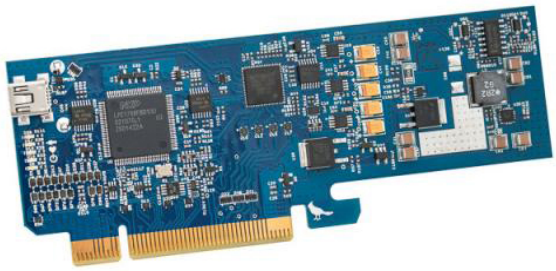


Overview

The Acroname® MTM-IO-Serial module (S62-MTM-IO-SERIAL), as part of Acroname's MTM (Manufacturing Test Module) product series, is a software-controlled USB 2.0 hub, designed for MTM-based manufacturing or R&D test systems. The MTM-IO-Serial allows MTM system designers to easily and modularly add USB 2.0 connectivity as well as serial UART and GPIO functions at two variable IO levels.

Built using Acroname's industry-proven and well-adopted BrainStem® technology, resources on the MTM-IO-Serial are controlled via Acroname's powerful and extensible BrainStem technology and software APIs.

Typical Application

- Manufacturing functional testing
- Validation testing
- Automated test development
- Embedded system development
- Firmware loading
- Serial communications
- USB switching and control

System Features

- 4 downstream USB 2.0 ports, software controlled Hi-Z disable
- 2 independent 1.8-5.0V adjustable high-current voltage rails, current limited to 150mA
- 4 adjustable-voltage serial UART ports (2 per adjustable rail) software controlled Hi-Z disable
- 4 adjustable-voltage digital GPIO (4 per adjustable rail)
- All GPIOs overvoltage and overcurrent protected
- 1 fixed high-current 5.0V output, current limited to 150mA
- 1 downstream USB 2.0 port on edge connector, always on for daisy-chaining multiple modules over USB
- 1 downstream USB 2.0 port type-A connector, always on for daisy-chaining multiple modules over USB
- 1 BrainStem I²C FM+ (1Mbit/s) bus

Description

As part of Acroname's MTM series, the MTM-IO-Serial module is a key component for manufacturing test for electronic devices using a USB 2.0 interface, serial UART and/or one or more IO interface voltages. Details on the MTM development platform architecture, BrainStem interface, and APIs are at https://acroname.com/reference.

The MTM-IO-Serial implements an onboard BrainStem controller running an RTOS (Real-Time Operating System), which provides a USB host connection, independent operating capability and the BrainStem interface, for control of the MTM resources identified in this datasheet (Rail0, Rail1, GPIO, UART, USB CHx, etc.).

The MTM-IO-Serial provides three key functions via the BrainStem API interface: (1) a 4-port programmable USB 2.0 hub with ability to separate enable/disable capability for data lines and Vbus connections; (2) two adjustable IO rails; (3) two banks of UART interfaces and GPIO pins, both of which are associated with an adjustable IO rail.

The serial UARTs are placed into two groups of independent, software-adjustable voltage rails with associated digital input/output (DIO) pins with logic levels based on rail voltage. This configuration allows the MTM-IO-Serial module UARTs and GPIOs to be used to interface to DUTs with two different voltage planes.

Within the MTM platform architecture, the MTM-IO-Serial module can operate either independently or as a component in a larger network of MTM modules. Each MTM-IO-Serial is uniquely addressable and controllable from a host by connecting via the onboard USB connection, the card-edge USB input or through other MTM modules on the local MTM/BrainStem I2C bus.

Acroname's BrainStem link is established over the selected input connection. The BrainStem link allows a connection to the onboard controller and access to the available resources in the MTM-IO-Serial. The MTM-IO-Serial can then be controlled via a host running BrainStem APIs or it can operate independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS.

IMPORTANT NOTE

The MTM-IO-SERIAL utilizes a PCIe connector interface but is for use strictly in MTM-based systems. It should never be installed in a PCI slot of a host computer directly. Insertion into a PC or non-MTM system could cause damage to the PC.

Absolute Maximum Ratings

Stresses beyond those listed under ABSOLUTE MAXIMUM RATINGS cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under RECOMMENDED OPERATING CONDITIONS is not implied. Exposure to absolute-maximum-rated conditions for extended periods affects device reliability and may permanently damage the device.

Voltage Rating	Minimum	Maximum	Units
Input Voltage, V_{supply}	-14.7	14.7	V
I2C0 SDA, SCL	-0.5	14.7	V
UART TX/RX	-0.5	6.5	V
DIO 0-7	-0.5	6.5	V
Module Address 0-3	-0.5	14.7	V
Reset	-0.5	14.7	V
USB D+, D-	-0.5	5.5	V
USB V_{bus}	-0.5	6.0	V
Rail 0-2	-0.5	14.7	V

Table 1: Absolute Maximum Ratings

Current Rating	Minimum	Maximum	Units
Input Current, I_{supply}	0.0	5.5	A

Table 2: Absolute Maximum Current Ratings

The MTM system is designed to be used in a system where V_{supply} is the highest voltage connected to all MTM modules. Each module is designed to withstand V_{supply} continuously connected to all IOs, excepting those specified above, including accidental reverse polarity connection between V_{supply} and ground (0V). As with all products, care should be taken to properly match interface voltages and use a well architected current-return path to ground for the targeted application.

Handling Ratings

Parameter	Conditions/Notes	Minimum	Typical	Maximum	Units
Ambient Operating Temperature, T_A	Non-Condensing	0	25	70	°C
Relative Humidity Range	Non-Condensing	0	-	95	%RH
Storage Temperature, T_{STG}		-10	-	+85	°C
Electrostatic Discharge, V_{ESD}	IEC 61000-4-2, level 4, contact discharge to edge connector interface	-8	-	+8	kV

Table 3: Handling Ratings

Recommended Operating Ratings

Specifications are valid at 25°C unless otherwise noted. Indoor application use only.

Parameter	Conditions/Notes	Minimum	Typical	Maximum	Units
Input Voltage, V_{supply}		6.0	-	12.0	V
Voltage to any IO pin		0	-	3.3	V
Voltage to any I2C pin		0	-	3.3	V
Relative Humidity Range	Non-Condensing	5	-	95	%RH

Table 4: Recommended Operating Ratings

Block Diagram

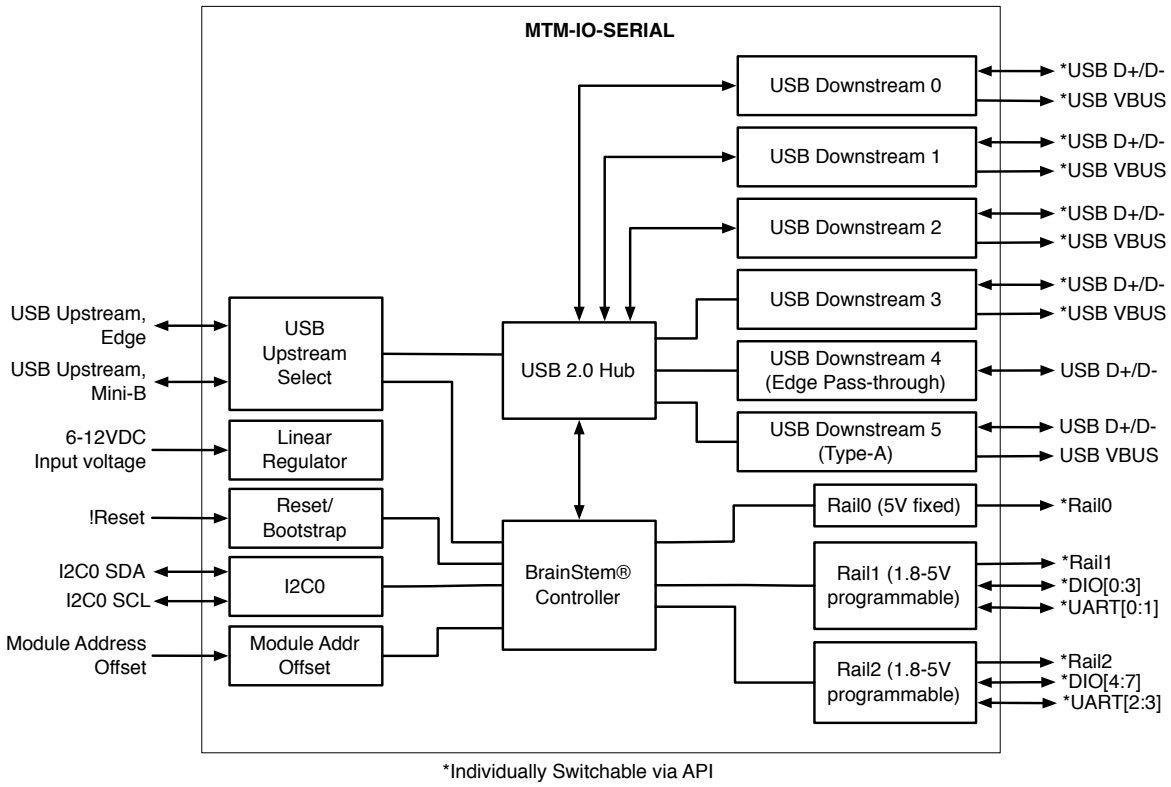


Figure 1: MTM-IO-Serial Block Diagram

Typical Performance Characteristics

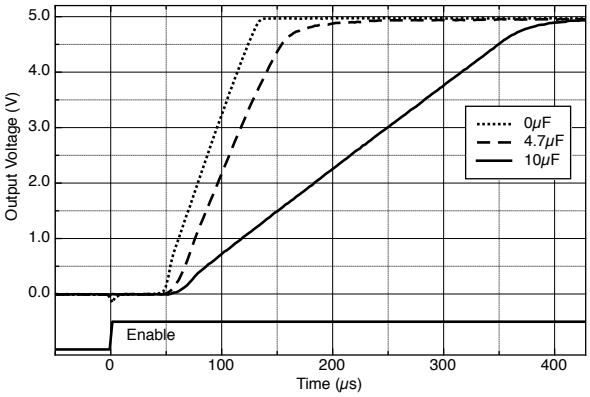
Specifications are valid at 25°C unless otherwise noted. Indoor application use only. Sample rates are typically limited by the USB throughput of the host operating system except where bulk capture is supported.

Parameter	Conditions/Notes	Minimum	Typical	Maximum	Units
Base Current Consumption, I_{supply}	V_{supply} = 6V V_{supply} = 12V	- -	135 343	- -	mA
Reset Low Threshold		-	1.2	-	V
I2C SDA, SCL Pins		-	3.3	-	V
UART Tx/Rx Logic High, V_{IH}		0.65 × V_{rail}	-	-	V
UART Tx/Rx Logic Low, V_{IL}		-	-	0.35 × V_{rail}	V
Digital Input Logic High, V_{IH}		0.65 × V_{rail}	-	-	V
Digital Input Logic Low, V_{IL}		-	-	0.35 × V_{rail}	V
Digital Input Resistance	Internal Pull-down	-	100	-	kΩ
Rail 0 Output Voltage, V_{rail0}	±2%	4.9	5.0	5.1	V
Rail 1-2 Output Voltage, V_{rail1}, V_{rail2}	Software controlled, ±2%	1.764	-	5.1V	V
Rail 0,1,2 Switch Output Current	Current Limited	100	150	200	mA
Rail 1-2 Voltage Error	$V_{rail} \geq$ 2.5V V_{rail} < 2.5V	- -	- -	1 2	%
Rail 1-2 Voltage	V_{rail} = 1.8V; ±2% V_{rail} = 3.3V; ±1% V_{rail} = 5.0V; ±1%	1.764 3.267 4.950	1.800 3.300 5.000	1.836 3.333 5.050	V
Digital Output Drive Current[1]	Output high; short to GND	-	20.0	30.0	mA
Digital Output Sink Current	Output low; short to V_{rail}	-	-10.0	-30.0	mA
Digital Output Short Duration	Output high	-	Infinite	-	hours
Digital Output Overvoltage	V_{supply} on pin	-	Infinite	-	hours
Digital Sample Rate[2]	via USB link, C++ Reflex	- -	1000 8200	- -	Hz Hz
Digital Output Jitter	Using Reflex only Reflex w/ BrainStem load	- -	- -	25 100	µs µs
Digital cmdCAPTURE Frequency Capture		0	-	1000	kHz
USB V_{bus} current limit		500	500	800	mA

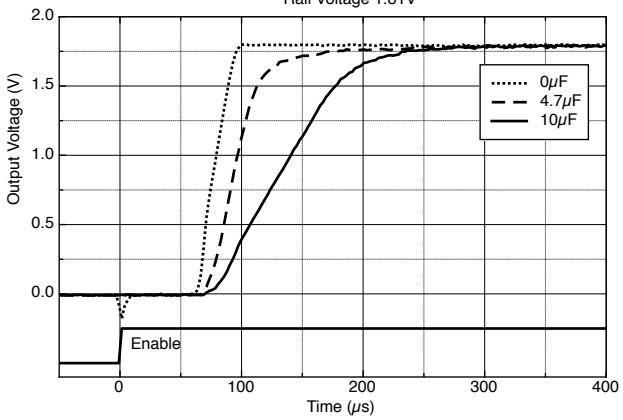
Table 5: Typical Performance Characteristics

[1] It is not recommended to continuously apply more than V_{rail} to any DIO or UART pin.
[2] Host dependent, test was done as a single instruction, subsequent instructions may affect performance.

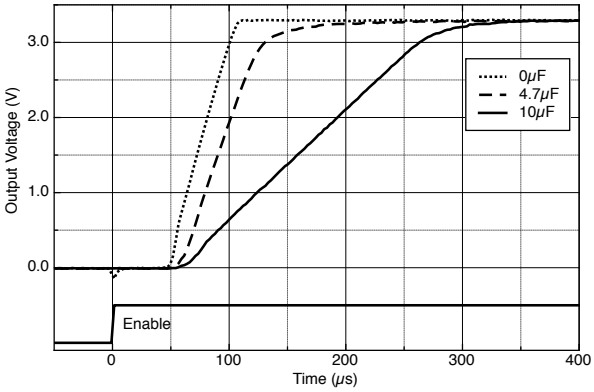
Fixed 5.0V Rail Enable Voltage vs Time



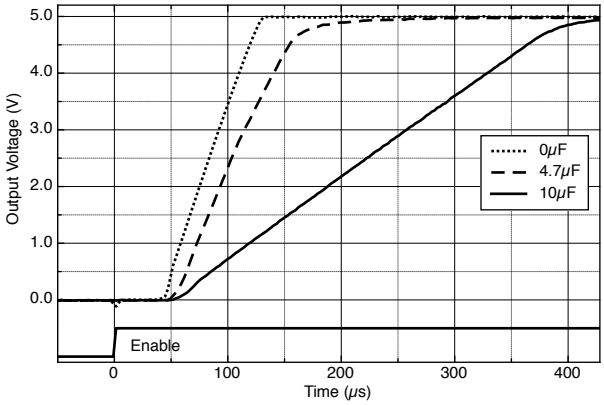
Adjustable Rail Enable Voltage vs Time
Rail Voltage 1.81V



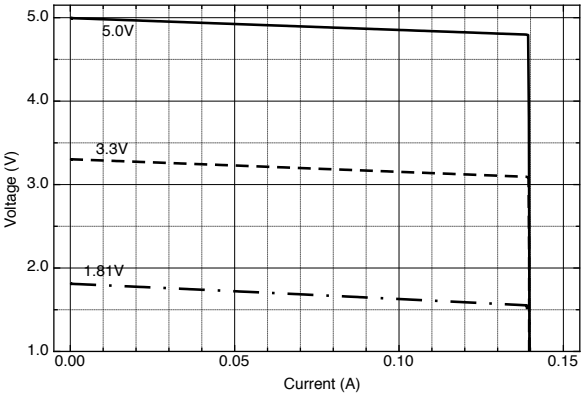
Adjustable Rail Enable Voltage vs Time
Rail Voltage 3.3V



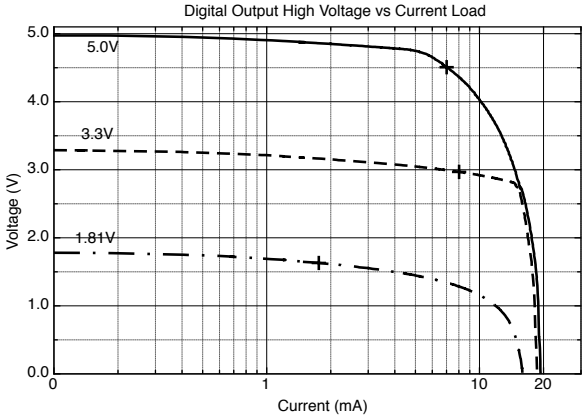
Adjustable Rail Enable Voltage vs Time
Rail Voltage 5.0V

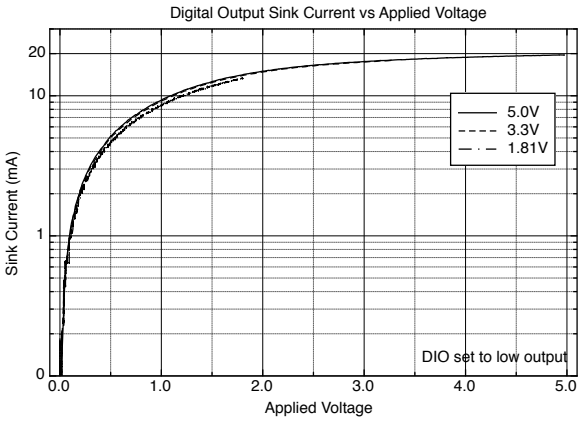


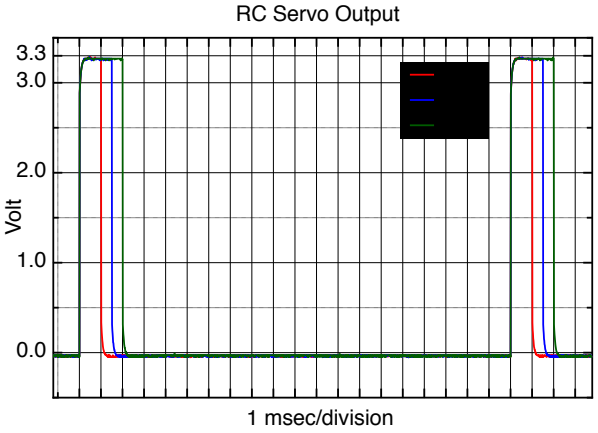
Adjustable Rail Output Voltage vs Output Load Current

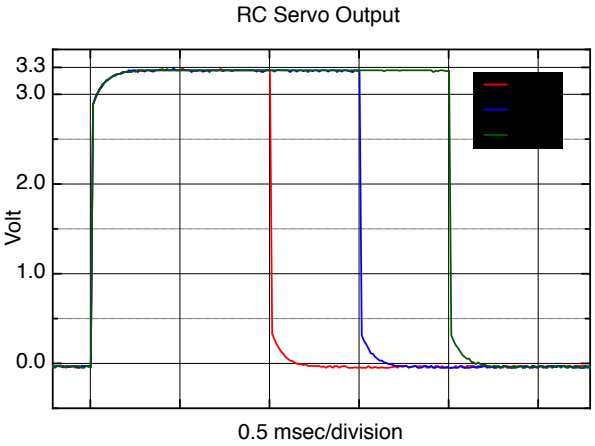


Digital Output High Voltage vs Current Load




Digital Output Sink Current vs Applied Voltage
Sink Current (mA)
20
10
1
0
0.0
1.0
2.0
3.0
4.0
5.0
Applied Voltage
5.0V
3.3V
1.81V
DIO set to low output

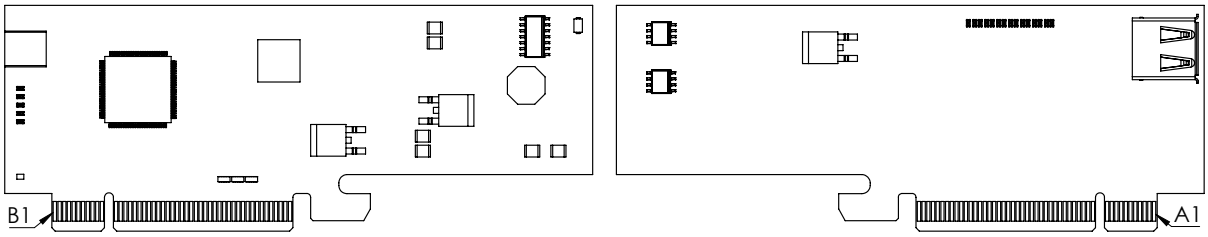

RC Servo Output
3.3
3.0
2.0
1.0
0.0
Volt
1 msec/division


RC Servo Output
3.3
3.0
2.0
1.0
0.0
Volt
0.5 msec/division

Pinout Descriptions

WARNING: MTM modules use a PCIe connector interface that is common in most desktop computers; however, they are NOT intended nor designed to work in these devices. Do NOT insert this product into any PCIe slot that wasn't specifically designed for MTM modules, such as a host PC. Installing this module into a standard PCI slot will result in damage to the module and the PC.

The MTM edge connector pin assignments are shown in the following table. Please refer to Table 4: Recommended Operating Ratings for appropriate signal levels.



Pins Common to all MTM Modules

Side A	Edge Connector Side A Description	Side B	Edge Connector Side B Description
1	GND	1	Input Voltage, V_{supply}
2	GND	2	Input Voltage, V_{supply}
3	GND	3	Input Voltage, V_{supply}
4	GND	4	Input Voltage, V_{supply}
5	Reset	5	Input Voltage, V_{supply}
6	GND	6	Reserved, Do Not Connect
7	GND	7	Reserved, Do Not Connect
8	I^2C0 SCL	8	GND
9	I^2C0 SDA	9	GND
10	GND	10	UART0 Transmit
11	GND	11	UART0 Receive
12	Module Address Offset 0	12	Module Address Offset 2
13	Module Address Offset 1	13	Module Address Offset 3

Table 6: Pins Common to all MTM Modules

Pins Specific to MTM-IO-Serial

Side A	Edge Connector Side A Description	Side B	Edge Connector Side B Description
14	Reserved, Do Not Connect	14	USB Upstream Data +
15	UART2 Tx	15	USB Upstream Data -
16	UART2 Rx	16	UART1 Tx
17	UART3 Tx	17	UART1 Rx
18	UART3 Rx	18	Digital IO 0
19	Reserved, Do Not Connect	19	Digital IO 1
20	Reserved, Do Not Connect	20	Digital IO 2
21	Reserved, Do Not Connect	21	Digital IO 3
22	Reserved, Do Not Connect	22	Digital IO 4
23	Reserved, Do Not Connect	23	Digital IO 5
24	Reserved, Do Not Connect	24	Digital IO 6
25	Reserved, Do Not Connect	25	Digital IO 7
26	Reserved, Do Not Connect	26	Reserved, Do Not Connect
27	Reserved, Do Not Connect	27	Reserved, Do Not Connect
28	Reserved, Do Not Connect	28	Reserved, Do Not Connect
29	Reserved, Do Not Connect	29	Reserved, Do Not Connect
30	Reserved, Do Not Connect	30	Reserved, Do Not Connect
31	Rail 1 Output	31	Rail 2 Output
32	Reserved, Do Not Connect	32	Rail 0 Output
33	GND	33	USB2 V_{bus}
34	USB0 D+	34	USB2 D-
35	USB0 D-	35	USB2 D+
36	USB0 V_{bus}	36	GND
37	GND	37	USB3 V_{bus}
38	USB1 D+	38	USB3 D-
39	USB1 D-	39	USB3 D+
40	USB1 V_{bus}	40	GND
41	Reserved, Do Not Connect	41	Reserved, Do Not Connect
42	Reserved, Do Not Connect	42	USB Edge D-
43	Reserved, Do Not Connect	43	USB Edge D+
44	Reserved, Do Not Connect	44	GND
45	V_{supply}	45	GND
46	V_{supply}	46	GND
47	V_{supply}	47	GND
48	V_{supply}	48	GND
49	V_{supply}	49	GND

Table 7: Pins Specific to MTM-IO-Serial

Module Hardware and Software Default Values

Software Control

The MTM-IO-Serial module firmware is built on Acroname's BrainStem technology and utilizes a subset of BrainStem entity implementations that are specific to the hardware's capabilities. Table 8 details the BrainStem API entities and macros used to interface with the MTM-IO-Serial module. For C and C++ developers, these macros are defined in aMTMIOSerial.h from the BrainStem development package. For Python development, the module MTMIOSerial class properly defines the extent of each entity array.

While Table 8 lists the BrainStem API entities available for this module, not all entity methods are supported by the MTM-IO-Serial. For a complete list of supported entity methods, see Table 16. Note that available method options may vary by entity index, as well as by entity, and calling an unsupported entity option will return an appropriate error (e.g.: `aErrInvalidEntity`, `aErrInvalidOption`, `aErrMode`, or `aErrUnimplemented`) as defined in `aError.h` for C and C++ and the `Result` class in Python.

All API example code snippets that follow are pseudocode loosely based on the C++ method calls - Python and Reflex are similar. Please consult the BrainStem Reference for specific implementation details[7]

Parameter	Index	Macro Name or Implemented Options	Notes
Module Definitions:			
Module Base Address	8	aMTMIOSERIAL_MODULE_BASE_ADDRESS	See aMTMIOSerial.h
Entity Class Definitions:			
digital Entity Quantity	8	aMTMIOSERIAL_NUM_DIGITALS	
i2c Entity Quantity	1		
rail Entity Quantity	3	aMTMIOSERIAL_NUM_RAILS	
5.0V Rail (RAIL0)	0	aMTMIOSERIAL_5VRAIL	
Adjustable Rail (RAIL1)	1	aMTMIOSERIAL_ADJRAIL1	
Adjustable Rail (RAIL2)	2	aMTMIOSERIAL_ADJRAIL2	
UART Entity Quantity	4	aMTMIOSERIAL_NUM_UART	
USB Entity Quantity	1	aMTMIOSERIAL_NUM_USB	
Store Entity Quantity	2	aMTMIOSERIAL_NUM_STORES	
system Entity Quantity	1		
timer Entity Quantity	8	aMTMIOSERIAL_NUM_TIMERS	
app Entity Quantity	4	aMTMIOSERIAL_NUM_APPS	
pointer Entity Quantity	4	aMTMIOSERIAL_NUM_POINTERS	
servo Entity Quantity	8	aMTMIOSERIAL_NUM_SERVOS	
signal Entity Quantity	5	aMTMIOSERIAL_NUM_SIGNALS	

Table 8: MTM-IO-Serial Hardware and Software Default Values[3]

[3] Refer to `aMTMIOSerial.h` within the BrainStem Development Kit download for actual file.

Capabilities and Interfaces

BrainStem Link and Module Networking

A BrainStem link can be established that will give the user access to the resources available on the MTM-IO-Serial. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS.

A BrainStem link to the MTM-IO-Serial can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-IO-Serial is attached to a host machine, a user can connect to it via software API:

```
stem.link.discoverAndConnect(linkType,
serialNumber, modelNumber)
```

The MTM-IO-Serial can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I²C. Each MTM-IO-Serial is uniquely addressable via hardware or software to avoid communication conflicts on the I²C bus. A software offset can be applied as follows:

```
stem.system.setModuleSoftwareOffset(address)
```

Module Address Hardware Offset Configuration

A hardware offset allows a user to modify the module's address on the BrainStem network. Using hardware offset pins is useful when more than one of the same module type is installed on a single BrainStem network. Applying a different hardware offset to each module of the same type in one network allows for all the modules to seamlessly and automatically configure 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 module type in a network, the module address hardware offset can be used to determine the module's physical location and thus its interconnection and intended function. For detailed information on BrainStem networking see the BrainStem Reference[7].

Each hardware offset pin can be left floating or pulled to ground with a 1kΩ resistor or smaller (pin may be directly 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 pin states are treated as a bit state within a 4bit number. This number is multiplied by 2 and added the to the module's base address. The hardware offset calculation is detailed in the following table:

HW Offset Pin				Address Offset	Module Base Address	Final Module Address
3	2	1	0			
NC	NC	NC	NC	0	4	4
NC	NC	NC	1	2	4	6
NC	NC	1	NC	4	4	8
NC	1	NC	NC	8	4	12
1	NC	NC	NC	16	4	20
1	NC	NC	1	(1+8) * 2	4	22

Table 9: Module Address Hardware Offset Examples

Upstream USB Connectivity Options

The MTM-IO-Serial supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the mini-B connector if 5V is present on V_{bus} at the mini-B connector.

System Entities

Every BrainStem module includes a single System Entity. The System Entity allows access to configuration settings such as the module address and I²C rate, measurements such as input voltage, control over the user LED, and many more.

Saving Entity Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the startup and operational behavior for the MTM-IO-Serial *away* from the factory default settings. Saving system settings creates a new default and often requires a reboot of the MTM-IO-Serial for changes to take effect; see Table 10: Entity Values Saved by system.save() for relevant settings. Use the following command to save changes to system settings before reboot:

```
stem.system.save()
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	I2C Pullup State
Heartbeat Rate	Boot Slot

Table 10: Entity Values Saved by system.save()

Store Entities

Every BrainStem module includes several Store entities and onboard memory slots to load Reflex files (for details on Reflex, see BrainStem Reference[7]). One Reflex file can be stored per

slot. Store[0] refers to the internal flash memory, with 12 available slots, and store[1] refers to RAM, with 1 available slot.

Digital Entities

The MTM-IO-Serial has eight (8) digital input/outputs (DIO) controlled by the digital entity.

The digital inputs and outputs on the MTM-IO-Serial module have software-adjustable logic levels and are limited to sourcing or sinking 20 mA. The DIO are split into two groups (`DIO0-3` and `DIO4-7`) and the logic level for all four pins within each group derive from the same voltage rail (`RAIL1` or `RAIL2`). While each pin can independently sink 20mA, the total DIO source current from a single rail is 20mA. A plot of expected output voltage as a function of current sourcing and a plot of current sinking ability as a function of applied voltage is shown in the Typical Performance Characteristics section. Because the current sourcing ability applies to the internal rail of each group, the voltage output level of the entire group can be affected by one or more pins sourcing or sinking high levels of current. Note: it is not recommended to apply more than the configured rail voltage to any DIO pin.

All DIO are input and output capable:

```
stem.digital[0].setConfiguration(mode)
stem.digital[0].getConfiguration(mode)
```

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

Value	Configuration
0	Input
1	Output
2	RC Servo Input
3	RC Servo Output
6	Signal Output
7	Signal Input

Table 11: Digital IO Configuration Values

Example: If a digital pin is configured as an output, set the digital logic state:

```
stem.digital[0].setState(state)
```

Example: If a digital pin is configured as an input, read the digital logic state:

```
stem.digital[0].getState(state)
```

The logic level for a given pin follows its rail and is controllable as explained in the Rail Entities section below. See Table 12: Digital IO to determine the corresponding rail for each pin.

Every DIO pin also has an internal pull-down resistor when configured as an input.

Configuring a digital pin as an RCServo input or output requires use of the RCServo Entity. The RCServo input and output modes are only available on a subset of the digital pins. Refer to Table 12: Digital IO for a complete list. **Note**: Signal entity is covered in a subsequent section

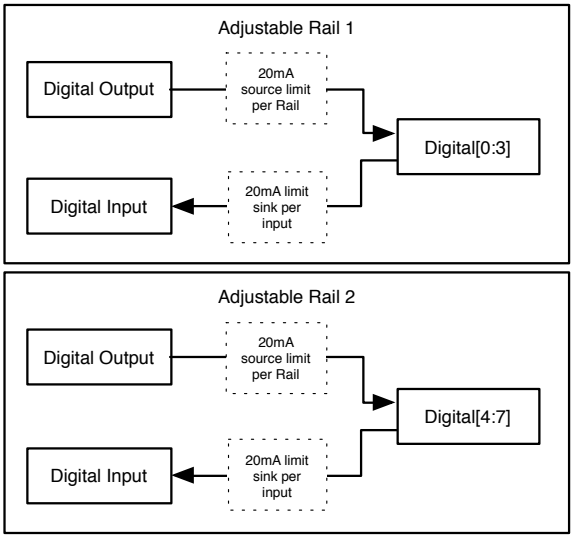


Figure 2: DIO and voltage rail grouping

Pin	Input	Output	Rail	Hi-Z	Servo	Signal
DIO 0	Yes	Yes	1	No	Input	Input
DIO 1	Yes	Yes	1	No	Input	Input / Output
DIO 2	Yes	Yes	1	No	Input	Input / Output
DIO 3	Yes	Yes	1	No	Input	Input / Output
DIO 4	Yes	Yes	2	No	Output	Input / Output
DIO 5	Yes	Yes	2	No	Output	-
DIO 6	Yes	Yes	2	No	Output	-
DIO 7	Yes	Yes	2	No	Output	-

Table 12: Digital IO Pin Configurations

The DIOs are controlled with an array of the BrainStem digital class. The MTM-IO-Serial module implements a subset of the digital class for each DIO index. The implemented entity options for digital class entity index are summarized below.

I²C Entities

The MTM-IO-Serial includes access to one I²C bus operating at a set 1Mbit/s rate.

Note: The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.

Example: Sending 2 bytes (0xABCD) through the I²C bus to a device with address 0x42:

```
stem.i2c.write(0x42, 2, 0xABCD)
```

Example: Reading 2 bytes of data from a device with address 0x42:

```
stem.i2c.read(0x42, 2, buffer)
```

Where *buffer* would be a char array in C++.

The maximum data size for individual read and write operations on an I²C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Each I²C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-IO-Serial in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I²C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I²C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added:

```
stem.i2c.setPullUp(bEnable)
```

RC Servo Entities

The MTM-IO-Serial board is equipped with 4 RC servo inputs and 4 RC servo outputs. The RC Servo entity is an overload of the Digital Entity and thus requires proper configuration before this entity can be enabled.

Example: Enabling RC servo input mode for digital pin 0:

```
stem.digital[0].setConfiguration(digitalConfi
gurationRCServoInput)
```

With the RC servo entity, digital output pins generate pulsed signal based on the RC Servo standard consisting of a period lasting 20ms and high pulse time between 1-2ms. The high time corresponds to a specific position determined by the specific servo being used. RC servo inputs, measure this high time and return the corresponding position for a servo.

Example: When operating as an RC servo input, enabling functionality and reading position:

```
stem.RCServo[0].setEnable(bool)
stem.RCServo[0].getPosition(position)
```

Example: When operating as an RC servo output, enabling functionality and setting position:

```
stem.RCServo[4].setEnable(bool)
stem.RCServo[4].setPosition(position)
```

Signal Entities

The MTM-IO-Serial board has 5 Signal Input and 4 Signal outputs. The Signal entity allows you to generate square waves by supplying a period and a time high value. The Signal inputs can also be used as counters.

The Signal entity is an overload of the Digital entity and must first be properly configured before it can be enabled. Using Table 12: Digital IO , configure the correct digital pins as a Signal input:

```
stem.digital[4].setConfiguration(digitalConfi
gurationSignalInput)
```

Now, configure period (t3 time) and the time high value (t2 time):

```
stem.signal[0].setT3Time(period)
stem.signal[0].setT2Time(timeHigh)
stem.signal[0].setEnable(enable)
```

Note: See the BrainStem Reference[7] guide for timing diagram.

Signal and Digital Entities' indices do not necessarily align. Counting for the first Signal Entity starts at the first digital pin that is equipped with a Signal overload (Digital 0 = Signal 0, see Table 12: Digital IO).

Rail Entities

Rails allow other devices and peripherals to consume power from the MTM-IO-Serial module in a controlled fashion. Three (3) different rails are available for use in a variety of application: a single fixed 5.0V rail (`RAIL0`) and 2 adjustable voltage rails (`RAIL1`, `RAIL2`). These rails are accessed through an array of BrainStem `rail` class entities. The MTM-IO-Serial module implements a subset of the BrainStem rail class for each of these rails. The implemented rail entity options for each entity index are summarized below.

Enabling Rails

All three rails can be switched on or off through using the API:

```
stem.rail[0].setEnable(state)
```

RAIL0 Operational Mode

`RAIL0` can be configured to use two different regulation stages: linear (LDO) or switch-mode power supply (SMPS):

```
stem.rail[0].setOperationalMode(mode)
stem.rail[0].getOperationalMode(mode)
```

The *mode* parameter is an integer that corresponds to the following:

- 0 (railOperationalModeAuto) default
- 1 (railOperationalModeLinear)
- 2 (railOperationalModeSwitcher)

Auto configuration chooses the switch-mode power supply if an input voltage greater than 7.25V is applied, and the linear regulator otherwise. The API can be used to read the actual operational state:

```
stem.rail[0].getOperationalState(state)
```

The value *state* is a 32-bit value. Bits 8-15 correspond to the active hardware configuration:

- 0 (railOperationalStateLinear)
- 1 (railOperationalStateSwitcher)

Refer to `aProtocoldefs.h` in the BrainStem Reference for more details[7].

For applications such as RF system testing, one might want to operate only in linear regulation mode to eliminate any potential EMI sources. While operating in linear mode, one must be aware of power dissipation through the linear regulation stage. A higher input voltage will result in higher power dissipation. When linear mode is desired and high current operation is desired it is recommended to run the input voltage close to the MTM-IO-Serial module's minimum input voltage. Switch mode power supply operation will allow a broader range of input voltages while maintaining high current demand limits.

Figure 3 presents a simplified block diagram for the 5.0V regulation paths.

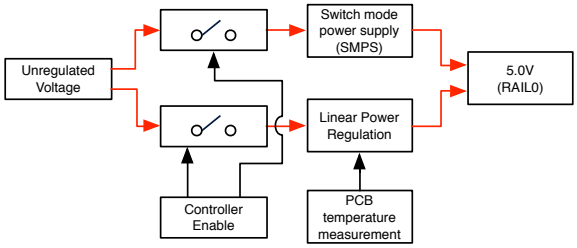


Figure 3: Paths for 5.0V Regulation Stage

RAIL0 Temperature

The printed circuit board (PCB) temperature can be monitored at the 5.0V rail (`RAIL0`) linear regulation stage. Reading this value is possible through the API:

```
stem.rail[0].getTemperature(temperature)
```

Temperature monitoring is also used internally to prevent the power regulation stage from overheating and self-preserving the power stage. If an overtemperature condition occurs, then the MTM-IO-Serial module will disable the linear regulator until safe operating temperatures are reached.

RAIL1 and RAIL2 Voltage Setting

`RAIL1` and `RAIL2` always use linear regulators to generate their adjustable voltages. They can be set or read using the API:

```
stem.rail[1].setVoltageSetpoint(microvolts)
stem.rail[1].getVoltageSetpoint(microvolts)
```

RAIL Protection

Each rail is current limited in hardware to 100mA and will operate in constant-current mode upon reaching 100mA. Extended operation in constant-current mode is discouraged and may result in thermal shutdown of the rail.

Each rail is automatically disconnected when an overvoltage condition is detected and automatically reconnected if the overvoltage condition ceases. Overvoltage detection is implemented in hardware and based on the rail's voltage setpoint.

UART Entities

UART entities provide a mechanism to enable and disable UART data lines.

All the UARTs that are passed down from the MTM-IO-Serial module can be turned on/off through software control. If a voltage is applied that is higher than the current `rail` voltage setpoint, each UART transmit line is current limited to 20mA sinking. Therefore, only a small amount of current will flow into

the device, preventing any damage to the MTM-IO-Serial module's hardware.

Each UART is paired with a specific voltage rail. `UART0` and `UART1` use `RAIL1`'s voltage reference. `UART2` and `UART3` use `RAIL2`'s voltage reference.

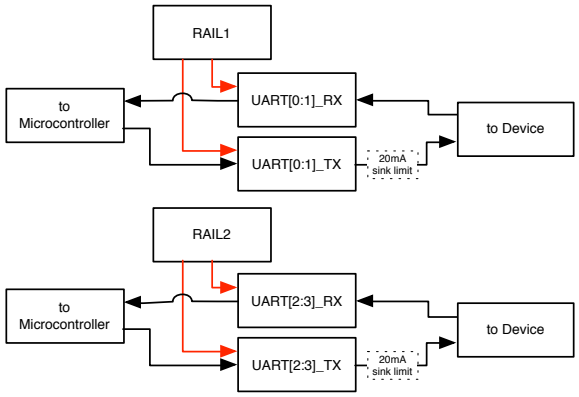


Figure 4: UART and Rail Voltage Pairings

When a UART is disabled by means of the command `UART`, all exposed UART data lines will be discharged by being pulled to Ground through a 10kΩ resistor.

USB Entities

The `usb` entity manages the software-controllable downstream USB 2.0 channels of the MTM-IO-Serial (there are also two non-software-controllable USB channels on the module, one through the edge connector and the other through the onboard type-A connector, which are always on), as well as the upstream USB connection mode. All downstream USB ports are configured as SDP (Standard Data Port).

USB Downstream

Each of the four software-controllable USB channels (ports 0-3) can be individually manipulated using the `usb` entity. The API individually controls port power, data, or both together:

```
stem.usb.setPowerEnable(port)          (just VBUS)
stem.usb.setPowerDisable(port)

stem.usb.setDataEnable(port)           (just D+/D-)
stem.usb.setDataDisable(port)

stem.usb.setPortEnable(port)           (both)
stem.usb.setPortDisable(port)
```

The *port* parameter is an integer that correlates to the software-controllable downstream channel (0-3).

USB Upstream

The MTM-IO-Serial has two (2) upstream USB connection options: through the edge connector or via the mini-B connector on the board itself. The upstream mode can be set or read using the `usb` entity:

```
stem.usb.setUpstreamMode(mode)
stem.usb.getUpstreamMode(mode)
```

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

- 0 (edge connector)
- 1 (mini-B connector)
- 2 (auto configuration) *default*

Auto configuration chooses the upstream connection based on the presence or absence of VBUS power at the mini-B connector; if VBUS is present, the mini-B connector is used, otherwise the edge connector is used. The actual upstream connection in use can be read using the `usb` entity:

```
stem.usb.getUpstreamState(mode)
```

USB Hub Operational Mode

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

```
stem.usb.getHubMode(state)
stem.usb.setHubMode(state)
```

The value *state* must be a 32-bit word, defined as the following:

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

Table 13: Hub Operational Mode Result Bitwise Description

USB Port State

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

```
stem.usb.getPortState(channel,state)
```

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

Bit	Port State Result Bitwise Description
0	USB VBUS Enabled
1	USB2 Data Enabled
2:18	Reserved
19	USB Error Flag
20	USB2 Boost Enabled
21:31	Reserved

Table 14: Port State: Result Bitwise Description

USB Hub Error Status Mapping

It is possible to retrieve current error states for all downstream ports in a single 32-bit word. As only 4 downstream USB ports are available, the *bank* parameter should be set to 0 for the USBHub2x4.

```
stem.usb.getHubErrorStatus(bank=0,status)
```

Errors can be cleared on each individual channel (0, 1, 2 or 3) by calling the following method:

```
stem.usb.clearPortErrorStatus(channel)
```

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

Bit	Hub Error Status Result Bitwise Description
0	USB CH0 overcurrent limit exceeded[4]
1	USB CH0 VBUS back drive[5]
2	USB CH0 hub power system failure
3	USB CH0 VBUS discharge[6]
4:7	Reserved
8	USB CH1 overcurrent limit exceeded[4]
9	USB CH1 VBUS back drive[5]
10	USB CH1 hub power system failure
11	USB CH1 VBUS discharge[6]
12:15	Reserved
16	USB CH2 overcurrent limit exceeded[4]
17	USB CH2 VBUS back drive[5]
18	USB CH2 hub power system failure
19	USB CH2 VBUS discharge[6]
20:23	Reserved
24	USB CH3 overcurrent limit exceeded
25	USB CH3 VBUS back drive[5]
26	USB CH3 hub power system failure
27	USB CH3 VBUS discharge[6]
28:31	Reserved

Table 15: Hub Error Status Result Bitwise Description

Reflex RTOS

Reflex is Acroname's real-time operating system (RTOS) language which runs in parallel to the module's firmware. Reflex allows users to build custom functionality directly into the device. Reflex code can be created to run autonomously on the module or a host can interact with it through BrainStem's Timer, Pointer App and other entities.

Timer Entities

The Timer entity provides simple scheduling for events in the reflex system. The MTM-IO-Serial includes 9 timers per reflex. Each timer represents a reflex definition to be executed upon expiration of a running timer. Timers can be controlled from a host, but the reflex code is executed on the device.

Example: Setting up and starting Timer 0 for single use:

```
stem.timer[0].setMode(timeModeSingle)
stem.timer[0].setExpiration(DELAY)
```

Reflex Definition: Timer 0 expiration callback:

```
reflex timer[0].expiration() { //Do Stuff }
```

Pointer Entities

Reflex and the Brainstem module share a piece of memory called the scratchpad which can be accessed via the Pointer Entity. The MTM-IO-Serial has 4 pointers per reflex which allow access to the pad in a similar manner as a file pointer.

Example: Configure and access the scratchpad in static mode:

```
stem.pointer[0].setMode(pointerModeStatic)
stem.pointer[0].getByte(byte)
```

Reflex Pad: Single unsigned char definition:

```
pad[0:0] unsigned char byteValue
```

App Entities

Apps are reflex definitions that can be directly trigger by the host. These definitions are also capable of passing a parameter into or out of the app reflex definition. The MTM-IO-Serial is equipped with 4 App Entities per reflex.

Example: Triggering App 0:

```
stem.app[0].execute(parameter)
```

Reflex Definition: App 0 callback:

[4] Current limit value is defined by API settings section on USB Downstream Channels.
[5] VBUS exceeds 5.150V for longer than 5ms.
[6] VBUS discharge circuitry is activated. At the end of the 200ms the hub will confirm that VBUS was discharged if the VBUS voltage is not below 0.750V.

```
reflex app[0](int appParam) { //Do Stuff }
```

MTM-IO-Serial Supported Entity Methods Summary

Detailed entity class descriptions can be found in the BrainStem Reference[7]. A summary of MTM-IO-Serial class options are shown below. Note that when using Entity classes with a single index (i.e., 0), the index parameter can be dropped. For example:

`stem.system[0].setLED(1)` → `stem.system.setLED(1)`

Entity Class	Entity Option	Variable(s) Notes
digital[0-7]	setConfiguration	
	getConfiguration	
	setState	
	getState	
rcservo[0-7]	setEnable	
	getEnable	
	setPosition	Index 4-7 only
	getPosition	
	setReverse	Index 4-7 only
	getReverse	
i2c[0]	write	
	read	
usb[0]	setPortEnable	
	setPortDisable	
	setDataEnable	
	setDataDisable	
	setHiSpeedDataEnable	
	setHiSpeedDataDisable	
	setPowerEnable	
	setPowerDisable	
	getPortError	
	clearPortErrorStatus	
	getSystemTemperature	In microcelsius
	setUpstreamMode	
	getUpstreamMode	
	getUpstreamState	
	getDownstreamDataSpeed	
	getHubMode	
	setHubMode	
	getPortState	
UART[0-3]	setEnable	
	getEnable	
rail[0]	setEnable	
	getEnable	
	getTemperature	In microcelsius
	setOperationalMode	
	getOperationalMode	
	getOperationalState	
	getVoltage	In microvolts
rail[1-2]	setEnable	
	getEnable	

Entity Class	Entity Option	Variable(s) Notes
	setVoltageSetpoint	In microvolts
	getVoltageSetpoint	In microvolts
	getVoltage	In microvolts
signal[0-5]	setEnable	
	getEnable	
	setInvert	
	getInvert	
	setT3Time	
	getT3Time	
	setT2Time	
	getT2Time	
store[0-2]	getSlotState	
	loadSlot	
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setSleep	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	
timer[0-7]	getExpiration	
	setExpiration	
	getMode	
	setMode	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	
	getTransferStore	

Entity Class	Entity Option	Variable(s) Notes
	setTransferStore	
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	
	setInt	
App[0-3]	execute	

Table 16: Supported MTM-IO-Serial BrainStem Entity API Methods[7]

[7] See BrainStem software API reference at https://acroname.com/reference/ for further details about all BrainStem API methods and information.

LED Indicators

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

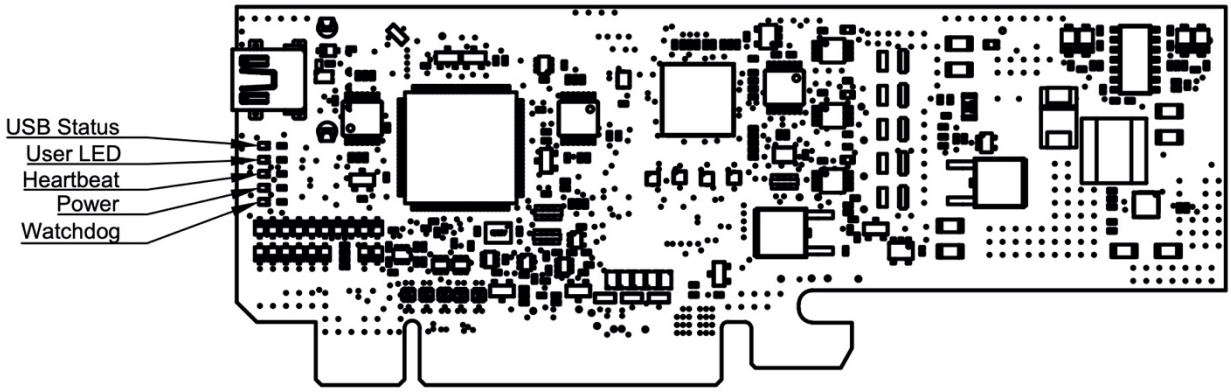


Figure 5: MTM-IO-Serial LED Indicators

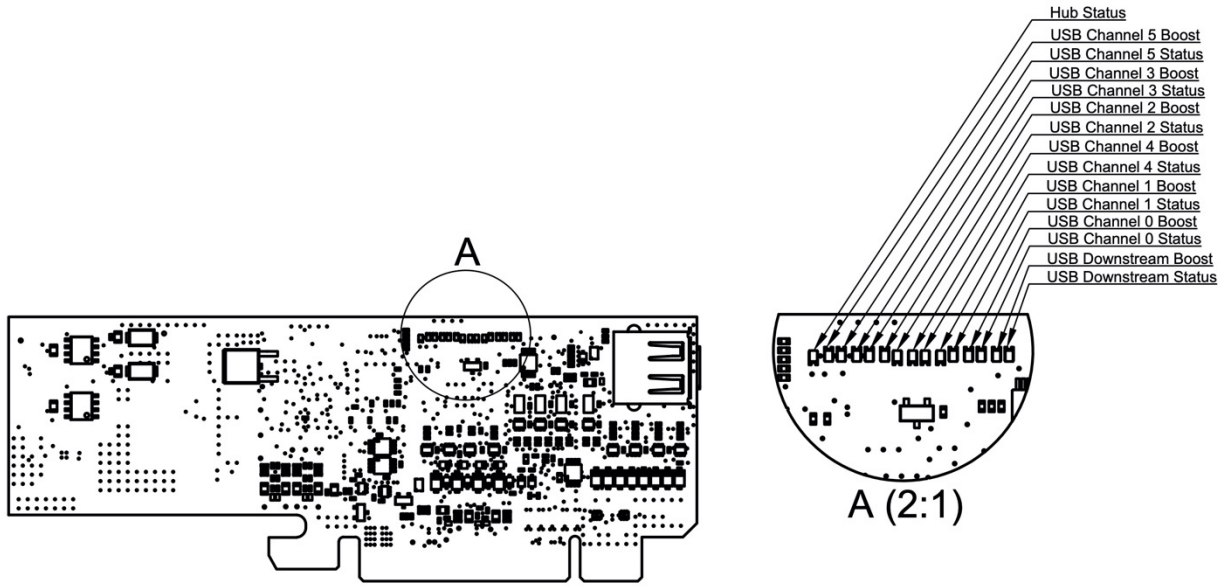


Figure 6: USB Status Indicators

Edge Connector Interface

All MTM products are designed with an edge connector interface that requires a compatible board-edge connector on the carrier PCB. Acroname recommends the through-hole PCI-Express (PCIe) Vertical Connector. The connectors can be combined with an optional retention clip, as shown below. Representative part numbers are show in Table 17, and equivalent connectors are offered from a multitude of vendors.

Manufacturer	Manufacturer Part Number	Description
Amphenol FCI Samtec	10018784-10202TLF PCIE-098-02-F-D-TH	PCI-Express 98-position vertical connector
Amphenol FCI	10042618-003LF	PCI-Express Retention Clip (optional)

Table 17: PCI-Express Edge Connectors for MTM Products

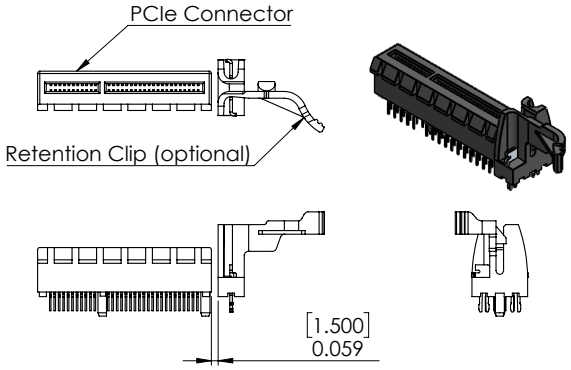


Figure 7: PCIe Vertical Connector with optional Retention Clip

MTM Edge Connector Specifications	Description
Contact Finish	Gold
Card Thickness	0.0625” [1.59mm]
Number of Rows	2
Number of Positions	Variable (see Table 17: PCI-Express Edge Connectors for MTM Products)
Pitch	0.039” (1.00mm)

Table 18: MTM Edge Connector Specifications

Mechanical

Dimensions are shown in inches [mm]. 3D CAD models are available through the MTM-IO-Serial product page's Downloads section. A 3D CAD viewer with many different CAD model formats available for download is available at https://a360.co/2Nc7y1B

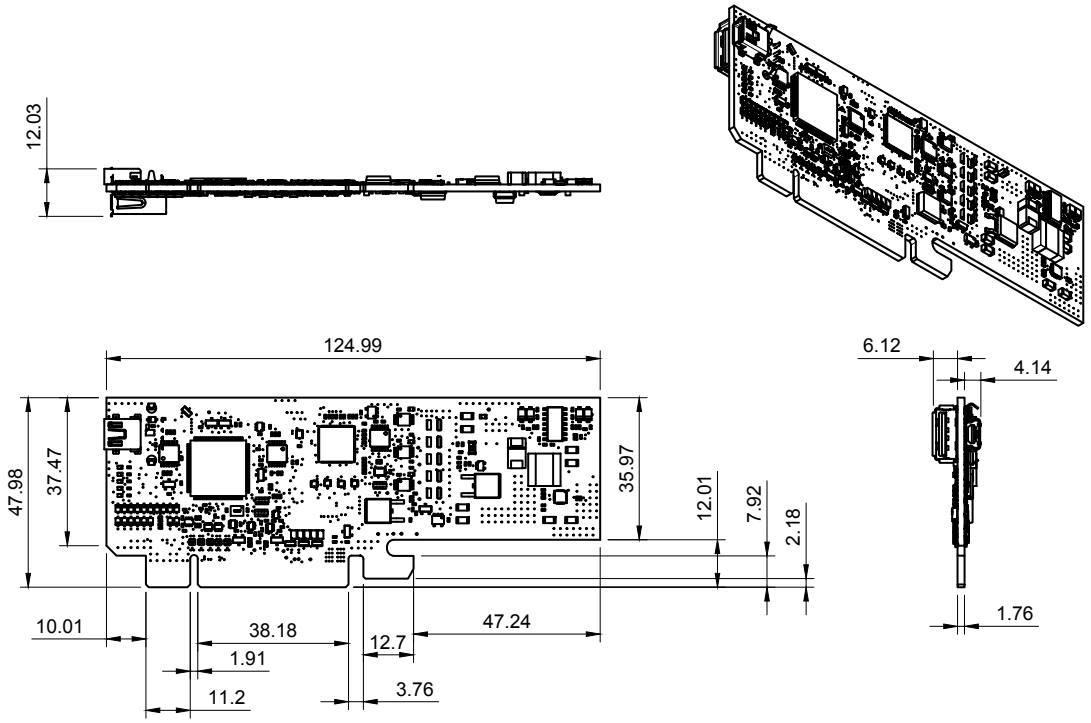


Figure 8: MTM-IO-Serial Mechanical

Product Support

Questions about the product operation or specifications are welcome through Acroname's contact portals. Software downloads, reference API and application examples are available online at:

https://acroname.com/support

Direct communication and additional technical support are available at:

https://acroname.com/contact-us

2741 Mapleton Avenue

Boulder, CO, USA 80304-3837

720-564-0373 (phone)

Document Revision History

All major documentation changes will be marked with a dated revision code

Rev	Date	Engineer	Description
1.0	July 2015	MJK	Initial Revision
1.1	January 2016	JTD	Updated to BrainStem 2.2 firmware
1.2	September 2016	RMN	Formatting, Error checking, updates
1.3	October 2016	LCD	Updated Overview, Features, Description sections, added DO jitter
1.4	December 2016	JG	Clarified I2C pull-ups; update supported API calls
1.5	March 2017	JTD	Updated block diagram, USB maximum ratings
1.6	May 2017	RMN	Replaced references of MUX with UART.
1.7	April 2018	RMN	Swapped hubState for portState
1.8	July 2020	ACRO	Update formatting, fusion 360 model, entity updates, diagram updates
1.9	September 2020	TDH	Rail API updates for Brainstem 2.8
1.10	January 2021	JLG	Fix type in port state table title
1.11	February 2021	MJK	Contact information for technical support
1.12	July 2022	JLG	Add cmdSIGNAL specification
1.13	September 2024	TDH	Correct typo in pinout table
1.14	March 2025	RMN	Corrected mention of 2x I2C buses (only 1x); Typo in rail voltage conditions.

Table 19: Document Revision History