SIB Integration API Documemntation

`public String getDeviceId()`

Get the device id for the device associated with the current context

- **Returns:** String - unique id of the device in the current context

`public void powerOn12V()`

Used to power on the device with 12 volts. Will not fail if the device is already powered on

`public void powerOn15V()`

Used to power on the device with with 15 volts. Will not fail if the device is already powered on

`public void powerOn(String voltage)`

Used to power on the device with the given voltage. Will not fail if the device is already powered on

- **Parameters:** `voltage` — - String that can be either 'ON_12V' or 'ON_15V'

`public void powerOff()`

Used to power off the device. Will not fail if the device is already powered off

`public void powerOff(long milliseconds)`

Used to power off the device. The power off will be attempted after the given milliseconds time interval has elapsed. Wait is asynchronous and will not block. Will not fail if the device is already powered off.

- **Parameters:** `milliseconds` — - Wait time
- **Exceptions:** `IllegalArgumentException` — - If the given wait time is negative

`public Boolean isPowerOn()`

Check if the device is currently powered on

- **Returns:** Boolean - true if on, else false

`public byte[] sendCommand(byte[] command)`

Send a synchronous command represented as a byte array to the device with a default timeout of 5s. Returns the response as a byte array or null if there was no response

- **Parameters:** `command` — - byte array representing the command
- **Returns:** byte array - byte array representing the response from the device, or null if no response within 5s

`public String sendCommand(String command)`

Send a synchronous command represented as a String to the device with a default timeout of 5s. Returns the response as a String or null if there was no response

- **Parameters:** `command` — - String representing the command
- **Returns:** String - String representing the response from the device, or null if no response within 5s

`public String sendCommandWithTimeout(String command, int timeoutMs)`

Send a synchronous command represented as a String to the device with a custom timeout. Returns the response as a String or null if there was no response

- **Parameters:**
  - `command` — - String representing the command
  - `timeoutMs` — - timeout in milliseconds
- **Returns:** String - String representing the response from the device, or null if no response within given timeout

`public String[] sendCommand(String command, int numResponses)`

Send a synchronous command represented as a String to the device with a default timeout of 5s and wait for the number of responses. Returns the response as a String array or empty array if there was no response

- **Parameters:**
  - `command` — - String representing the command
  - `numResponses` — - int representing how many responses to wait for before returning the responses
- **Returns:** String array - String array representing the responses from the device

`public String[] sendCommand(String command, int numResponses, int timeoutMs)`

Send a synchronous command represented as a String to the device with a custom timeout and wait for the number of responses. Returns the response as a String array or empty array if there was no response

- **Parameters:**
  - `command` — - String representing the command
  - `numResponses` — - int representing how many responses to wait for before returning the

responses
    - `timeoutMs` — - timeout in milliseconds
- **Returns:** String array - String array representing the responses from the device

`public String bytesToHex(byte[] bytes)`

Converts the given byte array argument to its hex representation as a String

- **Parameters:** `bytes` — - byte array to convert to hex
- **Returns:** String - representing the hex characters

`public byte[] hexToBytes(String hex)`

Converts the given hex string to a byte array

- **Parameters:** `hex` — - String representing the hex value to convert
- **Returns:** byte array - representing the hex value as a byte array

`public int hexToDecimal(String hex)`

Converts the given hex string to a decimal value

- **Parameters:** `hex` — - String representing the hex value to convert
- **Returns:** int - representing the hex value as an decimal value

`public float hexToFloatingPoint(String hex)`

Converts the given hex string to a 32 bit signed floating point number

- **Parameters:** `hex` — - String representing the hex value to convert
- **Returns:** float - representing the hex value as a floating point number
- **Exceptions:** `IllegalArgumentException` — - If the given hex is null or empty

`public byte[] floatToBytes(float value)`

Converts the given float value (32-bit precision IEEE 754 floating point) to a byte array

- **Parameters:** `value` — - float value to convert
- **Returns:** byte array - representing the float value as a byte array

`public byte[] floatToIntBytes(float value)`

Converts the given float value (32-bit precision IEEE 754 floating point) to a 4 byte (32-bit) int array that represents the rounded integer value

- **Parameters:** `value` — - float value to convert

- **Returns:** byte array - representing the float value as a 4 byte (32-bit) int

`public float bytesToFloat(byte[] bytes)`

Converts the big-endian byte array of length 4 to its float representation

- **Parameters:** `bytes` — - big-endian byte array of length 4 to convert
- **Returns:** float - representing the float value

`public void sleep(long ms) throws InterruptedException`

Sleep for a period of time. This will pause execution.

- **Parameters:** `ms` — - number of milliseconds to sleep, max allowed is 300000 ms (5mins)
- **Exceptions:** `InterruptedException` —

`public String bytesToDecimal(byte[] bytes)`

Translates a byte array containing the two's-complement binary representation of a decimal value.

The input array is assumed to be in big-endian byte-order. The most significant byte is in the zeroth element.

- **Parameters:** `bytes` — - two-s complement binary representation of a decimal value
- **Returns:** String - representing the decimal value

`public byte[] concatBytes(byte[] bytes1, byte[] bytes2)`

Returns the values from each provided array combined into a single array.

For example, concatBytes({a, b}, {c}} returns the byte array {a, b, c}}.

- **Parameters:**
  - `bytes1` — - first array of bytes to concat
  - `bytes2` — - second array of bytes to concat
- **Returns:** byte array - representing the first given array concatenated with the second given array

`public String[] getSensorIds()`

Returns an array of sensor ids for the current device

- **Returns:** String array - an array of sensor ids attached to the current device

`public String getSensorByIntegrationId(String integrationId)`

Return a sensor id by sensor integration id for the current device

- **Parameters:** `integrationId` — - id of the sensor integration
- **Returns:** String - a sensor id attached to the current device, or null if nothing found

`public String[] getSensorsByIntegrationId(String integrationId)`

Return sensorIds by sensor integration id for the current device

- **Parameters:** `integrationId` — - id of the sensor integration
- **Returns:** String array - an array of sensorIds attached to the current device, or null if nothing found

`public String getSensorByMuxIndex(int muxIndex)`

Return a sensor id by mux index for the current device

- **Parameters:** `muxIndex` — - mux index position (not 0-based)
- **Returns:** String - a sensor id attached to the current device, or null if nothing found

`public String getSensorByProperty(String key, Object value)`

Return a sensor id by sensor property key and value for the current device

- **Parameters:**
  - `key` — - String representation of a valid PropertyType
  - `value` — - value for the associated property
- **Returns:** String - a sensor id attached to the current device, or null if nothing found

`public String[] getSensorsByProperty(String key, Object value)`

Return sensorIds by sensor property key and value for the current device

- **Parameters:**
  - `key` — - String representation of a valid PropertyType
  - `value` — - value for the associated property
- **Returns:** String array - an array of sensorIds attached to the current device, or null if nothing found

`public boolean hasDeviceProperty(String key)`

Returns true if device in the current context has a value for the given property key

- **Parameters:** `key` — - String representing the property
- **Returns:** true if property exists, false otherwise

`public boolean isDeviceProperty(String key, Object value)`

Returns true if device has the given property and it equals the given value

- **Parameters:**
  - `key` — - String representing the property
  - `value` — - value to check
- **Returns:** true if property exists and equals value, else false

`public Object getDeviceProperty(String key)`

Returns a property value given the property key for the device in the current context

- **Parameters:** `key` — - String representing the property
- **Returns:** value of the property

`public Object getDeviceProperty(String key, Serializable defaultValue)`

Returns a property value given the property key for the device in the current context or the defaultValue if the property is not present

- **Parameters:**
  - `key` — - String representing the property
  - `defaultValue` — - default value
- **Returns:** value of the property or the defaultValue given

`public void setDeviceProperty(String key, Object value)`

Sets a property on the device in the current context

- **Parameters:**
  - `key` — - String representation of a valid PropertyType
  - `value` — - value to set for the associated property

`public void removeDeviceProperty(String key)`

Removes a property on the device

- **Parameters:** `key` — - String representation of a valid PropertyType

`public boolean hasSensorProperty(String sensorId, String key)`

Returns true if sensor given has a value for the given property key

- **Parameters:** `key` — - String representing the property
- **Returns:** true if property exists, false otherwise

`public boolean isSensorProperty(String sensorId, String key, Object value)`

Returns true if sensor has the given property and it equals the given value

- **Parameters:**
  - `sensorId` — - id of the sensor
  - `key` — - String representing the property
  - `value` — - value to check
- **Returns:** true if property exists and equals value, else false

`public Object getSensorProperty(String sensorId, String key)`

Returns a property value on the given sensor given the property key

- **Parameters:**
  - `sensorId` — - id of the sensor
  - `key` — - String representing the property
- **Returns:** value of the property

`public Object getSensorProperty(String sensorId, String key, Serializable defaultValue)`

Returns a property value on the given sensor given the property key or the defaultValue if the property is not present

- **Parameters:**
  - `sensorId` — - id of the sensor
  - `key` — - String representing the property
  - `defaultValue` — - default value
- **Returns:** value of the property or the defaultValue given

`public void setSensorProperty(String sensorId, String key, Object value)`

Sets a property on the sensor in the current context

- **Parameters:**
  - `sensorId` — - unique String representing a sensor
  - `key` — - String representation of a valid PropertyType

- `value` — - value to set for the associated property

`public void removeSensorProperty(String sensorId, String key)`

Removes a property on the sensor

- **Parameters:**
  - `sensorId` — - unique String representing a sensor
  - `key` — - String representation of a valid PropertyType

`public void setLocalProperty(String key, Object value)`

Sets a property in local storage.

- **Parameters:**
  - `key` — the key
  - `value` — the value

`public Object getLocalProperty(String key)`

Retrieves a property from local storage.

- **Parameters:** `key` — the key
- **Returns:** the associated value (or {@code null} if none present)

`public boolean hasLocalProperty(String key)`

Determines if a local property with the given key exists.

- **Parameters:** `key` — the key
- **Returns:** {@code true} if the property exists, {@code false} otherwise)

`public byte[] copyOfRange(byte[] original, int from, int to)`

Copies the specified range of the specified array into a new array. The initial index of the range (from) must lie between zero and original.length, inclusive. The value at original[from] is placed into the initial element of the copy (unless from == original.length or from == to). Values from subsequent elements in the original array are placed into subsequent elements in the copy. The final index of the range (to), which must be greater than or equal to from, may be greater than original.length, in which case (byte)0 is placed in all elements of the copy whose index is greater than or equal to original.length - from. The length of the returned array will be to - from.

- **Parameters:**
  - `original` — - byte array to copy from

- `from` — - initial index to start from (inclusive)
- `to` — - final index of the range to be copied (exclusive)

- **Returns:** byte array - a new array containing the specified range from the original array,

  truncated or padded with zeros to obtain the required length

`public void postObservation(String sensorId, double value)`

Used to post a single observation with a single double value (64-bit IEEE 754 floating point) for the given sensor with the current system timestamp

- **Parameters:**
  - `sensorId` — - unique String representing a sensor
  - `value` — - double value

`public void postObservations(String sensorId, double... values)`

Used to post a single observation with multiple double values (64-bit IEEE 754 floating point) for the given sensor with the current system timestamp

- **Parameters:**
  - `sensorId` — - unique String representing a sensor
  - `values` — - 1 or more double values

`public String sendModbusCommand(int slaveAddress, int functionCode, int startRegister, int registers)`

Sends a modbus command to the device and returns the result as a String hex value Returns a null response if there was no response received within 5s.

- **Parameters:**
  - `slaveAddress` — - integer representing the slave address
  - `functionCode` — - integer representing the modbus function to perform
  - `startRegister` — - integer represeting the first register to start from
  - `registers` — - integer representing the total number of registers to read/write to/from
- **Returns:** String - response as a hex value or null if no response within 5s.

`public String sendModbusCommandWithTimeout(int slaveAddress, int functionCode, int startRegister, int registers, int timeoutMs)`

Sends a modbus command to the device and returns the result as a String hex value Returns a null response if there was no response received within the given custom timeout

- Parameters:
  - `slaveAddress` — - integer representing the slave address
  - `functionCode` — - integer representing the modbus function to perform
  - `startRegister` — - integer represeting the first register to start from
  - `registers` — - integer representing the total number of registers to read/write to/from
  - `timeoutMs` — - timeout in milliseconds
- Returns: String - response as a hex value or null if no response within 5s.

`public String sendModbusWriteInt32CommandInt(int slaveAddress, int startRegister, int valueToWrite)`

Writes a 4-byte integer value to two contiguous modbus registers.

- Parameters:
  - `slaveAddress` — Modbus device slave address
  - `startRegister` — Start register to write to
  - `valueToWrite` — 4-byte integer value to write
- Returns: The modbus device's response

`public String sendModbusMultipleWriteFloat32CommandFloat(int slaveAddress, int startRegister, float valueToWrite)`

Writes a 4-byte IEEE float32 value to two contiguous modbus registers.

- Parameters:
  - `slaveAddress` — Modbus device slave address
  - `startRegister` — Start register to write to
  - `valueToWrite` — 4-byte float value to write
- Returns: The modbus device's response

`public String getModbusDataPayload(String modbusResponse)`

Given a modbus response, returns the data portion of the response as a hex string

- Parameters: `modbusResponse` — - a full hex modbus response
- Returns: String - data portion of the modbus response as hex

`public double convertUnit(double value, String from, String to)`

Converts a numeric value from one unit to the other.

- Parameters:

    - `value` — The value to convert
    - `from` — Unit of the original value
    - `to` — Desired output unit
- **Returns:** The value converted to the output unit.