Getting/Setting Attributes

At the heart of any application in which your Arduino interacts with an Afero development board are the actions of **getting** and **setting** attribute values. When you create a device Profile using the Afero Profile Editor, you declare a set of "Attributes", which can be thought of as variables that represent the state of your device, including values monitored and controlled by the MCU, and any GPIOs on the dev board that you might be using.

The functions described on this page allow your MCU to query and set the attribute values, as currently maintained by the Afero Cloud:

- `af_lib_get_attribute()`
- `af_lib_set_attribute()`

It's important to keep in mind that the attribute values are represented in multiple places in your project:

- Your MCU code presumably interfaces directly with your product hardware. As an example, the thermometer in your automated ice cream maker knows what the temperature is at all times; this is represented as a variable in your MCU code, and that variable has a corresponding attribute defined in the device Profile.
- ASR, which has been programmed with the device Profile of interest, maintains local copies of all associated attribute values.
- The Afero Cloud also knows the Profile for your device, and has copies of the attribute values.
- Your Profile *might* define "Default Values" for one or more attributes. If desired, these default values can be reloaded at boot time to reset the state of your device.

Obviously, it's critical that all representations of attribute values in the system stay in sync. Your MCU code, in conjunction with the Afero Cloud, will be responsible for this data synchronization, and it will be

performed through the use of the two functions to **get** and **set** attribute values described below. The rules by which this synchronization takes place are detailed in Attribute Value Change Rules.

af_lib_get_attribute()

Description

Request the value of a specified attribute. The `af_lib_get_attribute()` call only queues the request; your `attrEventCallback()` function will be called when the value is returned by ASR. The attribute value is returned via the callback.

When called:

- If no default value has been defined, `af_lib_get_attribute` will return `length=0`, `value=NULL`.
- If the default value has been defined and the attribute has not been modified, `af_lib_get_attribute` will return the default value.
- If the attribute value has been modified by the service or MCU, `af_lib_get_attribute` will return the most recent value set by either the service or the MCU.

Syntax

```
af_lib_get_attribute(af_lib_t *af_lib, const uint16_t attr_id)
```

Parameters

`af_lib`	Pointer to the active aflib instance.
`attr_id`	The attribute ID of the value you are requesting.

Returns

An afLib result code indicating whether request was queued successfully.

Example

> The code examples for the functions below illustrate the use of SPI communication. To use UART instead, only the minimal changes in afLib3 instantiation, as described in the afLib Lifecycle section, would be required.

In the example below, `af_lib_get_attribute()` is called in `loop()` to obtain the value of `AF_MY_ATTRIBUTE`. If the call returns success, we set `shouldGetAttr` to false, since we won't need to call `af_lib_get_attribute()` again. The value of the attribute of interest is actually obtained when `attrEventCallback()` is called. In that callback, we use the event type and attribute ID to select which action should be taken (in this case, we simply print to console).

```
#include <SPI.h>
#include "af_lib.h"
#include "arduino_spi.h"
#include "af_module_commands.h"
#include "af_module_states.h"
#include "arduino_transport.h"

// Pin Defines assume Arduino Uno
#define CS_PIN               10
#define INT_PIN               2

boolean shouldGetAttr = true;
af_lib_t *af_lib;

void attrEventCallback(const af_lib_event_type_t eventType,
                       const af_lib_error_t error,
                       const uint16_t attributeId,
                       const uint16_t valueLen,
                       const uint8_t* value) {
  // You will typically switch on the eventType in attrEventCallback(), to decide how you need
```

```
to handle the event.
  // For the sake of clarity, here we show only the event type associated with responses to
af_lib_getAttribute.
  // The range of event types will be detailed elsewhere.

  switch (eventType) {
    case AF_LIB_EVENT_ASR_GET_RESPONSE:
      // Response to af_lib_get_attribute()
      if (attributeId == AF_MY_ATTRIBUTE) {
        Serial.print("*attrEventCallback* Attr id: ");
        Serial.print(attributeId);
        Serial.print(" value: ");
        Serial.println(*value);
      }
        break;

    default:
        break;
    }

}

void setup() {
  Serial.begin(115200);
  while (!Serial) { // wait for serial port
    ;
  }

  // Initialize afLib
  af_transport_t* arduinoSPI = arduino_transport_create_spi(CS_PIN);
  af_lib = af_lib_create_with_unified_callback(attrEventCallback, arduinoSPI);
  arduino_spi_setup_interrupts(af_lib, digitalPinToInterrupt(INT_PIN));
}

void loop() {
  if (shouldGetAttr) {
    // Attribute name definitions are provided by device-description.h from your profile
    int result = af_lib_get_attribute(af_lib, AF_MY_ATTRIBUTE);
    if (result == AF_SUCCESS) {
      shouldGetAttr = false;
    }
    else {
      Serial.print("*af_lib_get_attribute* returned error: ");
      Serial.println(result);
    }
```

```
  }

  // Give afLib processing time by calling af_lib_loop(af_lib) in sketch's loop()
  af_lib_loop(af_lib);
}
```

af_lib_set_attribute()

Description

Request to change the value of a specified attribute. This call only queues the request. Your `attrEventCallback()` function will be called when afLib has handled the request.

It's important to recognize that `af_lib_set_attribute` may be called in several ways (e.g., from MCU code, as a result of actions by the mobile app, etc.), and for multiple attribute types (i.e., GPIO, MCU). Your `attrEventCallback()` will respond to a given attribute set event depending on how it was called, and for which attribute type. This is detailed in Callback & Events.

Syntax

There are several versions of the `af_lib_set_attribute` API that allow you to pass values of different byte-lengths to afLib3. Here are the currently supported types:

```
int af_lib_set_attribute_bool(af_lib_t *af_lib, const uint16_t attr_id, const bool value);

int af_lib_set_attribute_8(af_lib_t *af_lib, const uint16_t attr_id, const int8_t value);

int af_lib_set_attribute_16(af_lib_t *af_lib, const uint16_t attr_id, const int16_t value);

int af_lib_set_attribute_32(af_lib_t *af_lib, const uint16_t attr_id, const int32_t value);
```

```
int af_lib_set_attribute_64(af_lib_t *af_lib, const uint16_t attr_id, const int64_t value);

int af_lib_set_attribute_str(af_lib_t *af_lib, const uint16_t attr_id, const uint16_t
value_len, const char *value);

int af_lib_set_attribute_bytes(af_lib_t *af_lib, const uint16_t attr_id, const uint16_t
value_len, const uint8_t *value);
```

Parameters

`af_lib`	Pointer to the active aflib instance.
`attr_id`	The attribute ID for which you are requesting the value.
`valueLen`	The size in bytes of the attribute value. This parameter only needed when value size is not indicated by value type.
`value`	The new value for the attribute.

Returns

An afLib result code indicating whether request was queued successfully.

Example

In this example, our code will use `af_lib_set_attribute()` to set the value of a GPIO attribute.

In the case of the GPIO attribute (named `AF_GPIO_1`), ASR will change the value of the attribute to 1, set the level of the corresponding GPIO appropriately, then call the MCU's `attrEventCallback()` method, allowing us perform any actions desired once we know the operation is complete. In the example, we simply print.

Notice that the *event type* observed in `attrEventCallback()` after `af_lib_set_attribute()` depends on the type of attribute involved. For a GPIO attribute, the callback will have an event type of `AF_LIB_EVENT_ASR_SET_RESPONSE`. The event type provides an easy way for your code to differentiate the source and type of attribute-set taking place, and respond accordingly.

```
#include <SPI.h>
#include "af_lib.h"
#include "arduino_spi.h"
#include "af_module_commands.h"
#include "af_module_states.h"
#include "arduino_transport.h"
#include "profile/device-description.h"

// Pin Defines assume Arduino Uno
#define CS_PIN                10
#define INT_PIN                2

boolean shouldSetGPIOAttr = true;
boolean shouldSetMCUAttr = true;
uint16_t newGPIOvalue = 1;
af_lib_t *af_lib;

void attrEventCallback(const af_lib_event_type_t eventType,
                       const af_lib_error_t error,
                       const uint16_t attributeId,
                       const uint16_t valueLen,
                       const uint8_t* value) {

  // As with example above, only event types relevant to this task are shown, for clarity
  switch (eventType) {
    case AF_LIB_EVENT_ASR_SET_RESPONSE:
      // Response to af_lib_set_attribute() for a GPIO attribute
      Serial.println("It appears that af_lib_set_attribute was called for a GPIO attribute");
      if (attributeId == AF_GPIO_1) {
        Serial.print("*attrEventCallback* Attr id: ");
        Serial.print(attributeId);
        Serial.print(" value: ");
        Serial.println(*value);
      }
        break;

    default:
        break;
```

```
    }
}

void setup() {
  Serial.begin(115200);
  while (!Serial) { // wait for serial port
    ;
  }

  // Initialize afLib
  af_transport_t* arduinoSPI = arduino_transport_create_spi(CS_PIN);
  af_lib = af_lib_create_with_unified_callback(attrEventCallback, arduinoSPI);
  arduino_spi_setup_interrupts(af_lib, digitalPinToInterrupt(INT_PIN));
}

void loop() {
  if (shouldSetGPIOAttr) {
    // Attribute name definitions are provided by device-description.h from your profile
    int result = af_lib_set_attribute_16(af_lib, AF_GPIO_1, newGPIOvalue);
    if (result == AF_SUCCESS) {
      shouldSetGPIOAttr = false;
    }
    else {
      Serial.print("*set_attribute* for GPIO attribute returned error: ");
      Serial.println(result);
    }
  }

  // Give afLib processing time by calling af_lib_loop(af_lib) in sketch's loop()
  af_lib_loop(af_lib);
}
```

Next: *Callbacks and Events*

Updated September 25, 2019