSTEM\_BULK\_CAPTURE\_MIN\_HZ (*C macro*), 294  
 aMTM\_USBSTEM\_MODULE\_BASE\_ADDRESS (*C macro*), 294  
 aMTM\_USBSTEM\_NUM\_A2D (*C macro*), 294  
 aMTM\_USBSTEM\_NUM\_APPS (*C macro*), 294  
 aMTM\_USBSTEM\_NUM\_CLOCK (*C macro*), 294  
 aMTM\_USBSTEM\_NUM\_DIG (*C macro*), 294  
 aMTM\_USBSTEM\_NUM\_I2C (*C macro*), 294  
 aMTM\_USBSTEM\_NUM\_INPUT\_SIGNALS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_INTERNAL\_SLOTS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_OUTPUT\_SIGNALS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_POINTERS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_RAM\_SLOTS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_SD\_SLOTS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_SERVOS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_SIGNALS (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_STORES (*C macro*), 295  
 aMTM\_USBSTEM\_NUM\_TIMERS (*C macro*), 295  
 aMTMDAQ2 (*C++ class*), 280  
 aMTMDAQ2::analog (*C++ member*), 280  
 aMTMDAQ2::app (*C++ member*), 280  
 aMTMDAQ2::digital (*C++ member*), 280  
 aMTMDAQ2::getDifferentialInputRanges (*C++ function*), 281  
 aMTMDAQ2::getOutputRanges (*C++ function*), 281  
 aMTMDAQ2::getSingleEndedInputRanges (*C++ function*), 281  
 aMTMDAQ2::i2c (*C++ member*), 280  
 aMTMDAQ2::pointer (*C++ member*), 281  
 aMTMDAQ2::store (*C++ member*), 281  
 aMTMDAQ2::system (*C++ member*), 281  
 aMTMDAQ2::timer (*C++ member*), 281

aMTMDAQ2\_BULK\_CAPTURE\_MAX\_HZ (*C macro*), 282  
aMTMDAQ2\_BULK\_CAPTURE\_MIN\_HZ (*C macro*), 282  
aMTMDAQ2\_MODULE\_BASE\_ADDRESS (*C macro*), 281  
aMTMDAQ2\_NUM\_ANALOG\_INPUTS (*C macro*), 281  
aMTMDAQ2\_NUM\_ANALOG\_OUTPUTS (*C macro*), 281  
aMTMDAQ2\_NUM\_ANALOGS (*C macro*), 281  
aMTMDAQ2\_NUM\_APPS (*C macro*), 281  
aMTMDAQ2\_NUM\_DIGITALS (*C macro*), 282  
aMTMDAQ2\_NUM\_I2C (*C macro*), 282  
aMTMDAQ2\_NUM\_INTERNAL\_SLOTS (*C macro*), 282  
aMTMDAQ2\_NUM\_POINTERS (*C macro*), 282  
aMTMDAQ2\_NUM\_RAM\_SLOTS (*C macro*), 282  
aMTMDAQ2\_NUM\_STORES (*C macro*), 282  
aMTMDAQ2\_NUM\_TIMERS (*C macro*), 282  
aMTMetherStem (*C++ class*), 282  
aMTMIOSerial (*C++ class*), 284  
aMTMIOSerial::app (*C++ member*), 284  
aMTMIOSerial::digital (*C++ member*), 284  
aMTMIOSerial::i2c (*C++ member*), 284  
aMTMIOSerial::pointer (*C++ member*), 285  
aMTMIOSerial::rail (*C++ member*), 285  
aMTMIOSerial::servo (*C++ member*), 285  
aMTMIOSerial::signal (*C++ member*), 285  
aMTMIOSerial::store (*C++ member*), 285  
aMTMIOSerial::system (*C++ member*), 285  
aMTMIOSerial::temperature (*C++ member*), 285  
aMTMIOSerial::timer (*C++ member*), 285  
aMTMIOSerial::uart (*C++ member*), 284  
aMTMIOSerial::usb (*C++ member*), 285  
aMTMIOSERIAL\_5VRAIL (*C macro*), 286  
aMTMIOSERIAL\_ADJRAIL1 (*C macro*), 286  
aMTMIOSERIAL\_ADJRAIL2 (*C macro*), 286  
aMTMIOSERIAL\_ERROR\_VBUS\_OVERCURRENT (*C macro*), 288  
aMTMIOSERIAL\_MAX\_MICROVOLTAGE (*C macro*), 286  
aMTMIOSERIAL\_MIN\_MICROVOLTAGE (*C macro*), 286  
aMTMIOSERIAL\_MODULE\_BASE\_ADDRESS (*C macro*), 285  
aMTMIOSERIAL\_NUM\_APPS (*C macro*), 285  
aMTMIOSERIAL\_NUM\_DIGITALS (*C macro*), 285  
aMTMIOSERIAL\_NUM\_I2C (*C macro*), 285  
aMTMIOSERIAL\_NUM\_INPUT\_SIGNALS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_INTERNAL\_SLOTS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_OUTPUT\_SIGNALS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_POINTERS (*C macro*), 285  
aMTMIOSERIAL\_NUM\_RAILS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_RAM\_SLOTS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_SERVOS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_SIGNALS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_STORES (*C macro*), 286  
aMTMIOSERIAL\_NUM\_TIMERS (*C macro*), 286  
aMTMIOSERIAL\_NUM\_UART (*C macro*), 286  
aMTMIOSERIAL\_NUM\_USB (*C macro*), 287  
aMTMIOSERIAL\_USB2\_BOOST\_ENABLED (*C macro*), 287  
aMTMIOSERIAL\_USB2\_DATA\_ENABLED (*C macro*), 287  
aMTMIOSERIAL\_USB\_ERROR\_FLAG (*C macro*), 287  
aMTMIOSERIAL\_USB\_NUM\_CHANNELS (*C macro*), 287  
aMTMIOSERIAL\_USB\_VBUS\_ENABLED (*C macro*), 287  
aMTMLoad1 (*C++ class*), 288  
aMTMLoad1::app (*C++ member*), 288  
aMTMLoad1::digital (*C++ member*), 288  
aMTMLoad1::i2c (*C++ member*), 288  
aMTMLoad1::pointer (*C++ member*), 288  
aMTMLoad1::rail (*C++ member*), 288  
aMTMLoad1::store (*C++ member*), 288  
aMTMLoad1::system (*C++ member*), 288  
aMTMLoad1::temperature (*C++ member*), 288  
aMTMLoad1::timer (*C++ member*), 288  
aMTMLOAD1\_MAX\_CURRENT\_LIMIT\_MICROAMPS (*C macro*), 290  
aMTMLOAD1\_MAX\_MICROAMPS (*C macro*), 289

aMTMLOAD1\_MAX\_MICROVOLTAGE (*C macro*), 289  
aMTMLOAD1\_MAX\_MILLIOHMS (*C macro*), 289  
aMTMLOAD1\_MAX\_MILLIWATTS (*C macro*), 289  
aMTMLOAD1\_MAX\_POWER\_LIMIT\_MILLIWATTS (*C macro*), 290  
aMTMLOAD1\_MAX\_VOLTAGE\_LIMIT\_MICROVOLTS (*C macro*), 290  
aMTMLOAD1\_MIN\_CURRENT\_LIMIT\_MICROAMPS (*C macro*), 290  
aMTMLOAD1\_MIN\_MICROAMPS (*C macro*), 289  
aMTMLOAD1\_MIN\_MICROVOLTAGE (*C macro*), 289  
aMTMLOAD1\_MIN\_MILLIOHMS (*C macro*), 289  
aMTMLOAD1\_MIN\_MILLIWATTS (*C macro*), 289  
aMTMLOAD1\_MIN\_POWER\_LIMIT\_MILLIWATTS (*C macro*), 290  
aMTMLOAD1\_MIN\_VOLTAGE\_LIMIT\_MICROVOLTS (*C macro*), 290  
aMTMLOAD1\_MODULE\_BASE\_ADDRESS (*C macro*), 289  
aMTMLOAD1\_NUM\_APPS (*C macro*), 289  
aMTMLOAD1\_NUM\_DIGITALS (*C macro*), 289  
aMTMLOAD1\_NUM\_I2C (*C macro*), 289  
aMTMLOAD1\_NUM\_INTERNAL\_SLOTS (*C macro*), 290  
aMTMLOAD1\_NUM\_POINTERS (*C macro*), 289  
aMTMLOAD1\_NUM\_RAILS (*C macro*), 289  
aMTMLOAD1\_NUM\_RAM\_SLOTS (*C macro*), 290  
aMTMLOAD1\_NUM\_STORES (*C macro*), 290  
aMTMLOAD1\_NUM\_TEMPERATURES (*C macro*), 290  
aMTMLOAD1\_NUM\_TIMERS (*C macro*), 290  
aMTMLOAD1\_RAIL0 (*C macro*), 289  
aMTMPM1 (*C++ class*), 290  
aMTMPM1::app (*C++ member*), 291  
aMTMPM1::digital (*C++ member*), 291  
aMTMPM1::i2c (*C++ member*), 291  
aMTMPM1::pointer (*C++ member*), 291  
aMTMPM1::rail (*C++ member*), 291  
aMTMPM1::store (*C++ member*), 291  
aMTMPM1::system (*C++ member*), 291  
aMTMPM1::temperature (*C++ member*), 291  
aMTMPM1::timer (*C++ member*), 291  
aMTMPM1\_MAX\_CURRENT\_LIMIT\_MICROAMPS (*C macro*), 292  
aMTMPM1\_MAX\_MICROVOLTAGE (*C macro*), 292  
aMTMPM1\_MIN\_CURRENT\_LIMIT\_MICROAMPS (*C macro*), 292  
aMTMPM1\_MIN\_MICROVOLTAGE (*C macro*), 292  
aMTMPM1\_MODULE\_BASE\_ADDRESS (*C macro*), 291  
aMTMPM1\_NUM\_APPS (*C macro*), 291  
aMTMPM1\_NUM\_DIGITALS (*C macro*), 291  
aMTMPM1\_NUM\_I2C (*C macro*), 291  
aMTMPM1\_NUM\_INTERNAL\_SLOTS (*C macro*), 292  
aMTMPM1\_NUM\_POINTERS (*C macro*), 291  
aMTMPM1\_NUM\_RAILS (*C macro*), 291  
aMTMPM1\_NUM\_RAM\_SLOTS (*C macro*), 292  
aMTMPM1\_NUM\_STORES (*C macro*), 292  
aMTMPM1\_NUM\_TEMPERATURES (*C macro*), 292  
aMTMPM1\_NUM\_TIMERS (*C macro*), 292  
aMTMPM1\_RAIL0 (*C macro*), 292  
aMTMPM1\_RAIL1 (*C macro*), 292  
aMTMRelay (*C++ class*), 292  
aMTMRelay::app (*C++ member*), 293  
aMTMRelay::digital (*C++ member*), 293  
aMTMRelay::i2c (*C++ member*), 293  
aMTMRelay::pointer (*C++ member*), 293  
aMTMRelay::relay (*C++ member*), 293  
aMTMRelay::store (*C++ member*), 293  
aMTMRelay::system (*C++ member*), 293  
aMTMRelay::timer (*C++ member*), 293  
aMTMRELAY\_MODULE\_BASE\_ADDRESS (*C macro*), 293  
aMTMRELAY\_NUM\_APPS (*C macro*), 293  
aMTMRELAY\_NUM\_DIGITALS (*C macro*), 293  
aMTMRELAY\_NUM\_I2C (*C macro*), 293  
aMTMRELAY\_NUM\_INTERNAL\_SLOTS (*C macro*), 294  
aMTMRELAY\_NUM\_POINTERS (*C macro*), 293  
aMTMRELAY\_NUM\_RAM\_SLOTS (*C macro*), 294  
aMTMRELAY\_NUM\_RELAYS (*C macro*), 293

aMTMRELAY\_NUM\_STORES (*C macro*), 293  
aMTMRELAY\_NUM\_TIMERS (*C macro*), 294  
aMTMStemModule (*C++ class*), 295  
aMTMStemModule::analog (*C++ member*), 296  
aMTMStemModule::app (*C++ member*), 296  
aMTMStemModule::clock (*C++ member*), 296  
aMTMStemModule::digital (*C++ member*), 296  
aMTMStemModule::i2c (*C++ member*), 296  
aMTMStemModule::pointer (*C++ member*), 296  
aMTMStemModule::servo (*C++ member*), 296  
aMTMStemModule::signal (*C++ member*), 296  
aMTMStemModule::store (*C++ member*), 296  
aMTMStemModule::system (*C++ member*), 296  
aMTMStemModule::timer (*C++ member*), 296  
aMTMUSBStem (*C++ class*), 294  
aMutex\_Create (*C++ function*), 401  
aMutex\_Destroy (*C++ function*), 401  
aMutex\_Identifier (*C++ function*), 401  
aMutex\_Lock (*C++ function*), 401  
aMutex\_TryLock (*C++ function*), 402  
aMutex\_Unlock (*C++ function*), 402  
aMutexRef (*C++ type*), 401  
Analog (*class in brainstem.entity*), 174  
analog\_getBulkCaptureNumberOfSamples (*C++ function*), 527  
analog\_getBulkCaptureSampleRate (*C++ function*), 526  
analog\_getBulkCaptureState (*C++ function*), 527  
analog\_getConfiguration (*C++ function*), 526  
analog\_getEnable (*C++ function*), 524  
analog\_getRange (*C++ function*), 524  
analog\_getValue (*C++ function*), 523  
analog\_getVoltage (*C++ function*), 524  
analog\_Hz\_Maximum (*C macro*), 414  
analog\_Hz\_Minimum (*C macro*), 413  
analog\_initiateBulkCapture (*C++ function*), 527  
analog\_setBulkCaptureNumberOfSamples (*C++ function*), 527  
analog\_setBulkCaptureSampleRate (*C++ function*), 526  
analog\_setConfiguration (*C++ function*), 526  
analog\_setEnable (*C++ function*), 525  
analog\_setRange (*C++ function*), 525  
analogSetValue (*C++ function*), 524  
analog\_setVoltage (*C++ function*), 525  
analogBulkCapture (*C macro*), 414  
analogBulkCaptureNumberOfSamples (*C macro*), 414  
analogBulkCaptureSampleRate (*C macro*), 413  
analogBulkCaptureState (*C macro*), 414  
analogConfiguration (*C macro*), 413  
analogConfigurationHiZ (*C macro*), 413  
analogConfigurationInput (*C macro*), 413  
analogConfigurationOutput (*C macro*), 413  
analogEnable (*C macro*), 415  
analogRange (*C macro*), 414  
analogRange\_P0V064N0V064 (*C macro*), 414  
analogRange\_P0V128N0V128 (*C macro*), 414  
analogRange\_P0V256N0V256 (*C macro*), 414  
analogRange\_P0V512N0V512 (*C macro*), 415  
analogRange\_P0V64N0V64 (*C macro*), 414  
analogRange\_P10V24N0V0 (*C macro*), 415  
analogRange\_P10V24N10V24 (*C macro*), 415  
analogRange\_P1V024N1V024 (*C macro*), 415  
analogRange\_P1V28N0V0 (*C macro*), 414  
analogRange\_P1V28N1V28 (*C macro*), 414  
analogRange\_P2V048N0V0 (*C macro*), 415  
analogRange\_P2V56N0V0 (*C macro*), 415  
analogRange\_P2V56N2V56 (*C macro*), 415  
analogRange\_P4V096N0V0 (*C macro*), 415  
analogRange\_P5V12N0V0 (*C macro*), 415  
analogRange\_P5V12N5V12 (*C macro*), 415  
analogValue (*C macro*), 413

analogVoltage (*C macro*), 413  
 aPacket (*C++ struct*), 402  
 aPacket\_AddByte (*C++ function*), 403  
 aPacket\_Create (*C++ function*), 403  
 aPacket\_CreateWithData (*C++ function*), 403  
 aPacket\_Destroy (*C++ function*), 404  
 aPacket\_IsComplete (*C++ function*), 403  
 aPacket\_Reset (*C++ function*), 403  
 App (*class in brainstem.entity*), 178  
 app\_execute (*C++ function*), 528  
 app\_executeAndReturn (*C++ function*), 528  
 appExecute (*C macro*), 411  
 appReturn (*C macro*), 411  
 aSerial\_Bits (*C++ enum*), 434  
 aSerial\_Bits::aBITS\_7 (*C++ enumerator*), 434  
 aSerial\_Bits::aBITS\_8 (*C++ enumerator*), 434  
 aSerial\_Stop\_bits (*C++ enum*), 434  
 aSerial\_Stop\_bits::aSTOP\_BITS\_1 (*C++ enumerator*), 434  
 aSerial\_Stop\_bits::aSTOP\_BITS\_2 (*C++ enumerator*), 434  
 aSHOWERR (*C macro*), 387  
 aSNPRINTF (*C macro*), 387  
 aStream\_Create (*C++ function*), 434  
 aStream\_CreateFileInput (*C++ function*), 434  
 aStream\_CreateFileOutput (*C++ function*), 435  
 aStream\_CreateLogStream (*C++ function*), 438  
 aStream\_CreateMemory (*C++ function*), 436  
 aStream\_CreatePipe (*C++ function*), 437  
 aStream\_CreateSerial (*C++ function*), 435  
 aStream\_CreateSocket (*C++ function*), 435  
 aStream\_CreateUSB (*C++ function*), 436  
 aStream\_Destroy (*C++ function*), 442  
 aStream\_Flush (*C++ function*), 442  
 aStream\_Read (*C++ function*), 438  
 aStream\_ReadCString (*C++ function*), 440  
 aStream\_ReadCStringRecord (*C++ function*), 441  
 aStream\_ReadRecord (*C++ function*), 439  
 aStream\_Write (*C++ function*), 439  
 aStream\_WriteCString (*C++ function*), 440  
 aStream\_WriteCStringRecord (*C++ function*), 441  
 aStream\_WriteRecord (*C++ function*), 440  
 aStreamBuffer\_Create (*C++ function*), 437  
 aStreamBuffer\_Flush (*C++ function*), 438  
 aStreamBuffer\_Get (*C++ function*), 437  
 aStreamDeleteProc (*C++ type*), 432  
 aStreamGetProc (*C++ type*), 432  
 aStreamPutProc (*C++ type*), 432  
 aStreamRef (*C++ type*), 431  
 aStreamWriteProc (*C++ type*), 433  
 aStringCatSafe (*C macro*), 387  
 aStringCopySafe (*C macro*), 387  
 aSystemBootSlotNone (*C macro*), 406  
 aTime\_GetMSTicks (*C++ function*), 442  
 aTime\_MSleep (*C++ function*), 442  
 aUEI\_RetrieveInt (*C++ function*), 444  
 aUEI\_RetrieveShort (*C++ function*), 444  
 aUEI\_StoreInt (*C++ function*), 444  
 aUEI\_StoreShort (*C++ function*), 444  
 aUSB\_UPSTREAM\_CONFIG\_AUTO (*C macro*), 287  
 aUSB\_UPSTREAM\_CONFIG\_EDGE (*C macro*), 287  
 aUSB\_UPSTREAM\_CONFIG\_ONBOARD (*C macro*), 287  
 aUSB\_UPSTREAM\_EDGE (*C macro*), 287  
 aUSB\_UPSTREAM\_ONBOARD (*C macro*), 287  
 aUSBCSwitch (*C++ class*), 274  
 aUSBCSwitch::app (*C++ member*), 277  
 aUSBCSwitch::daughtercard\_type (*C++ enum*), 276  
 aUSBCSwitch::daughtercard\_type::NO\_DAUGHTERCARD (*C++ enumerator*), 276  
 aUSBCSwitch::daughtercard\_type::PASSIVE\_DAUGHTERCARD (*C++ enumerator*), 276  
 aUSBCSwitch::daughtercard\_type::REDRIVER\_DAUGHTERCARD (*C++ enumerator*), 276

aUSBCSwitch::daughtercard\_type::UNKNOWN\_DAUGHTERCARD (*C++ enumerator*), 276  
aUSBCSwitch::equalizer (*C++ member*), 277  
aUSBCSwitch::EQUALIZER\_2P0\_RECEIVER\_CONFIGS (*C++ enum*), 276  
aUSBCSwitch::EQUALIZER\_2P0\_RECEIVER\_CONFIGS::LEVEL\_1\_2P0 (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_2P0\_RECEIVER\_CONFIGS::LEVEL\_2\_2P0 (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_2P0\_TRANSMITTER\_CONFIGS (*C++ enum*), 276  
aUSBCSwitch::EQUALIZER\_2P0\_TRANSMITTER\_CONFIGS::TRANSMITTER\_2P0\_0mV (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_2P0\_TRANSMITTER\_CONFIGS::TRANSMITTER\_2P0\_40mV (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_2P0\_TRANSMITTER\_CONFIGS::TRANSMITTER\_2P0\_60mV (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_2P0\_TRANSMITTER\_CONFIGS::TRANSMITTER\_2P0\_80mV (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS (*C++ enum*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_10\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_11\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_12\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_13\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_14\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_15\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_16\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_1\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_2\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_3\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_4\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_5\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_6\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_7\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_8\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_RECEIVER\_CONFIGS::LEVEL\_9\_3P0 (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS (*C++ enum*), 274  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_0db\_COM\_0db\_1100mV (*C++ enumerator*), 274  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_0db\_COM\_0db\_1300mV (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_0db\_COM\_0db\_900mV (*C++ enumerator*), 274  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_0db\_COM\_1db\_1100mV (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_0db\_COM\_1db\_900mV (*C++ enumerator*), 274  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_1db\_COM\_0db\_1100mV (*C++ enumerator*), 274  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_1db\_COM\_0db\_900mV (*C++ enumerator*), 274  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_1db\_COM\_1db\_900mV (*C++ enumerator*), 274  
aUSBCSwitch::EQUALIZER\_3P0\_TRANSMITTER\_CONFIGS::MUX\_2db\_COM\_2db\_1100mV (*C++ enumerator*), 275  
aUSBCSwitch::EQUALIZER\_CHANNELS (*C++ enum*), 276  
aUSBCSwitch::EQUALIZER\_CHANNELS::BOTH (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_CHANNELS::COMMON (*C++ enumerator*), 276  
aUSBCSwitch::EQUALIZER\_CHANNELS::MUX (*C++ enumerator*), 276  
aUSBCSwitch::mux (*C++ member*), 277  
aUSBCSwitch::pointer (*C++ member*), 277  
aUSBCSwitch::store (*C++ member*), 277  
aUSBCSwitch::system (*C++ member*), 277  
aUSBCSwitch::timer (*C++ member*), 277  
aUSBCSwitch::usb (*C++ member*), 277  
aUSBCSWITCH\_MODULE (*C macro*), 277  
aUSBCSWITCH\_NUM\_APPS (*C macro*), 277  
aUSBCSWITCH\_NUM\_EQ (*C macro*), 278  
aUSBCSWITCH\_NUM\_INTERNAL\_SLOTS (*C macro*), 277  
aUSBCSWITCH\_NUM\_MUX (*C macro*), 278  
aUSBCSWITCH\_NUM\_MUX\_CHANNELS (*C macro*), 278  
aUSBCSWITCH\_NUM\_POINTERS (*C macro*), 277  
aUSBCSWITCH\_NUM\_RAM\_SLOTS (*C macro*), 277  
aUSBCSWITCH\_NUM\_STORES (*C macro*), 277  
aUSBCSWITCH\_NUM\_TIMERS (*C macro*), 277  
aUSBCSWITCH\_NUM\_USB (*C macro*), 278  
aUSBHub2x4 (*C++ class*), 272  
aUSBHub2x4::app (*C++ member*), 272  
aUSBHub2x4::pointer (*C++ member*), 272  
aUSBHub2x4::store (*C++ member*), 272  
aUSBHub2x4::system (*C++ member*), 272  
aUSBHub2x4::temperature (*C++ member*), 272  
aUSBHub2x4::timer (*C++ member*), 272  
aUSBHub2x4::usb (*C++ member*), 272  
aUSBHUB2X4\_CONSTANT\_CURRENT (*C macro*), 274  
aUSBHUB2X4\_DEVICE\_ATTACHED (*C macro*), 273

aUSBHub2X4\_ERROR\_DISCHARGE (*C macro*), 274  
 aUSBHUB2X4\_ERROR\_OVER\_TEMPERATURE (*C macro*), 274  
 aUSBHUB2X4\_ERROR\_VBUS\_OVERCURRENT (*C macro*), 274  
 aUSBHUB2X4\_MODULE (*C macro*), 273  
 aUSBHUB2X4\_NUM\_APPS (*C macro*), 273  
 aUSBHUB2X4\_NUM\_INTERNAL\_SLOTS (*C macro*), 273  
 aUSBHUB2X4\_NUM\_POINTERS (*C macro*), 273  
 aUSBHUB2X4\_NUM\_RAM\_SLOTS (*C macro*), 273  
 aUSBHUB2X4\_NUM\_STORES (*C macro*), 273  
 aUSBHUB2X4\_NUM\_TIMERS (*C macro*), 273  
 aUSBHUB2X4\_NUM\_USB (*C macro*), 273  
 aUSBHUB2x4\_NUM\_USB\_PORTS (*C macro*), 273  
 aUSBHUB2X4\_USB2\_BOOST\_ENABLED (*C macro*), 273  
 aUSBHUB2X4\_USB2\_DATA\_ENABLED (*C macro*), 273  
 aUSBHUB2X4\_USB\_ERROR\_FLAG (*C macro*), 273  
 aUSBHUB2X4\_USB\_VBUS\_ENABLED (*C macro*), 273  
 aUSBHub3c (*C++ class*), 266  
 aUSBHub3c::app (*C++ member*), 267  
 aUSBHub3c::hub (*C++ member*), 267  
 aUSBHub3c::HubClass (*C++ class*), 268  
 aUSBHub3c::i2c (*C++ member*), 268  
 aUSBHub3c::pd (*C++ member*), 267  
 aUSBHub3c::pointer (*C++ member*), 267  
 aUSBHub3c::PORT\_ID (*C++ enum*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_0 (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_1 (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_2 (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_3 (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_4 (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_5 (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_CONTROL (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID::kPORT\_ID\_POWER\_C (*C++ enumerator*), 267  
 aUSBHub3c::PORT\_ID\_t (*C++ type*), 267  
 aUSBHub3c::rail (*C++ member*), 267  
 aUSBHub3c::store (*C++ member*), 267  
 aUSBHub3c::system (*C++ member*), 268  
 aUSBHub3c::temperature (*C++ member*), 268  
 aUSBHub3c::timer (*C++ member*), 268  
 aUSBHub3c::uart (*C++ member*), 268  
 aUSBHub3c::usb (*C++ member*), 268  
 aUSBHUB3C\_MODULE (*C macro*), 268  
 aUSBHUB3C\_NUM\_APPS (*C macro*), 268  
 aUSBHUB3C\_NUM\_I2C (*C macro*), 269  
 aUSBHUB3C\_NUM\_INTERNAL\_SLOTS (*C macro*), 268  
 aUSBHUB3C\_NUM\_PD\_PORTS (*C macro*), 269  
 aUSBHUB3C\_NUM\_PD\_RULES\_PER\_PORT (*C macro*), 269  
 aUSBHUB3C\_NUM\_POINTERS (*C macro*), 268  
 aUSBHUB3C\_NUM\_RAILS (*C macro*), 269  
 aUSBHUB3C\_NUM\_RAM\_SLOTS (*C macro*), 268  
 aUSBHUB3C\_NUM\_STORES (*C macro*), 268  
 aUSBHUB3C\_NUM\_TEMPERATURES (*C macro*), 269  
 aUSBHUB3C\_NUM\_TIMERS (*C macro*), 269  
 aUSBHUB3C\_NUM\_UART (*C macro*), 269  
 aUSBHUB3C\_NUM\_USB (*C macro*), 269  
 aUSBHUB3C\_NUM\_USB\_PORTS (*C macro*), 269  
 aUSBHUB3C\_STORE\_EEPROM\_INDEX (*C macro*), 269  
 aUSBHUB3C\_STORE\_INTERNAL\_INDEX (*C macro*), 268  
 aUSBHUB3C\_STORE\_RAM\_INDEX (*C macro*), 269  
 aUSBHub3p (*C++ class*), 269  
 aUSBHub3p::app (*C++ member*), 270  
 aUSBHub3p::pointer (*C++ member*), 270  
 aUSBHub3p::store (*C++ member*), 270  
 aUSBHub3p::system (*C++ member*), 270  
 aUSBHub3p::temperature (*C++ member*), 270  
 aUSBHub3p::timer (*C++ member*), 270  
 aUSBHub3p::usb (*C++ member*), 270  
 aUSBHUB3P\_DEVICE\_ATTACHED (*C macro*), 271  
 aUSBHUB3P\_ERROR\_HUB\_POWER (*C macro*), 271

aUSBHUB3P\_ERROR\_OVER\_TEMPERATURE (*C macro*), 272  
aUSBHUB3P\_ERROR\_VBUS\_BACKDRIVE (*C macro*), 271  
aUSBHUB3P\_ERROR\_VBUS\_OVERCURRENT (*C macro*), 271  
aUSBHUB3P\_MODULE (*C macro*), 270  
aUSBHUB3P\_NUM\_APPS (*C macro*), 270  
aUSBHUB3P\_NUM\_INTERNAL\_SLOTS (*C macro*), 270  
aUSBHUB3P\_NUM\_POINTERS (*C macro*), 270  
aUSBHUB3P\_NUM\_RAM\_SLOTS (*C macro*), 270  
aUSBHUB3P\_NUM\_STORES (*C macro*), 270  
aUSBHUB3P\_NUM\_TIMERS (*C macro*), 270  
aUSBHUB3P\_NUM\_USB (*C macro*), 270  
aUSBHUB3P\_NUM\_USB\_PORTS (*C macro*), 271  
aUSBHUB3P\_USB2\_BOOST\_ENABLED (*C macro*), 271  
aUSBHUB3P\_USB2\_DATA\_ENABLED (*C macro*), 271  
aUSBHUB3P\_USB3\_DATA\_ENABLED (*C macro*), 271  
aUSBHUB3P\_USB\_ERROR\_FLAG (*C macro*), 271  
aUSBHUB3P\_USB\_SPEED\_USB2 (*C macro*), 271  
aUSBHUB3P\_USB\_SPEED\_USB3 (*C macro*), 271  
aUSBHUB3P\_USB\_VBUS\_ENABLED (*C macro*), 271  
aVALIDPACKET (*C++ function*), 402  
aVersion\_DestroyFeatureList (*C++ function*), 447  
aVersion\_GetFeatureList (*C++ function*), 447  
aVersion\_GetMajor (*C++ function*), 446  
aVersion\_GetMinor (*C++ function*), 446  
aVersion\_GetPatch (*C++ function*), 446  
aVersion\_GetString (*C++ function*), 446  
aVersion\_IsAtLeast (*C++ function*), 446  
aVERSION\_MAJOR (*C macro*), 445  
aVERSION\_MINOR (*C macro*), 445  
aVersion\_ParseMajor (*C++ function*), 445  
aVersion\_ParseMinor (*C++ function*), 445  
aVersion\_ParsePatch (*C++ function*), 446  
aVersion\_ParseString (*C++ function*), 446  
aVERSION\_PATCH (*C macro*), 445

## B

bAutoNetworking (*brainstem.module.Module* property), 200  
bContinueSearch (*C++ type*), 389  
bitSlotError (*C macro*), 410  
brainstem.defs  
    module, 181  
brainstem.discover  
    module, 184  
brainstem.entity  
    module, 193  
brainstem.link  
    module, 195  
brainstem.module  
    module, 196  
brainstem.result  
    module, 234  
brainstem.version  
    module, 260  
bulkCaptureError (*C macro*), 414  
bulkCaptureFinished (*C macro*), 414  
bulkCaptureIdle (*C macro*), 414  
bulkCapturePending (*C macro*), 414

## C

call\_UEI () (*brainstem.module.Entity* method), 197  
capacityBuild (*C macro*), 422  
capacityClassQuantity (*C macro*), 422  
capacityEntityGroup (*C macro*), 422  
capacitySubClassQuantity (*C macro*), 422  
capacitySubClassSize (*C macro*), 422  
capacityUEI (*C macro*), 422  
clearFaults () (*brainstem.entity.Rail* method), 226

```

clearPortErrorStatus () (brainstem.entity.USB method), 251
Clock (class in brainstem.entity), 179
clock_getDay (C++ function), 530
clock_getHour (C++ function), 530
clock_getMinute (C++ function), 531
clock_getMonth (C++ function), 529
clock_getSecond (C++ function), 531
clock_getYear (C++ function), 529
clock_setDay (C++ function), 530
clock_setHour (C++ function), 530
clock_setMinute (C++ function), 531
clock_setMonth (C++ function), 529
clock_setSecond (C++ function), 531
clock_setYear (C++ function), 529
clockDay (C macro), 425
clockHour (C macro), 425
clockMinute (C macro), 425
clockMonth (C macro), 425
clockSecond (C macro), 425
clockYear (C macro), 425
cmdANALOG (C macro), 413
cmdAPP (C macro), 410
cmdCAPACITY (C macro), 422
cmdCLOCK (C macro), 424
cmdDEBUG (C macro), 413
cmdDIGITAL (C macro), 416
cmdLAST (C macro), 431
cmdMUX (C macro), 411
cmdPOINTER (C macro), 412
cmdRAIL (C macro), 417
cmdsLOT (C macro), 409
cmdSTORE (C macro), 423
cmdSYSTEM (C macro), 406
cmdTEMPERATURE (C macro), 421
cmdTIMER (C macro), 424
cmdUPGRADE (C macro), 431
cmdUSB (C macro), 425
command (brainstem.module.Entity property), 197
connect () (brainstem.module.Module method), 200
connect () (brainstem.stem.EtherStem method), 171
connect () (brainstem.stem.MTMDAQ1 method), 170
connect () (brainstem.stem.MTMDAQ2 method), 162
connect () (brainstem.stem.MTMEtherStem method), 163
connect () (brainstem.stem.MTMIOSerial method), 164
connect () (brainstem.stem.MTMLOAD1 method), 165
connect () (brainstem.stem.MTMPM1 method), 166
connect () (brainstem.stem.MTMRelay method), 167
connect () (brainstem.stem.MTMUSBStem method), 168
connect () (brainstem.stem.USBCSwitch method), 160
connect () (brainstem.stem.USBHub2x4 method), 158
connect () (brainstem.stem.USBHub3p method), 156
connect () (brainstem.stem.USBStem method), 172
connectFromSpec () (brainstem.module.Module method), 200
connectThroughLinkModule () (brainstem.module.Module method), 200

```

## D

```

dataType (C++ enum), 443
dataType::aUEI_BYTE (C++ enumerator), 443
dataType::aUEI_BYTES (C++ enumerator), 443
dataType::aUEI_INT (C++ enumerator), 443
dataType::aUEI_SHORT (C++ enumerator), 443
dataType::aUEI_VOID (C++ enumerator), 443
DefaultOperationalRailMode_Value (C macro), 418
DefaultPointerMode (C macro), 412
DefaultTimerMode (C macro), 424
DeviceNode (class in brainstem.discover), 184
Digital (class in brainstem.entity), 182

```

digital\_getConfiguration (*C++ function*), 532  
digital\_getState (*C++ function*), 533  
digital\_getStateAll (*C++ function*), 533  
digital\_setConfiguration (*C++ function*), 532  
digital\_setState (*C++ function*), 532  
digital\_setStateAll (*C++ function*), 533  
digitalConfiguration (*C macro*), 416  
digitalConfigurationHiZ (*C macro*), 416  
digitalConfigurationInput (*C macro*), 416  
digitalConfigurationInputNoPull (*C macro*), 416  
digitalConfigurationInputPullDown (*C macro*), 416  
digitalConfigurationInputPullUp (*C macro*), 416  
digitalConfigurationOutput (*C macro*), 416  
digitalConfigurationRCServoInput (*C macro*), 416  
digitalConfigurationRCServoOutput (*C macro*), 416  
digitalConfigurationSignalCounterInput (*C macro*), 416  
digitalConfigurationSignalInput (*C macro*), 416  
digitalConfigurationSignalOutput (*C macro*), 416  
digitalState (*C macro*), 416  
digitalStateAll (*C macro*), 417  
disconnect () (*brainstem.module.Module method*), 200  
discoverAndConnect () (*brainstem.module.Module method*), 200  
drain\_UEI () (*brainstem.module.Entity method*), 197

## E

Entity (*class in brainstem.module*), 196  
Equalizer (*class in brainstem.entity*), 193  
equalizer\_getReceiverConfig (*C++ function*), 534  
equalizer\_getTransmitterConfig (*C++ function*), 534  
equalizer\_setReceiverConfig (*C++ function*), 534  
equalizer\_setTransmitterConfig (*C++ function*), 534  
error (*brainstem.result.Result property*), 234  
EtherStem (*class in brainstem.stem*), 170  
execute () (*brainstem.entity.App method*), 178  
executeAndwaitForReturn () (*brainstem.entity.App method*), 178

## F

findAllModules () (*in module brainstem.discover*), 185  
findFirstModule () (*in module brainstem.discover*), 185  
findModule () (*in module brainstem.discover*), 185

## G

get\_UEI () (*brainstem.module.Entity method*), 197  
get\_UEI\_with\_param () (*brainstem.module.Entity method*), 198  
get\_UEIBytes32 () (*brainstem.module.Entity method*), 197  
get\_UEIBytes8 () (*brainstem.module.Entity method*), 198  
get\_usbPortStateCOM\_ORIENT\_STATUS (*C macro*), 278  
get\_usbPortStateDaughterCard (*C macro*), 279  
get\_usbPortStateMUX\_ORIENT\_STATUS (*C macro*), 279  
get\_usbPortStateSPEED\_STATUS (*C macro*), 279  
get\_version\_string () (*in module brainstem.version*), 260  
getAllocatedPower () (*brainstem.entity.Port method*), 207  
getAltModeConfig () (*brainstem.entity.USB method*), 251  
getAvailablePower () (*brainstem.entity.Port method*), 207  
getBaudRate () (*brainstem.entity.UART method*), 248  
getBootSlot () (*brainstem.entity.System method*), 236  
getBulkCaptureNumberOfSamples () (*brainstem.entity.Analog method*), 174  
getBulkCaptureSampleRate () (*brainstem.entity.Analog method*), 174  
getBulkCaptureState () (*brainstem.entity.Analog method*), 175  
getCableCurrentMax () (*brainstem.entity.PowerDelivery method*), 218  
getCableFlip () (*brainstem.entity.USB method*), 251  
getCableOrientation () (*brainstem.entity.PowerDelivery method*), 218  
getCableSpeedMax () (*brainstem.entity.PowerDelivery method*), 218  
getCableType () (*brainstem.entity.PowerDelivery method*), 219  
getCableVoltageMax () (*brainstem.entity.PowerDelivery method*), 219  
getCC1Current () (*brainstem.entity.USB method*), 251

getCC1Enabled() (*brainstem.entity.Port method*), 207  
getCC1Voltage() (*brainstem.entity.USB method*), 251  
getCC2Current() (*brainstem.entity.USB method*), 251  
getCC2Enabled() (*brainstem.entity.Port method*), 207  
getCC2Voltage() (*brainstem.entity.USB method*), 251  
getCCEnabled() (*brainstem.entity.Port method*), 207  
getChannel() (*brainstem.entity.Mux method*), 202  
getChar() (*brainstem.entity.Pointer method*), 204  
getConfiguration() (*brainstem.entity.Analog method*), 175  
getConfiguration() (*brainstem.entity.Digital method*), 182  
getConfiguration() (*brainstem.entity.Mux method*), 202  
getConnectionState() (*brainstem.entity.PowerDelivery method*), 219  
getConnectMode() (*brainstem.entity.USB method*), 251  
getCurrent() (*brainstem.entity.Rail method*), 226  
getCurrentLimit() (*brainstem.entity.Port method*), 208  
getCurrentLimit() (*brainstem.entity.Rail method*), 226  
getCurrentLimitMode() (*brainstem.entity.Port method*), 208  
getCurrentSetpoint() (*brainstem.entity.Rail method*), 226  
getDataEnabled() (*brainstem.entity.Port method*), 208  
getDataHS1Enabled() (*brainstem.entity.Port method*), 208  
getDataHS2Enabled() (*brainstem.entity.Port method*), 208  
getDataHSEnabled() (*brainstem.entity.Port method*), 209  
getDataHSRoutingBehavior() (*brainstem.entity.Port method*), 209  
getDataRole() (*brainstem.entity.Port method*), 209  
getDataRoleBehavior() (*brainstem.entity.USBSystem method*), 257  
getDataRoleBehaviorConfig() (*brainstem.entity.USBSystem method*), 257  
getDataRoleList() (*brainstem.entity.USBSystem method*), 257  
getDataSpeed() (*brainstem.entity.Port method*), 210  
getDataSS1Enabled() (*brainstem.entity.Port method*), 209  
getDataSS2Enabled() (*brainstem.entity.Port method*), 209  
getDataSSEnabled() (*brainstem.entity.Port method*), 209  
getDataSSRoutingBehavior() (*brainstem.entity.Port method*), 210  
getDay() (*brainstem.entity.Clock method*), 179  
getDownstreamBoostMode() (*brainstem.entity.USB method*), 252  
getDownstreamDataSpeed() (*brainstem.entity.USB method*), 252  
getDownstreamDevices() (*in module brainstemdiscover*), 185  
getEnable() (*brainstem.entity.Analog method*), 175  
getEnable() (*brainstem.entity.Mux method*), 202  
getEnable() (*brainstem.entity.Rail method*), 226  
getEnable() (*brainstem.entity.RCServo method*), 232  
getEnable() (*brainstem.entity.Relay method*), 233  
getEnable() (*brainstem.entity.Signal method*), 235  
getEnable() (*brainstem.entity.UART method*), 248  
getEnabled() (*brainstem.entity.Port method*), 210  
getEnabledList() (*brainstem.entity.USBSystem method*), 257  
getEnumerationDelay() (*brainstem.entity.USB method*), 252  
getEnumerationDelay() (*brainstem.entity.USBSystem method*), 258  
getErrors() (*brainstem.entity.Port method*), 210  
getErrors() (*brainstem.entity.System method*), 236  
getExpiration() (*brainstem.entity.Timer method*), 247  
getFastRoleSwapCurrent() (*brainstem.entity.PowerDelivery method*), 219  
getFlagMode() (*brainstem.entity.PowerDelivery method*), 219  
getHardwareVersion() (*brainstem.entity.System method*), 236  
getHBInterval() (*brainstem.entity.System method*), 236  
getHour() (*brainstem.entity.Clock method*), 179  
getHSBoost() (*brainstem.entity.Port method*), 210  
getHubMode() (*brainstem.entity.USB method*), 252  
getInputCurrent() (*brainstem.entity.System method*), 237  
getInputPowerBehavior() (*brainstem.entity.System method*), 237  
getInputPowerBehaviorConfig() (*brainstem.entity.System method*), 237  
getInputPowerSource() (*brainstem.entity.System method*), 237  
getInputVoltage() (*brainstem.entity.System method*), 237  
getInt() (*brainstem.entity.Pointer method*), 204  
getInvert() (*brainstem.entity.Signal method*), 235  
getKelvinSensingEnable() (*brainstem.entity.Rail method*), 227  
getKelvinSensingState() (*brainstem.entity.Rail method*), 227  
getLED() (*brainstem.entity.System method*), 237  
getLinkInterface() (*brainstem.entity.System method*), 238

getMaximumTemperature () (*brainstem.entity.System method*), 238  
getMinimumTemperature () (*brainstem.entity.System method*), 238  
getMinute () (*brainstem.entity.Clock method*), 179  
getMode () (*brainstem.entity.Pointer method*), 204  
getMode () (*brainstem.entity.Port method*), 210  
getMode () (*brainstem.entity.Timer method*), 248  
getModel () (*brainstem.entity.System method*), 238  
getModeList () (*brainstem.entity.USBSYSTEM method*), 258  
getModule () (*brainstem.entity.System method*), 238  
getModuleBaseAddress () (*brainstem.entity.System method*), 239  
getModuleHardwareOffset () (*brainstem.entity.System method*), 239  
getModuleSoftwareOffset () (*brainstem.entity.System method*), 239  
getMonth () (*brainstem.entity.Clock method*), 179  
getName () (*brainstem.entity.Port method*), 211  
getName () (*brainstem.entity.System method*), 239  
getNumberOfPowerDataObjects () (*brainstem.entity.PowerDelivery method*), 220  
getOffset () (*brainstem.entity.Pointer method*), 204  
getOperationalMode () (*brainstem.entity.Rail method*), 227  
getOperationalState () (*brainstem.entity.Rail method*), 227  
getOverride () (*brainstem.entity.PowerDelivery method*), 220  
getPeakCurrentConfiguration () (*brainstem.entity.PowerDelivery method*), 220  
getPortCurrent () (*brainstem.entity.USB method*), 252  
getPortCurrentLimit () (*brainstem.entity.USB method*), 252  
getPortError () (*brainstem.entity.USB method*), 252  
getPortMode () (*brainstem.entity.USB method*), 252  
getPortState () (*brainstem.entity.USB method*), 252  
getPortVoltage () (*brainstem.entity.USB method*), 252  
getPosition () (*brainstem.entity.RCServo method*), 232  
getPower () (*brainstem.entity.Rail method*), 227  
getPowerBehavior () (*brainstem.entity.USBSYSTEM method*), 258  
getPowerBehaviorConfig () (*brainstem.entity.USBSYSTEM method*), 258  
getPowerDataObject () (*brainstem.entity.PowerDelivery method*), 220  
getPowerDataObjectEnabled () (*brainstem.entity.PowerDelivery method*), 221  
getPowerDataObjectEnabledList () (*brainstem.entity.PowerDelivery method*), 221  
getPowerDataObjectList () (*brainstem.entity.PowerDelivery method*), 221  
getPowerEnabled () (*brainstem.entity.Port method*), 211  
getPowerLimit () (*brainstem.entity.Port method*), 211  
getPowerLimit () (*brainstem.entity.Rail method*), 227  
getPowerLimit () (*brainstem.entity.System method*), 239  
getPowerLimitMax () (*brainstem.entity.System method*), 239  
getPowerLimitMode () (*brainstem.entity.Port method*), 211  
getPowerLimitState () (*brainstem.entity.System method*), 240  
getPowerMode () (*brainstem.entity.Port method*), 211  
getPowerRole () (*brainstem.entity.PowerDelivery method*), 221  
getPowerRolePreferred () (*brainstem.entity.PowerDelivery method*), 222  
getPowerSetpoint () (*brainstem.entity.Rail method*), 228  
getProtocol () (*brainstem.entity.UART method*), 249  
getRange () (*brainstem.entity.Analog method*), 175  
getReceiverConfig () (*brainstem.entity.Equalizer method*), 193  
getRequestDataObject () (*brainstem.entity.PowerDelivery method*), 222  
getResistance () (*brainstem.entity.Rail method*), 228  
getResistanceSetpoint () (*brainstem.entity.Rail method*), 228  
getReverse () (*brainstem.entity.RCServo method*), 232  
getRouter () (*brainstem.entity.System method*), 240  
getRouterAddressSetting () (*brainstem.entity.System method*), 240  
getSBU1Voltage () (*brainstem.entity.USB method*), 253  
getSBU2Voltage () (*brainstem.entity.USB method*), 253  
getSecond () (*brainstem.entity.Clock method*), 179  
getSelectorMode () (*brainstem.entity.USBSYSTEM method*), 258  
getSerialNumber () (*brainstem.entity.System method*), 240  
getShort () (*brainstem.entity.Pointer method*), 205  
getSlotCapacity () (*brainstem.entity.Store method*), 245  
getSlotLocked () (*brainstem.entity.Store method*), 245  
getSlotSize () (*brainstem.entity.Store method*), 245  
getSlotState () (*brainstem.entity.Store method*), 245  
getSpeed () (*brainstem.entity.I2C method*), 194  
getSplitMode () (*brainstem.entity.Mux method*), 203  
getState () (*brainstem.entity.Digital method*), 182

getState() (*brainstem.entity.Port method*), 211  
 getStateAll() (*brainstem.entity.Digital method*), 183  
 getStateList() (*brainstem.entity.USBSYSTEM method*), 258  
 getStatus() (*brainstem.module.Module method*), 201  
 getT2Time() (*brainstem.entity.Signal method*), 235  
 getT3Time() (*brainstem.entity.Signal method*), 235  
 getTemperature() (*brainstem.entity.Rail method*), 228  
 getTemperature() (*brainstem.entity.System method*), 240  
 getTransferStore() (*brainstem.entity.Pointer method*), 205  
 getTransmitterConfig() (*brainstem.entity.Equalizer method*), 193  
 getUnregulatedCurrent() (*brainstem.entity.System method*), 240  
 getUnregulatedVoltage() (*brainstem.entity.System method*), 241  
 getUpstream() (*brainstem.entity.USBSYSTEM method*), 259  
 getUpstreamBoostMode() (*brainstem.entity.USB method*), 253  
 getUpstreamMode() (*brainstem.entity.USB method*), 253  
 getUpstreamState() (*brainstem.entity.USB method*), 253  
 getUptime() (*brainstem.entity.System method*), 241  
 getValue() (*brainstem.entity.Analog method*), 175  
 getValue() (*brainstem.entity.Temperature method*), 246  
 getValueMax() (*brainstem.entity.Temperature method*), 247  
 getValueMin() (*brainstem.entity.Temperature method*), 247  
 getVbusAccumulatedPower() (*brainstem.entity.Port method*), 212  
 getVbusCurrent() (*brainstem.entity.Port method*), 212  
 getVbusVoltage() (*brainstem.entity.Port method*), 212  
 getVconn1Enabled() (*brainstem.entity.Port method*), 212  
 getVconn2Enabled() (*brainstem.entity.Port method*), 212  
 getVconnAccumulatedPower() (*brainstem.entity.Port method*), 212  
 getVconnCurrent() (*brainstem.entity.Port method*), 213  
 getVconnEnabled() (*brainstem.entity.Port method*), 213  
 getVconnVoltage() (*brainstem.entity.Port method*), 213  
 getVersion() (*brainstem.entity.System method*), 241  
 getVoltage() (*brainstem.entity.Analog method*), 176  
 getVoltage() (*brainstem.entity.Mux method*), 203  
 getVoltage() (*brainstem.entity.Rail method*), 228  
 getVoltage() (*brainstem.entity.Relay method*), 234  
 getVoltageMaxLimit() (*brainstem.entity.Rail method*), 229  
 getVoltageMinLimit() (*brainstem.entity.Rail method*), 229  
 getVoltageSetpoint() (*brainstem.entity.Port method*), 213  
 getVoltageSetpoint() (*brainstem.entity.Rail method*), 229  
 getYear() (*brainstem.entity.Clock method*), 179

**I**

I2C (*class in brainstem.entity*), 194  
 i2c\_getSpeed (*C++ function*), 536  
 i2c\_read (*C++ function*), 535  
 i2c\_setPullup (*C++ function*), 536  
 i2c\_setSpeed (*C++ function*), 536  
 i2c\_write (*C++ function*), 535  
 index (*brainstem.module.Entity property*), 198  
 initiateBulkCapture() (*brainstem.entity.Analog method*), 176  
 ip\_address (*brainstem.link.Spec attribute*), 196  
 isConnected() (*brainstem.module.Module method*), 201

**K**

kelvinSensingOff\_Value (*C macro*), 417  
 kelvinSensingOn\_Value (*C macro*), 417

**L**

link (*brainstem.module.Module property*), 201  
 linkSpec (*C++ struct*), 388  
 linkSpec::model (*C++ member*), 389  
 linkSpec::module (*C++ member*), 389  
 linkSpec::router (*C++ member*), 389  
 linkSpec::router\_serial\_num (*C++ member*), 389  
 linkSpec::serial\_num (*C++ member*), 389  
 linkSpec::t (*C++ member*), 389

```
linkSpec::type (C++ member), 389
linkStatus (C++ enum), 397
linkStatus::INITIALIZING (C++ enumerator), 397
linkStatus::INVALID_LINK_STREAM (C++ enumerator), 397
linkStatus::IO_ERROR (C++ enumerator), 398
linkStatus::RESETTING (C++ enumerator), 398
linkStatus::RUNNING (C++ enumerator), 397
linkStatus::STOPPED (C++ enumerator), 397
linkStatus::STOPPING (C++ enumerator), 397
linkStatus::SYNCING (C++ enumerator), 397
linkStatus::UNKNOWN_ERROR (C++ enumerator), 398
linkType (C++ enum), 388
linkType::INVALID (C++ enumerator), 388
linkType::TCPIP (C++ enumerator), 388
linkType::USB (C++ enumerator), 388
loadSlot () (brainstem.entity.Store method), 246
logEvents () (brainstem.entity.System method), 241
```

## M

```
model (brainstem.link.Spec attribute), 196
model (brainstem.module.Module property), 201
MODEL_ETHERSTEM (in module brainstem.defs), 181
model_info () (in module brainstem.defs), 181
MODEL_MTM_DAQ_1 (in module brainstem.defs), 181
MODEL_MTM_DAQ_2 (in module brainstem.defs), 181
MODEL_MTM_ETHERSTEM (in module brainstem.defs), 181
MODEL_MTM_IOSERIAL (in module brainstem.defs), 181
MODEL_MTM_PM_1 (in module brainstem.defs), 181
MODEL_MTM_RELAY (in module brainstem.defs), 181
MODEL_MTM_USBSTEM (in module brainstem.defs), 181
model_name () (in module brainstem.defs), 182
MODEL_USB_C_SWITCH (in module brainstem.defs), 181
MODEL_USBHUB_2X4 (in module brainstem.defs), 181
MODEL_USBHUB_3C (in module brainstem.defs), 181
MODEL_USBHUB_3P (in module brainstem.defs), 181
MODEL_USBSTEM (in module brainstem.defs), 181
module
    brainstem.defs, 181
    brainstem.discover, 184
    brainstem.entity, 193
    brainstem.link, 195
    brainstem.module, 196
    brainstem.result, 234
    brainstem.version, 260
module (brainstem.link.Spec attribute), 195
module (brainstem.module.Entity property), 198
Module (class in brainstem.module), 199
module_clearAllStems (C++ function), 539
module_connectThroughLinkModule (C++ function), 538
module_createStem (C++ function), 537
module_disconnect (C++ function), 537
module_disconnectAndDestoryStem (C++ function), 537
module_discoverAndConnect (C++ function), 537
module_getModuleAddress (C++ function), 538
module_isConnected (C++ function), 538
module_reconnect (C++ function), 538
module_sDiscover (C++ function), 537
module_setModuleAddress (C++ function), 538
module_setNetworkingMode (C++ function), 539
MTMDAQ1 (class in brainstem.stem), 169
MTMDAQ2 (class in brainstem.stem), 161
MTMEtherStem (class in brainstem.stem), 162
MTMIOSerial (class in brainstem.stem), 163
MTMLOAD1 (class in brainstem.stem), 165
MTMPM1 (class in brainstem.stem), 166
MTMRelay (class in brainstem.stem), 167
MTMUSBStem (class in brainstem.stem), 168
```

Mux (*class in brainstem.entity*), 202  
 mux\_getChannel (*C++ function*), 540  
 mux\_getChannelVoltage (*C++ function*), 540  
 mux\_getConfiguration (*C++ function*), 540  
 mux\_getEnable (*C++ function*), 539  
 mux\_getSplitMode (*C++ function*), 541  
 mux\_setChannel (*C++ function*), 540  
 mux\_setConfiguration (*C++ function*), 541  
 mux\_setEnable (*C++ function*), 539  
 mux\_setSplitMode (*C++ function*), 541  
 muxChannel (*C macro*), 411  
 muxConfig (*C macro*), 411  
 muxConfig\_channelPriority (*C macro*), 411  
 muxConfig\_default (*C macro*), 411  
 muxConfig\_splitMode (*C macro*), 411  
 muxEnable (*C macro*), 411  
 muxSplit (*C macro*), 411  
 muxVoltage (*C macro*), 411

## O

os\_new\_LN (*C macro*), 387

## P

Pointer (*class in brainstem.entity*), 204  
 pointer\_getChar (*C++ function*), 544  
 pointer\_getInt (*C++ function*), 545  
 pointer\_getMode (*C++ function*), 542  
 pointer\_getOffset (*C++ function*), 542  
 pointer\_getShort (*C++ function*), 544  
 pointer\_getTransferStore (*C++ function*), 543  
 pointer\_initiateTransferFromStore (*C++ function*), 543  
 pointer\_initiateTransferToStore (*C++ function*), 543  
 pointer\_setChar (*C++ function*), 544  
 pointer\_setInt (*C++ function*), 545  
 pointer\_setMode (*C++ function*), 542  
 pointer\_setOffset (*C++ function*), 542  
 pointer\_setShort (*C++ function*), 544  
 pointer\_setTransferStore (*C++ function*), 543  
 pointerChar (*C macro*), 412  
 pointerInt (*C macro*), 412  
 pointerMode (*C macro*), 412  
 pointerModeIncrement (*C macro*), 412  
 pointerModeStatic (*C macro*), 412  
 pointerOffset (*C macro*), 412  
 pointerShort (*C macro*), 412  
 pointerTransferFromStore (*C macro*), 412  
 pointerTransferStore (*C macro*), 412  
 pointerTransferToStore (*C macro*), 412  
 Port (*class in brainstem.entity*), 207  
 port\_getAllocatedPower (*C++ function*), 557  
 port\_getAvailablePower (*C++ function*), 557  
 port\_getCC1Enabled (*C++ function*), 553  
 port\_getCC2Enabled (*C++ function*), 554  
 port\_getCCEnabled (*C++ function*), 553  
 port\_getCurrentLimit (*C++ function*), 556  
 port\_getCurrentLimitMode (*C++ function*), 557  
 port\_getDataEnabled (*C++ function*), 547  
 port\_getDataHS1Enabled (*C++ function*), 548  
 port\_getDataHS2Enabled (*C++ function*), 549  
 port\_getDataHSEnabled (*C++ function*), 548  
 port\_getDataHSRoutingBehavior (*C++ function*), 559  
 port\_getDataRole (*C++ function*), 551  
 port\_getDataSpeed (*C++ function*), 555  
 port\_getDataSS1Enabled (*C++ function*), 550  
 port\_getDataSS2Enabled (*C++ function*), 550  
 port\_getDataSSEnabled (*C++ function*), 549  
 port\_getDataSSRoutingBehavior (*C++ function*), 560

port\_getEnabled (*C++ function*), 547  
port\_getErrors (*C++ function*), 556  
port\_getHSBoost (*C++ function*), 561  
port\_getMode (*C++ function*), 555  
port\_getName (*C++ function*), 559  
port\_getPowerEnabled (*C++ function*), 551  
port\_getPowerLimit (*C++ function*), 558  
port\_getPowerLimitMode (*C++ function*), 558  
port\_getPowerMode (*C++ function*), 546  
port\_getState (*C++ function*), 555  
port\_getVbusAccumulatedPower (*C++ function*), 560  
port\_getVbusCurrent (*C++ function*), 546  
port\_getVbusVoltage (*C++ function*), 545  
port\_getVconn1Enabled (*C++ function*), 552  
port\_getVconn2Enabled (*C++ function*), 552  
port\_getVconnAccumulatedPower (*C++ function*), 561  
port\_getVconnCurrent (*C++ function*), 546  
port\_getVconnEnabled (*C++ function*), 551  
port\_getVconnVoltage (*C++ function*), 546  
port\_getVoltageSetpoint (*C++ function*), 554  
port\_resetEntityToFactoryDefaults (*C++ function*), 562  
port\_resetVbusAccumulatedPower (*C++ function*), 560  
port\_resetVconnAccumulatedPower (*C++ function*), 561  
port\_setCC1Enabled (*C++ function*), 554  
port\_setCC2Enabled (*C++ function*), 554  
port\_setCCEnabled (*C++ function*), 553  
port\_setCurrentLimit (*C++ function*), 556  
port\_setCurrentLimitMode (*C++ function*), 557  
port\_setDataEnabled (*C++ function*), 547  
port\_setDataHS1Enabled (*C++ function*), 548  
port\_setDataHS2Enabled (*C++ function*), 549  
port\_setDataHSEnabled (*C++ function*), 548  
port\_setDataHSRoutingBehavior (*C++ function*), 559  
port\_setDataSS1Enabled (*C++ function*), 550  
port\_setDataSS2Enabled (*C++ function*), 550  
port\_setDataSSEnabled (*C++ function*), 549  
port\_setDataSSRoutingBehavior (*C++ function*), 560  
port\_setEnabled (*C++ function*), 547  
port\_setHSBoost (*C++ function*), 561  
port\_setMode (*C++ function*), 556  
port\_setName (*C++ function*), 559  
port\_setPowerEnabled (*C++ function*), 551  
port\_setPowerLimit (*C++ function*), 558  
port\_setPowerLimitMode (*C++ function*), 558  
port\_setPowerMode (*C++ function*), 547  
port\_setVconn1Enabled (*C++ function*), 552  
port\_setVconn2Enabled (*C++ function*), 553  
port\_setVconnEnabled (*C++ function*), 552  
port\_setVoltageSetpoint (*C++ function*), 555  
PowerDelivery (*class in brainstem.entity*), 218  
powerdelivery\_getCableCurrentMax (*C++ function*), 567  
powerdelivery\_getCableOrientation (*C++ function*), 568  
powerdelivery\_getCableSpeedMax (*C++ function*), 568  
powerdelivery\_getCableType (*C++ function*), 568  
powerdelivery\_getCableVoltageMax (*C++ function*), 567  
powerdelivery\_getConnectionState (*C++ function*), 562  
powerdelivery\_getFastRoleSwapCurrent (*C++ function*), 571  
powerdelivery\_getFlagMode (*C++ function*), 570  
powerdelivery\_getNumberOfPowerDataObjects (*C++ function*), 562  
powerdelivery\_getOverride (*C++ function*), 569  
powerdelivery\_getPeakCurrentConfiguration (*C++ function*), 570  
powerdelivery\_getPowerDataObject (*C++ function*), 563  
powerdelivery\_getPowerDataObjectEnabled (*C++ function*), 564  
powerdelivery\_getPowerDataObjectEnabledList (*C++ function*), 565  
powerdelivery\_getPowerDataObjectList (*C++ function*), 564  
powerdelivery\_getPowerRole (*C++ function*), 566  
powerdelivery\_getPowerRolePreferred (*C++ function*), 567  
powerdelivery\_getRequestDataObject (*C++ function*), 565

powerdelivery\_packDataObjectAttributes (*C++ function*), 572  
 powerdelivery\_request (*C++ function*), 568  
 powerdelivery\_requestStatus (*C++ function*), 569  
 powerdelivery\_resetEntityToFactoryDefaults (*C++ function*), 570  
 powerdelivery\_resetPowerDataObjectToDefault (*C++ function*), 563  
 powerdelivery\_setFastRoleSwapCurrent (*C++ function*), 571  
 powerdelivery\_setFlagMode (*C++ function*), 570  
 powerdelivery\_setOverride (*C++ function*), 569  
 powerdelivery\_setPeakCurrentConfiguration (*C++ function*), 571  
 powerdelivery\_setPowerDataObject (*C++ function*), 563  
 powerdelivery\_setPowerDataObjectEnabled (*C++ function*), 565  
 powerdelivery\_setPowerRole (*C++ function*), 566  
 powerdelivery\_setPowerRolePreferred (*C++ function*), 567  
 powerdelivery\_setRequestDataObject (*C++ function*), 566  
 powerdelivery\_unpackDataObjectAttributes (*C++ function*), 572

## R

Rail (*class in brainstem.entity*), 225  
 rail\_clearFaults (*C++ function*), 580  
 rail\_getCurrent (*C++ function*), 573  
 rail\_getCurrentLimit (*C++ function*), 574  
 rail\_getCurrentSetpoint (*C++ function*), 573  
 rail\_getEnable (*C++ function*), 574  
 rail\_getKelvinSensingEnable (*C++ function*), 579  
 rail\_getKelvinSensingState (*C++ function*), 579  
 rail\_getOperationalMode (*C++ function*), 580  
 rail\_getOperationalState (*C++ function*), 580  
 rail\_getPower (*C++ function*), 577  
 rail\_getPowerLimit (*C++ function*), 578  
 rail\_getPowerSetpoint (*C++ function*), 577  
 rail\_getResistance (*C++ function*), 578  
 rail\_getResistanceSetpoint (*C++ function*), 578  
 rail\_getTemperature (*C++ function*), 574  
 rail\_getVoltage (*C++ function*), 575  
 rail\_getVoltageMaxLimit (*C++ function*), 576  
 rail\_getVoltageMinLimit (*C++ function*), 576  
 rail\_getVoltageSetpoint (*C++ function*), 575  
 rail\_setCurrentLimit (*C++ function*), 574  
 rail\_setCurrentSetpoint (*C++ function*), 573  
 rail\_setEnable (*C++ function*), 575  
 rail\_setKelvinSensingEnable (*C++ function*), 579  
 rail\_setOperationalMode (*C++ function*), 579  
 rail\_setPowerLimit (*C++ function*), 577  
 rail\_setPowerSetpoint (*C++ function*), 577  
 rail\_setResistanceSetpoint (*C++ function*), 578  
 rail\_setVoltageMaxLimit (*C++ function*), 576  
 rail\_setVoltageMinLimit (*C++ function*), 576  
 rail\_setVoltageSetpoint (*C++ function*), 575  
 railClearFaults (*C macro*), 420  
 railCurrent (*C macro*), 417  
 railCurrentLimit (*C macro*), 417  
 railCurrentSetpoint (*C macro*), 420  
 railEnable (*C macro*), 417  
 railFactoryReserved (*C macro*), 420  
 railFactoryReserved2 (*C macro*), 421  
 railKelvinSensingEnable (*C macro*), 417  
 railKelvinSensingState (*C macro*), 418  
 railOperationalMode (*C macro*), 418  
 railOperationalMode\_HardwareConfiguration\_Offset (*C macro*), 418  
 railOperationalMode\_Mode\_Offset (*C macro*), 418  
 railOperationalModeAuto\_Value (*C macro*), 418  
 railOperationalModeConstantCurrent\_Value (*C macro*), 418  
 railOperationalModeConstantPower\_Value (*C macro*), 418  
 railOperationalModeConstantResistance\_Value (*C macro*), 418  
 railOperationalModeConstantVoltage\_Value (*C macro*), 418  
 railOperationalModeFactoryReserved\_Value (*C macro*), 418  
 railOperationalModeLinear\_Value (*C macro*), 418

railOperationalModeSwitcher\_Value (*C macro*), 418  
    railOperationalModeSwitcherLinear\_Value (*C macro*), 418  
    railOperationalState (*C macro*), 418  
    railOperationalState\_Enabled\_Bit (*C macro*), 419  
    railOperationalState\_Fault\_Bit (*C macro*), 419  
    railOperationalState\_HardwareConfiguration\_Offset (*C macro*), 419  
    railOperationalState\_Initializing\_Bit (*C macro*), 419  
    railOperationalStateConstantCurrent\_Value (*C macro*), 420  
    railOperationalStateConstantPower\_Value (*C macro*), 420  
    railOperationalStateConstantResistance\_Value (*C macro*), 420  
    railOperationalStateConstantVoltage\_Value (*C macro*), 420  
    railOperationalStateLinear\_Value (*C macro*), 419  
    railOperationalStateOperatingMode\_Offset (*C macro*), 419  
    railOperationalStateOverCurrentFault\_Bit (*C macro*), 419  
    railOperationalStateOverPowerFault\_Bit (*C macro*), 419  
    railOperationalStateOverTemperatureFault\_Bit (*C macro*), 419  
    railOperationalStateOverVoltageFault\_Bit (*C macro*), 419  
    railOperationalStateReverseCurrentFault\_Bit (*C macro*), 419  
    railOperationalStateReversePolarityFault\_Bit (*C macro*), 419  
    railOperationalStateSwitcher\_Value (*C macro*), 419  
    railOperationalStateSwitcherLinear\_Value (*C macro*), 419  
    railOperationalStateUnderVoltageFault\_Bit (*C macro*), 419  
    railPower (*C macro*), 420  
    railPowerLimit (*C macro*), 420  
    railPowerSetpoint (*C macro*), 420  
    railResistance (*C macro*), 420  
    railResistanceSetpoint (*C macro*), 420  
    railTemperature (*C macro*), 417  
    railValue (*C macro*), 417  
    railVoltage (*C macro*), 417  
    railVoltageMaxLimit (*C macro*), 420  
    railVoltageMinLimit (*C macro*), 420  
    railVoltageSetpoint (*C macro*), 420  
    RCServo (*class in brainstem.entity*), 232  
    rcservo\_getEnable (*C++ function*), 581  
    rcservo\_getPosition (*C++ function*), 581  
    rcservo\_getReverse (*C++ function*), 582  
    rcservo\_setEnable (*C++ function*), 581  
    rcservo\_setPosition (*C++ function*), 581  
    rcservo\_setReverse (*C++ function*), 582  
    read() (*brainstem.entity.I2C method*), 194  
    reconnect() (*brainstem.module.Module method*), 201  
    Relay (*class in brainstem.entity*), 233  
    relay\_getEnable (*C++ function*), 583  
    relay\_getVoltage (*C++ function*), 583  
    relay\_setEnable (*C++ function*), 582  
    request() (*brainstem.entity.PowerDelivery method*), 222  
    requestStatus() (*brainstem.entity.PowerDelivery method*), 222  
    reset() (*brainstem.entity.System method*), 241  
    reset() (*brainstem.entity.Temperature method*), 247  
    resetDeviceToFactoryDefaults() (*brainstem.entity.System method*), 241  
    resetEntityToFactoryDefaults() (*brainstem.entity.Port method*), 213  
    resetEntityToFactoryDefaults() (*brainstem.entity.PowerDelivery method*), 223  
    resetEntityToFactoryDefaults() (*brainstem.entity.System method*), 242  
    resetEntityToFactoryDefaults() (*brainstem.entity.Temperature method*), 247  
    resetEntityToFactoryDefaults() (*brainstem.entity.USBSSystem method*), 259  
    resetPowerDataObjectToDefault() (*brainstem.entity.PowerDelivery method*), 223  
    resetVbusAccumulatedPower() (*brainstem.entity.Port method*), 213  
    resetVconnAccumulatedPower() (*brainstem.entity.Port method*), 213  
    Result (*class in brainstem.result*), 234  
    routeToMe() (*brainstem.entity.System method*), 242

## S

    save() (*brainstem.entity.System method*), 242  
    serial\_number (*brainstem.link.Spec attribute*), 195  
    set\_UEI16() (*brainstem.module.Entity method*), 198  
    set\_UEI32() (*brainstem.module.Entity method*), 198

set\_UEI32\_with\_subindex() (*brainstem.module.Entity method*), 199  
set\_UEI8() (*brainstem.module.Entity method*), 199  
set\_UEI8\_with\_subindex() (*brainstem.module.Entity method*), 199  
set\_usbPortStateCOM\_ORIENT\_STATUS (*C macro*), 278  
set\_usbPortStateMUX\_ORIENT\_STATUS (*C macro*), 278  
set\_usbPortStateSPEED\_STATUS (*C macro*), 279  
setAltModeConfig() (*brainstem.entity.USB method*), 253  
setBaudRate() (*brainstem.entity.UART method*), 249  
setBootSlot() (*brainstem.entity.System method*), 242  
setBulkCaptureNumberOfSamples() (*brainstem.entity.Analog method*), 176  
setBulkCaptureSampleRate() (*brainstem.entity.Analog method*), 176  
setCableFlip() (*brainstem.entity.USB method*), 254  
setCC1Enable() (*brainstem.entity.USB method*), 253  
setCC1Enabled() (*brainstem.entity.Port method*), 213  
setCC2Enable() (*brainstem.entity.USB method*), 253  
setCC2Enabled() (*brainstem.entity.Port method*), 214  
setCCEnabled() (*brainstem.entity.Port method*), 214  
setChannel() (*brainstem.entity.Mux method*), 203  
setChar() (*brainstem.entity.Pointer method*), 205  
setConfiguration() (*brainstem.entity.Analog method*), 177  
setConfiguration() (*brainstem.entity.Digital method*), 183  
setConfiguration() (*brainstem.entity.Mux method*), 203  
setConnectMode() (*brainstem.entity.USB method*), 254  
setCurrentLimit() (*brainstem.entity.Port method*), 214  
setCurrentLimit() (*brainstem.entity.Rail method*), 229  
setCurrentLimitMode() (*brainstem.entity.Port method*), 214  
setCurrentSetpoint() (*brainstem.entity.Rail method*), 229  
setDataDisable() (*brainstem.entity.USB method*), 254  
setDataEnable() (*brainstem.entity.USB method*), 254  
setDataEnabled() (*brainstem.entity.Port method*), 214  
setDataHS1Enabled() (*brainstem.entity.Port method*), 214  
setDataHS2Enabled() (*brainstem.entity.Port method*), 215  
setDataHSEnabled() (*brainstem.entity.Port method*), 215  
setDataHSRoutingBehavior() (*brainstem.entity.Port method*), 215  
setDataRoleBehavior() (*brainstem.entity.USBSystem method*), 259  
setDataRoleBehaviorConfig() (*brainstem.entity.USBSystem method*), 259  
setDataSS1Enabled() (*brainstem.entity.Port method*), 215  
setDataSS2Enabled() (*brainstem.entity.Port method*), 215  
setDataSSEnabled() (*brainstem.entity.Port method*), 216  
setDataSSRoutingBehavior() (*brainstem.entity.Port method*), 216  
setDay() (*brainstem.entity.Clock method*), 179  
setDownstreamBoostMode() (*brainstem.entity.USB method*), 254  
setEnabled() (*brainstem.entity.Analog method*), 177  
setEnabled() (*brainstem.entity.Mux method*), 203  
setEnabled() (*brainstem.entity.Rail method*), 230  
setEnabled() (*brainstem.entity.RCServo method*), 232  
setEnabled() (*brainstem.entity.Relay method*), 234  
setEnabled() (*brainstem.entity.Signal method*), 235  
setEnabled() (*brainstem.entity.UART method*), 249  
setEnabled() (*brainstem.entity.Port method*), 216  
setEnabledList() (*brainstem.entity.USBSystem method*), 259  
setEnumerationDelay() (*brainstem.entity.USB method*), 254  
setEnumerationDelay() (*brainstem.entity.USBSystem method*), 259  
setExpiration() (*brainstem.entity.Timer method*), 248  
setFastRoleSwapCurrent() (*brainstem.entity.PowerDelivery method*), 223  
setFlagMode() (*brainstem.entity.PowerDelivery method*), 223  
setHBInterval() (*brainstem.entity.System method*), 242  
setHiSpeedDataDisable() (*brainstem.entity.USB method*), 255  
setHiSpeedDataEnable() (*brainstem.entity.USB method*), 255  
setHour() (*brainstem.entity.Clock method*), 180  
setHSBoost() (*brainstem.entity.Port method*), 216  
setHubMode() (*brainstem.entity.USB method*), 255  
setInputPowerBehavior() (*brainstem.entity.System method*), 243  
setInputPowerBehaviorConfig() (*brainstem.entity.System method*), 243  
setInt() (*brainstem.entity.Pointer method*), 205  
setInvert() (*brainstem.entity.Signal method*), 235  
setKelvinSensingEnable() (*brainstem.entity.Rail method*), 230  
setLED() (*brainstem.entity.System method*), 243

setLinkInterface() (*brainstem.entity.System method*), 243  
setMinute() (*brainstem.entity.Clock method*), 180  
setMode() (*brainstem.entity.Pointer method*), 205  
setMode() (*brainstem.entity.Port method*), 216  
setMode() (*brainstem.entity.Timer method*), 248  
setModeList() (*brainstem.entity.USBSYSTEM method*), 259  
setModuleAddress() (*brainstem.module.Module method*), 201  
setModuleSoftwareOffset() (*brainstem.entity.System method*), 243  
setMonth() (*brainstem.entity.Clock method*), 180  
setName() (*brainstem.entity.Port method*), 216  
setName() (*brainstem.entity.System method*), 244  
setNetworkingMode() (*brainstem.module.Module method*), 201  
setOffset() (*brainstem.entity.Pointer method*), 206  
setOperationalMode() (*brainstem.entity.Rail method*), 230  
setOverride() (*brainstem.entity.PowerDelivery method*), 223  
setPeakCurrentConfiguration() (*brainstem.entity.PowerDelivery method*), 223  
setPortCurrentLimit() (*brainstem.entity.USB method*), 255  
setPortDisable() (*brainstem.entity.USB method*), 255  
setPortEnable() (*brainstem.entity.USB method*), 255  
setPortMode() (*brainstem.entity.USB method*), 255  
setPosition() (*brainstem.entity.RCServo method*), 233  
setPowerBehavior() (*brainstem.entity.USBSYSTEM method*), 260  
setPowerBehaviorConfig() (*brainstem.entity.USBSYSTEM method*), 260  
setPowerDataObject() (*brainstem.entity.PowerDelivery method*), 224  
setPowerDataObjectEnabled() (*brainstem.entity.PowerDelivery method*), 224  
setPowerDisable() (*brainstem.entity.USB method*), 256  
setPowerEnable() (*brainstem.entity.USB method*), 256  
setPowerEnabled() (*brainstem.entity.Port method*), 216  
setPowerLimit() (*brainstem.entity.Port method*), 217  
setPowerLimit() (*brainstem.entity.Rail method*), 230  
setPowerLimitMax() (*brainstem.entity.System method*), 244  
setPowerLimitMode() (*brainstem.entity.Port method*), 217  
setPowerMode() (*brainstem.entity.Port method*), 217  
setPowerRole() (*brainstem.entity.PowerDelivery method*), 224  
setPowerRolePreferred() (*brainstem.entity.PowerDelivery method*), 224  
setPowerSetpoint() (*brainstem.entity.Rail method*), 230  
setProtocol() (*brainstem.entity.UART method*), 249  
setPullup() (*brainstem.entity.I2C method*), 194  
setRange() (*brainstem.entity.Analog method*), 177  
setReceiverConfig() (*brainstem.entity.Equalizer method*), 193  
setRequestDataObject() (*brainstem.entity.PowerDelivery method*), 224  
setResistanceSetpoint() (*brainstem.entity.Rail method*), 231  
setReverse() (*brainstem.entity.RCServo method*), 233  
setRouter() (*brainstem.entity.System method*), 244  
setsSBEEnable() (*brainstem.entity.USB method*), 256  
setSecond() (*brainstem.entity.Clock method*), 180  
setSelectorMode() (*brainstem.entity.USBSYSTEM method*), 260  
setShort() (*brainstem.entity.Pointer method*), 206  
setSlotLocked() (*brainstem.entity.Store method*), 246  
setSpeed() (*brainstem.entity.I2C method*), 194  
setSplitMode() (*brainstem.entity.Mux method*), 204  
setState() (*brainstem.entity.Digital method*), 183  
setStateAll() (*brainstem.entity.Digital method*), 183  
setSuperSpeedDataDisable() (*brainstem.entity.USB method*), 256  
setSuperSpeedDataEnable() (*brainstem.entity.USB method*), 256  
setT2Time() (*brainstem.entity.Signal method*), 235  
setT3Time() (*brainstem.entity.Signal method*), 236  
setTransferStore() (*brainstem.entity.Pointer method*), 206  
setTransmitterConfig() (*brainstem.entity.Equalizer method*), 193  
setUpstream() (*brainstem.entity.USBSYSTEM method*), 260  
setUpstreamBoostMode() (*brainstem.entity.USB method*), 256  
setUpstreamMode() (*brainstem.entity.USB method*), 257  
setValue() (*brainstem.entity.Analog method*), 177  
setVconn1Enabled() (*brainstem.entity.Port method*), 217  
setVconn2Enabled() (*brainstem.entity.Port method*), 217  
setVconnEnabled() (*brainstem.entity.Port method*), 217  
setVoltage() (*brainstem.entity.Analog method*), 177  
setVoltageMaxLimit() (*brainstem.entity.Rail method*), 231

setVoltageMinLimit () (*brainstem.entity.Rail method*), 231  
setVoltageSetpoint () (*brainstem.entity.Port method*), 218  
setVoltageSetpoint () (*brainstem.entity.Rail method*), 231  
setYear () (*brainstem.entity.Clock method*), 180  
Signal (*class in brainstem.entity*), 235  
signal\_getEnable (*C++ function*), 583  
signal\_getInvert (*C++ function*), 584  
signal\_getT2Time (*C++ function*), 585  
signal\_getT3Time (*C++ function*), 585  
signal\_setEnable (*C++ function*), 583  
signal\_setInvert (*C++ function*), 584  
signal\_setT2Time (*C++ function*), 585  
signal\_setT3Time (*C++ function*), 584  
slotCapacity (*C macro*), 410  
slotClose (*C macro*), 410  
slotDisable () (*brainstem.entity.Store method*), 246  
slotEnable () (*brainstem.entity.Store method*), 246  
slotOpenRead (*C macro*), 410  
slotOpenWrite (*C macro*), 410  
slotRead (*C macro*), 410  
slotSeek (*C macro*), 410  
slotSize (*C macro*), 410  
slotWrite (*C macro*), 410  
spec (*brainstem.module.Module property*), 202  
Spec (*class in brainstem.link*), 195  
Status (*class in brainstem.link*), 196  
Store (*class in brainstem.entity*), 245  
store\_getSlotCapacity (*C++ function*), 587  
store\_getSlotLocked (*C++ function*), 588  
store\_getSlotSize (*C++ function*), 587  
store\_getSlotState (*C++ function*), 586  
store\_loadSlot (*C++ function*), 586  
store\_setSlotLocked (*C++ function*), 588  
store\_slotDisable (*C++ function*), 587  
store\_slotEnable (*C++ function*), 586  
store\_unloadSlot (*C++ function*), 586  
storeCloseSlot (*C macro*), 423  
storeLock (*C macro*), 423  
storeNumberOfOptions (*C macro*), 423  
storeReadSlot (*C macro*), 423  
storeSlotDisable (*C macro*), 423  
storeSlotEnable (*C macro*), 423  
storeSlotState (*C macro*), 423  
storeWriteSlot (*C macro*), 423  
System (*class in brainstem.entity*), 236  
system\_getBootSlot (*C++ function*), 590  
system\_getHardwareVersion (*C++ function*), 591  
system\_getHBInterval (*C++ function*), 589  
system\_getInputCurrent (*C++ function*), 593  
system\_getInputPowerBehavior (*C++ function*), 596  
system\_getInputPowerBehaviorConfig (*C++ function*), 597  
system\_getInputPowerSource (*C++ function*), 596  
system\_getInputVoltage (*C++ function*), 593  
system\_getLED (*C++ function*), 590  
system\_getLinkInterface (*C++ function*), 598  
system\_getMaximumTemperature (*C++ function*), 593  
system\_getMinimumTemperature (*C++ function*), 592  
system\_getModel (*C++ function*), 591  
system\_getModule (*C++ function*), 588  
system\_getModuleBaseAddress (*C++ function*), 589  
system\_getModuleHardwareOffset (*C++ function*), 593  
system\_getModuleSoftwareOffset (*C++ function*), 594  
system\_getName (*C++ function*), 597  
system\_getPowerLimit (*C++ function*), 595  
system\_getPowerLimitMax (*C++ function*), 595  
system\_getPowerLimitState (*C++ function*), 595  
system\_getRouter (*C++ function*), 589  
system\_getRouterAddressSetting (*C++ function*), 594

system\_getSerialNumber (*C++ function*), 591  
system\_getTemperature (*C++ function*), 592  
system\_getUnregulatedCurrent (*C++ function*), 596  
system\_getUnregulatedVoltage (*C++ function*), 595  
system\_getUptime (*C++ function*), 592  
system\_getVersion (*C++ function*), 591  
system\_logEvents (*C++ function*), 592  
system\_reset (*C++ function*), 592  
system\_resetDeviceToFactoryDefaults (*C++ function*), 598  
system\_resetEntityToFactoryDefaults (*C++ function*), 598  
system\_routeToMe (*C++ function*), 594  
system\_save (*C++ function*), 591  
system\_setBootSlot (*C++ function*), 590  
system\_setHBIInterval (*C++ function*), 589  
system\_setInputPowerBehavior (*C++ function*), 596  
system\_setInputPowerBehaviorConfig (*C++ function*), 597  
system\_setLED (*C++ function*), 590  
system\_setLinkInterface (*C++ function*), 598  
system\_setModuleSoftwareOffset (*C++ function*), 594  
system\_setName (*C++ function*), 597  
system\_setPowerLimitMax (*C++ function*), 595  
system\_setRouter (*C++ function*), 589  
systemBootSlot (*C macro*), 406  
systemErrors (*C macro*), 409  
systemErrors\_OutputPowerProtection\_Bit (*C macro*), 409  
systemErrors\_ThermalProtection\_Bit (*C macro*), 409  
systemHardwareVersion (*C macro*), 409  
systemHBIInterval (*C macro*), 406  
systemInputCurrent (*C macro*), 407  
systemInputPowerBehavior (*C macro*), 408  
systemInputPowerBehaviorConfig (*C macro*), 408  
systemInputPowerSource (*C macro*), 408  
systemInputVoltage (*C macro*), 406  
systemIPAddress (*C macro*), 407  
systemIPConfiguration (*C macro*), 407  
systemIPModeDefault (*C macro*), 407  
systemIPModeDHCP (*C macro*), 407  
systemIPModeStatic (*C macro*), 407  
systemIPStaticAddressSetting (*C macro*), 407  
systemLED (*C macro*), 406  
systemLinkAuto (*C macro*), 408  
systemLinkInterface (*C macro*), 408  
systemLinkUSBControl (*C macro*), 408  
systemLinkUSBHub (*C macro*), 409  
systemLogEvents (*C macro*), 407  
systemMaxTemperature (*C macro*), 407  
systemMinTemperature (*C macro*), 408  
systemModel (*C macro*), 406  
systemModule (*C macro*), 406  
systemModuleBaseAddress (*C macro*), 407  
systemModuleHardwareOffset (*C macro*), 407  
systemModuleSoftwareOffset (*C macro*), 407  
systemName (*C macro*), 408  
systemNumberOfOptions (*C macro*), 409  
systemPowerLimit (*C macro*), 408  
systemPowerLimitMax (*C macro*), 408  
systemPowerLimitState (*C macro*), 408  
systemReserved (*C macro*), 409  
systemReset (*C macro*), 406  
systemResetDeviceToFactoryDefaults (*C macro*), 408  
systemResetEntityToFactoryDefaults (*C macro*), 408  
systemRouter (*C macro*), 406  
systemRouterAddressSetting (*C macro*), 407  
systemRouteToMe (*C macro*), 407  
systemSave (*C macro*), 406  
systemSerialNumber (*C macro*), 406  
systemSleep (*C macro*), 406  
systemTemperature (*C macro*), 408

systemUnregulatedCurrent (*C macro*), 408  
 systemUnregulatedVoltage (*C macro*), 407  
 systemUptime (*C macro*), 407  
 systemVersion (*C macro*), 406

**T**

tcp\_port (*brainstem.link.Spec attribute*), 196  
 TCPIP (*brainstem.link.Spec attribute*), 196  
 Temperature (*class in brainstem.entity*), 246  
 temperature\_getValue (*C++ function*), 599  
 temperature\_getValueMax (*C++ function*), 599  
 temperature\_getValueMin (*C++ function*), 599  
 temperature\_resetEntityToFactoryDefaults (*C++ function*), 599  
 temperatureMaximumMicroCelsius (*C macro*), 421  
 temperatureMicroCelsius (*C macro*), 421  
 temperatureMinimumMicroCelsius (*C macro*), 421  
 temperatureNumberOfOptions (*C macro*), 421  
 temperatureResetEntityToFactoryDefaults (*C macro*), 421  
 Timer (*class in brainstem.entity*), 247  
 timer\_getExpiration (*C++ function*), 600  
 timer\_getMode (*C++ function*), 600  
 timer\_setExpiration (*C++ function*), 600  
 timer\_setMode (*C++ function*), 600  
 timerExpiration (*C macro*), 424  
 timerMode (*C macro*), 424  
 timerModeRepeat (*C macro*), 424  
 timerModeSingle (*C macro*), 424  
 transferToStore () (*brainstem.entity.Pointer method*), 206  
 transport (*brainstem.link.Spec attribute*), 195  
 transterFromStore () (*brainstem.entity.Pointer method*), 206

**U**

UART (*class in brainstem.entity*), 248  
 uart\_getEnable (*C++ function*), 601  
 uart\_setEnable (*C++ function*), 601  
 uei (*C++ struct*), 443  
 uei::byteVal (*C++ member*), 444  
 uei::command (*C++ member*), 444  
 uei::intVal (*C++ member*), 444  
 uei::module (*C++ member*), 444  
 uei::option (*C++ member*), 444  
 uei::shortVal (*C++ member*), 444  
 uei::specifier (*C++ member*), 444  
 uei::type (*C++ member*), 444  
 ueiBYTES\_CONTINUE (*C macro*), 405  
 ueiBYTES\_CONTINUE\_MASK (*C macro*), 405  
 ueiOPTION\_ACK (*C macro*), 405  
 ueiOPTION\_GET (*C macro*), 405  
 ueiOPTION\_MASK (*C macro*), 405  
 ueiOPTION\_OP\_MASK (*C macro*), 405  
 ueiOPTION\_SET (*C macro*), 405  
 ueiOPTION\_VAL (*C macro*), 405  
 ueiREPLY\_ERROR (*C macro*), 405  
 ueiREPLY\_STREAM (*C macro*), 405  
 ueiSPECIFIER\_INDEX\_MASK (*C macro*), 404  
 ueiSPECIFIER\_RETURN\_HOST (*C macro*), 404  
 ueiSPECIFIER\_RETURN\_I2C (*C macro*), 404  
 ueiSPECIFIER\_RETURN\_MASK (*C macro*), 404  
 ueiSPECIFIER\_RETURN\_VM (*C macro*), 405  
 unloadSlot () (*brainstem.entity.Store method*), 246  
 unpack\_version () (*in module brainstem.version*), 260  
 USB (*brainstem.link.Spec attribute*), 196  
 USB (*class in brainstem.entity*), 250  
 usb\_clearPortErrorStatus (*C++ function*), 605  
 usb\_getAltModeConfig (*C++ function*), 613  
 usb\_getCableFlip (*C++ function*), 613  
 usb\_getCC1Current (*C++ function*), 611

usb\_getCC1Enable (*C++ function*), 610  
usb\_getCC1Voltage (*C++ function*), 611  
usb\_getCC2Current (*C++ function*), 611  
usb\_getCC2Enable (*C++ function*), 611  
usb\_getCC2Voltage (*C++ function*), 611  
usb\_getConnectMode (*C++ function*), 609  
usb\_getDownstreamBoostMode (*C++ function*), 608  
usb\_getDownstreamDataSpeed (*C++ function*), 609  
usb\_getEnumerationDelay (*C++ function*), 606  
usb\_getHubMode (*C++ function*), 604  
usb\_getPortCurrent (*C++ function*), 604  
usb\_getPortCurrentLimit (*C++ function*), 606  
usb\_getPortError (*C++ function*), 607  
usb\_getPortMode (*C++ function*), 607  
usb\_getPortState (*C++ function*), 607  
usb\_getPortVoltage (*C++ function*), 604  
usb\_getSBU1Voltage (*C++ function*), 613  
usb\_getSBU2Voltage (*C++ function*), 614  
usb\_getSBUEnable (*C++ function*), 612  
usb\_getUpstreamBoostMode (*C++ function*), 608  
usb\_getUpstreamMode (*C++ function*), 605  
usb\_getUpstreamState (*C++ function*), 605  
usb\_setAltModeConfig (*C++ function*), 613  
usb\_setCableFlip (*C++ function*), 612  
usb\_setCC1Enable (*C++ function*), 610  
usb\_setCC2Enable (*C++ function*), 610  
usb\_setConnectMode (*C++ function*), 609  
usb\_setDataDisable (*C++ function*), 602  
usb\_setDataEnable (*C++ function*), 602  
usb\_setDownstreamBoostMode (*C++ function*), 608  
usb\_setEnumerationDelay (*C++ function*), 606  
usb\_setHiSpeedDataDisable (*C++ function*), 603  
usb\_setHiSpeedDataEnable (*C++ function*), 602  
usb\_setHubMode (*C++ function*), 604  
usb\_setPortCurrentLimit (*C++ function*), 606  
usb\_setPortDisable (*C++ function*), 602  
usb\_setPortEnable (*C++ function*), 602  
usb\_setPortMode (*C++ function*), 607  
usb\_setPowerDisable (*C++ function*), 604  
usb\_setPowerEnable (*C++ function*), 603  
usb\_setSBUEnable (*C++ function*), 612  
usb\_setSuperSpeedDataDisable (*C++ function*), 603  
usb\_setSuperSpeedDataEnable (*C++ function*), 603  
usb\_setUpstreamBoostMode (*C++ function*), 608  
usb\_setUpstreamMode (*C++ function*), 605  
usbAltMode (*C macro*), 430  
usbAltMode\_2LaneDP\_ComToHost\_wUSB3 (*C macro*), 430  
usbAltMode\_2LaneDP\_ComToHost\_wUSB3\_Inverted (*C macro*), 431  
usbAltMode\_2LaneDP\_MuxToHost\_wUSB3 (*C macro*), 430  
usbAltMode\_2LaneDP\_MuxToHost\_wUSB3\_Inverted (*C macro*), 431  
usbAltMode\_4LaneDP\_ComToHost (*C macro*), 430  
usbAltMode\_4LaneDP\_MuxToHost (*C macro*), 430  
usbAltMode\_disabled (*C macro*), 430  
usbAltMode\_normal (*C macro*), 430  
usbAutoConnect (*C macro*), 429  
usbBoostMode\_0 (*C macro*), 427  
usbBoostMode\_12 (*C macro*), 427  
usbBoostMode\_4 (*C macro*), 427  
usbBoostMode\_8 (*C macro*), 427  
usbCableFlip (*C macro*), 430  
usbCC1Current (*C macro*), 430  
usbCC1Enable (*C macro*), 429  
usbCC1Voltage (*C macro*), 430  
usbCC2Current (*C macro*), 430  
usbCC2Enable (*C macro*), 429  
usbCC2Voltage (*C macro*), 430  
usbConnectMode (*C macro*), 429  
USBCSwitch (*class in brainstem.stem*), 158

usbDataDisable (*C macro*), 425  
usbDataEnable (*C macro*), 425  
usbDownstreamBoostMode (*C macro*), 427  
usbDownstreamDataSpeed (*C macro*), 429  
usbDownstreamDataSpeed\_hs (*C macro*), 429  
usbDownstreamDataSpeed\_ls (*C macro*), 429  
usbDownstreamDataSpeed\_na (*C macro*), 429  
usbDownstreamDataSpeed\_ss (*C macro*), 429  
usbHiSpeedDataDisable (*C macro*), 429  
usbHiSpeedDataEnable (*C macro*), 429  
USBHub2x4 (*class in brainstem.stem*), 157  
USBHub3p (*class in brainstem.stem*), 155  
usbHubEnumerationDelay (*C macro*), 427  
usbHubMode (*C macro*), 426  
usbManualConnect (*C macro*), 429  
usbPortClearErrorStatus (*C macro*), 426  
usbPortCurrent (*C macro*), 426  
usbPortCurrentLimit (*C macro*), 427  
usbPortDisable (*C macro*), 425  
usbPortEnable (*C macro*), 425  
usbPortError (*C macro*), 430  
usbPortMode (*C macro*), 427  
usbPortMode\_AutoConnectEnable (*C macro*), 428  
usbPortMode\_CC1Enable (*C macro*), 428  
usbPortMode\_CC1InjectEnable (*C macro*), 428  
usbPortMode\_CC2Enable (*C macro*), 428  
usbPortMode\_CC2InjectEnable (*C macro*), 429  
usbPortMode\_CCFlipEnable (*C macro*), 428  
usbPortMode\_cdp (*C macro*), 427  
usbPortMode\_charging (*C macro*), 427  
usbPortMode\_passive (*C macro*), 427  
usbPortMode\_SBUEnable (*C macro*), 428  
usbPortMode\_SBUFlipEnable (*C macro*), 428  
usbPortMode\_sdp (*C macro*), 427  
usbPortMode\_SSFlipEnable (*C macro*), 428  
usbPortMode\_SuperSpeed1Enable (*C macro*), 428  
usbPortMode\_SuperSpeed2Enable (*C macro*), 428  
usbPortMode\_USB2AEnable (*C macro*), 427  
usbPortMode\_USB2BEnable (*C macro*), 428  
usbPortMode\_USB2BoostEnable (*C macro*), 428  
usbPortMode\_USB2FlipEnable (*C macro*), 428  
usbPortMode\_USB3BoostEnable (*C macro*), 428  
usbPortMode\_VBusEnable (*C macro*), 428  
usbPortState (*C macro*), 430  
usbPortStateCC1 (*C macro*), 278  
usbPortStateCC1Detect (*C macro*), 279  
usbPortStateCC1Inject (*C macro*), 279  
usbPortStateCC1LogicState (*C macro*), 280  
usbPortStateCC2 (*C macro*), 278  
usbPortStateCC2Detect (*C macro*), 279  
usbPortStateCC2Inject (*C macro*), 279  
usbPortStateCC2LogicState (*C macro*), 280  
usbPortStateCCFlip (*C macro*), 279  
usbPortStateConnectionEstablished (*C macro*), 279  
usbPortStateErrorFlag (*C macro*), 279  
usbPortStateOff (*C macro*), 280  
usbPortStateSBU (*C macro*), 278  
usbPortStateSBUFlip (*C macro*), 279  
usbPortStateSideA (*C macro*), 280  
usbPortStateSideB (*C macro*), 280  
usbPortStateSideUndefined (*C macro*), 280  
usbPortStateSS1 (*C macro*), 278  
usbPortStateSS2 (*C macro*), 278  
usbPortStateSSFlip (*C macro*), 279  
usbPortStateUSB2A (*C macro*), 278  
usbPortStateUSB2B (*C macro*), 278  
usbPortStateUSB2Boost (*C macro*), 279  
usbPortStateUSB2Flip (*C macro*), 279

usbPortStateUSB3Boost (*C macro*), 279  
usbPortStateVBUS (*C macro*), 278  
usbPortVoltage (*C macro*), 426  
usbPowerDisable (*C macro*), 426  
usbPowerEnable (*C macro*), 426  
usbSBU1Voltage (*C macro*), 431  
usbSBU2Voltage (*C macro*), 431  
usbSBUEnable (*C macro*), 430  
USBStem (*class in brainstem.stem*), 171  
usbSuperSpeedDataDisable (*C macro*), 429  
usbSuperSpeedDataEnable (*C macro*), 429  
USBSystem (*class in brainstem.entity*), 257  
usbsystem\_getDataRoleBehavior (*C++ function*), 618  
usbsystem\_getDataRoleBehaviorConfig (*C++ function*), 618  
usbsystem\_getDataRoleList (*C++ function*), 615  
usbsystem\_getEnabledList (*C++ function*), 615  
usbsystem\_getEnumerationDelay (*C++ function*), 614  
usbsystem\_getModeList (*C++ function*), 616  
usbsystem\_getPowerBehavior (*C++ function*), 617  
usbsystem\_getPowerBehaviorConfig (*C++ function*), 617  
usbsystem\_getStateList (*C++ function*), 616  
usbsystem\_getUpstream (*C++ function*), 614  
usbsystem\_resetEntityToFactoryDefaults (*C++ function*), 619  
usbsystem\_setDataRoleBehavior (*C++ function*), 618  
usbsystem\_setDataRoleBehaviorConfig (*C++ function*), 619  
usbsystem\_setEnabledList (*C++ function*), 615  
usbsystem\_setEnumerationDelay (*C++ function*), 615  
usbsystem\_setModeList (*C++ function*), 616  
usbsystem\_setPowerBehavior (*C++ function*), 617  
usbsystem\_setPowerBehaviorConfig (*C++ function*), 617  
usbsystem\_setUpstream (*C++ function*), 614  
usbUpstreamBoostMode (*C macro*), 427  
usbUpstreamMode (*C macro*), 426  
usbUpstreamModeAuto (*C macro*), 426  
usbUpstreamModeDefault (*C macro*), 426  
usbUpstreamModeNone (*C macro*), 426  
usbUpstreamModePort0 (*C macro*), 426  
usbUpstreamModePort1 (*C macro*), 426  
usbUpstreamState (*C macro*), 426  
usbUpstreamStateNone (*C macro*), 426  
usbUpstreamStatePort0 (*C macro*), 426  
usbUpstreamStatePort1 (*C macro*), 427

## V

VALIDPACKET (*C++ member*), 402  
value (*brainstem.result.Result property*), 234

## W

write () (*brainstem.entity.I2C method*), 194