Callbacks and Events

Calls to `af_lib_get_attribute()` and `af_lib_set_attribute*()` are enqueued for handling by afLib; and once handled, result in execution of a callback function named `attrEventCallback()`. You must define `attrEventCallback()` in your MCU code, and pass it as an argument to `af_lib_create()`. This callback contains your application-specific code to respond to attribute changes.

Your `attrEventCallback()` will be called with an **event type** and an error code. You'll use these data to select an appropriate response. This page describes how your code can distinguish the events and respond accordingly.

attrEventCallback()

Description

Application callback that is executed any time ASR has data to communicate with the MCU.

Syntax

```
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)
```

Parameters

`eventType`	The event type for this request.
`error`	An afLib result code
`attribute_id`	The ID of the attribute involved.
`value_len`	The size in bytes of the attribute value.
`value`	The new value for the attribute.

Event Types

The following event types are defined in `af_lib.h`:

EVENT TYPE	INTERPRETATION
`AF_LIB_EVENT_UNKNOWN`	Not normally seen; intended for debugging.
`AF_LIB_EVENT_ASR_SET_RESPONSE`	Response to `af_lib_set_attribute()` for an ASR attribute.
`AF_LIB_EVENT_MCU_SET_REQ_SENT`	Request from `af_lib_set_attribute()` for an MCU attribute has been sent to ASR.
`AF_LIB_EVENT_MCU_SET_REQ_REJECTION`	Request from `af_lib_set_attribute()` for an MCU attribute was rejected by ASR.
`AF_LIB_EVENT_ASR_GET_RESPONSE`	Response to `af_lib_get_attribute()`.
`AF_LIB_EVENT_MCU_DEFAULT_NOTIFICATION`	Unsolicited default notification for an MCU attribute.
`AF_LIB_EVENT_ASR_NOTIFICATION`	Unsolicited notification of non-MCU attribute change.
`AF_LIB_EVENT_MCU_SET_REQUEST`	Request from ASR to MCU to set an MCU attribute, requires a call to `af_lib_send_set_response()`.

EVENT TYPE	INTERPRETATION
`AF_LIB_EVENT_COMMUNICATION_BREAKDOWN`	The communication between the MCU and the ASR seems to have stopped, take the appropriate action (i.e. rebooting the ASR).

Returns

None.

Special Case: MCU Confirmation of Sets Requested by ASR

Requests from ASR to set an MCU attribute value will arrive at the MCU via an `AF_LIB_EVENT_MCU_SET_REQUEST` event in the `attrEventCallback()`. It is the responsibility of the MCU to act on the request as appropriate, and then respond True or False, indicating whether or not the set was completed as requested, via the `af_lib_send_set_response()` call.

af_lib_send_set_response()

Description

Call required in response to callback event AF_LIB_EVENT_MCU_SET_REQUEST, which is used by MCU to confirm/deny that the set request by ASR was successfully completed by the MCU. If the MCU was able to set the attribute to the requested value, then this call can just return the values passed to it in this callback. If the MCU was unable to set the requested value, then this call should return the value that the MCU will use instead of the value requested.

Syntax

```
void af_lib_send_set_response(af_lib_t *af_lib,
                        const uint16_t attribute_id,
                        bool set_succeeded,
                        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 set request was sent.
`set_succeeded`	Boolean indicating whether or not MCU successfully changed attribute the value.
`valueLen`	The size in bytes of the attribute value.
`value`	The value for the attribute.

Returns

Error.

Example 1: Translating End-User Actions and afLib3 Function Calls into Events

`attrEventCallback()` will be called by afLib3 whenever ASR has information for the MCU. Your code will depend on the eventType parameter to determine why the callback has been fired, and therefore how you want to respond.

Let's consider a few specific cases, just to make this clear:

- If the button is pressed on an Afero dev board (changing the value of GPIO 3 directly), the `attrEventCallback()` will be executed with eventType `AF_LIB_EVENT_ASR_NOTIFICATION`.
- If your MCU code calls `af_lib_get_attribute()`, the value will be returned via the callback with eventType `AF_LIB_EVENT_ASR_GET_RESPONSE`.
- Consider that an end-user taps a control in your mobile application to change an attribute value. If the attribute changed is a GPIO, your code will be informed of the change by `attrEventCallback()` with eventType `AF_LIB_EVENT_ASR_NOTIFICATION`.

However…

- If the attribute changed by the mobile app is an MCU attribute, your code will receive `AF_LIB_EVENT_MCU_SET_REQUEST`.
- Your MCU application should watch for event type AF_LIB_EVENT_MCU_DEFAULT_NOTIFICATION. For each occurrence of this event, it is the responsibility of MCU to call the appropriate version of `af_lib_set_attribute()` with the desired value of the corresponding attribute. This will usually be one of two variants:
    - If your application is designed so that a reboot should cause a given attribute to reset to the value specified in the device Profile, then `af_lib_set_attribute()` should be called with the value sent in the AF_LIB_EVENT_MCU_DEFAULT_NOTIFICATION event. This scheme can be useful when you want to be able to change default attribute values for devices in the field via an OTA update.
    - If your application is designed so that the current value of an attribute as held by the MCU is the value to be used, regardless of reboots, then `af_lib_set_attribute()` should be called with the attribute value held by the MCU. This scheme can be useful when your application assumes the MCU will ensure continuity of attribute values despite any reboots of ASR, for example, because of a Profile OTA.

  IMPORTANT: The AF_LIB_EVENT_MCU_DEFAULT_NOTIFICATION events will be sent during the "attribute update-all" sequence that is part of the boot process. This sequence happens **before** your application enters the AF_MODULE_STATE_INITIALIZED state – and you must **not** call `af_lib_set_attribute()` until after you enter AF_MODULE_STATE_INITIALIZED. This means your application code may need to temporarily store each pending attribute value upon receipt of the AF_LIB_EVENT_MCU_DEFAULT_NOTIFICATION, and then use the stored value(s) once the application state reaches AF_MODULE_STATE_INITIALIZED.

Example 2: Structure of a Prototypical attrEventCallback()

Typically, your callback code will branch on the eventType, and within each case, will branch again depending on the attributeID. Given that, below is an example "shell" of an `attrEventCallback()` that has a case for all eventTypes, and is also set up with cases for all system states, in the case of an `AF_LIB_EVENT_ASR_NOTIFICATION` event for attribute `AF_SYSTEM_ASR_STATE`. You might find this useful to copy/paste into your code:

```
// Callback executed any time ASR has information for the MCU
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) {

  switch (eventType) {
    case AF_LIB_EVENT_UNKNOWN:
      // Should never occur (indicates an eventType = 0); primarily for debugging
      break;

    case AF_LIB_EVENT_ASR_SET_RESPONSE:
      // Response to af_lib_set_attribute() for an ASR attribute
      break;

    case AF_LIB_EVENT_MCU_SET_REQ_SENT:
      // Request from af_lib_set_attribute() for an MCU attribute has been sent to ASR
      break;

    case AF_LIB_EVENT_MCU_SET_REQ_REJECTION:
      // Request from af_lib_set_attribute() for an MCU attribute was rejected by ASR
      break;

    case AF_LIB_EVENT_ASR_GET_RESPONSE:
      // Response to af_lib_get_attribute()
      break;

    case AF_LIB_EVENT_MCU_DEFAULT_NOTIFICATION:
      // Unsolicited default notification for an MCU attribute
      break;

    case AF_LIB_EVENT_ASR_NOTIFICATION:
      // Unsolicited notification of non-MCU attribute change
```

```
      switch (attributeId) {

        case AF_SYSTEM_ASR_STATE:
          switch (value[0]) {
            case AF_MODULE_STATE_REBOOTED:
              // ASR has rebooted. MCU Attributes are erased and must be reset
              break;

            case AF_MODULE_STATE_LINKED:
              // Linked - ASR has rebooted (MCU Attributes erased) and then linked
              break;

            case AF_MODULE_STATE_UPDATING:
              // Updating - There's an OTA in progress
              break;

            case AF_MODULE_STATE_UPDATE_READY:
              // Update ready - OTA is complete; reboot needed to begin using the new code
              break;

            case AF_MODULE_STATE_INITIALIZED:
              // Initialized - ASR now ready to service requests
              break;

            case AF_MODULE_STATE_RELINKED:
              //Relinked - ASR has relinked with cloud
              break;

            default:
              Serial.print("Unexpected state - "); Serial.println(value[0]);
              break;
          }
          break;

        default:
          break;
      }
      break;

    case AF_LIB_EVENT_MCU_SET_REQUEST:
      // Request from ASR to MCU to set an MCU attribute, requires a call to
af_lib_send_set_response()
      switch (attributeId) {

        default:
```

```
            break;
        }
        break;

      default:
        break;
    }
}
```

Example 3: afBlink

This afBlink example uses a Teensy to control the LED onboard an Afero dev board. The LED can be made to start/stop blinking by a tap on a UI control in the mobile app, or by a press on the dev board button. The key lesson from this example comes from understanding how the code in `attrEventCallback()` distinguishes between various events, and handles them accordingly. Here are four cases:

- In the case of a tap on the “Start” button in the mobile app UI…
  - The mobile app translates the button tap into an `af_lib_set_bool()` for attribute `AF_BLINK`.
  - As a result, `attrEventCallback()` will be called with an eventType `AF_LIB_EVENT_MCU_SET_REQUEST` and attributeId `AF_BLINK`.
  - So, when we get eventType `AF_LIB_EVENT_MCU_SET_REQUEST` belonging to attribute `AF_BLINK`, our code sets the value of our variable `blinking`, and then calls `af_lib_send_set_response` to confirm that MCU was able to update its state.
  - Within our `loop()` function, if `blinking` is true, we call `toggleLED()` every two seconds, which blinks the LED!
- In the case of a button press on the Afero dev board…
  - The Afero dev board changes the local attribute value and sends an update, which causes…
  - `attrEventCallback()` to be called with an eventType `AF_LIB_EVENT_ASR_NOTIFICATION` and attributeId `AF_MODULO_BUTTON`.
  - When we get eventType `AF_LIB_EVENT_ASR_NOTIFICATION`, belonging to attribute `AF_MODULO_BUTTON`, our code toggles the value of our global `curButtonValue`.

- Within our `loop()` function, if our `curButtonValue` is different from `prevButtonValue` (that is, if the button value has changed), then we call `af_lib_set_bool()` on `AF_BLINK`, setting that attribute, which controls blinking.
- *That* `af_lib_set_bool()` call triggers the same sequence of events as example #1 above, which results in the LED blinking.

- Any time the Afero dev board reboots…

  The MCU is required to call `set_attribute` for each MCU attribute whenever ASR reboots, as described in MCU Attributes and Update All. So:

  - Upon reboot, `attrEventCallback()` will be called with an eventType `AF_LIB_EVENT_ASR_NOTIFICATION` and attributeId `AF_SYSTEM_ASR_STATE`.
  - Presented that combination of eventType and attributeId, our code checks the `value` delivered via the callback; in this case, that value is `AF_MODULE_STATE_REBOOTED`.
  - Our code responds by setting `initializationPending` and `attr1SyncPending` to true. Both values are checked in `loop()`:
    - Any time `initializationPending` is true, we avoid making afLib calls until we receive the `AF_SYSTEM_ASR_STATE` `AF_MODULE_STATE_INITIALIZED` signal that tells us that ASR is ready to handle requests.
    - If `attr1SyncPending` is true, we call `af_lib_set_attribute_bool` for attribute 1 (`AF_BLINK`), setting it to the current value of the `blinking` variable.

- If the Afero dev board receives an OTA update…
  - When the OTA is complete, `attrEventCallback()` will be called with an eventType `AF_LIB_EVENT_ASR_NOTIFICATION` and attributeId `AF_SYSTEM_ASR_STATE`.
  - We check the `value` delivered via the callback; in this case it's `AF_MODULE_UPDATE_READY`.
  - Our code responds by setting `rebootPending` to true. This value is checked in `loop()`, and if true, we trigger a reboot.

```
#include <SPI.h>

#include "af_lib.h"
#include "arduino_spi.h"
```

```
#include "arduino_uart.h"
#include "af_module_commands.h"
#include "af_module_states.h"
#include "arduino_transport.h"

#include "profile/afBlink/device-description.h"

#define ARDUINO_USE_SPI                   1

// Check for teensy and abort if not
#if defined(TEENSYDUINO)
#else
#error "Sorry, this example written for teensy. E.G. pins are defined accordingly."
#endif

#define INT_PIN                   14    // Modulo uses this to initiate communication
#define CS_PIN                    10    // Standard SPI chip select (aka SS)
#define RESET                     21    // This is used to reboot the Modulo when the Teensy
boots
#define RX_PIN                    7
#define TX_PIN                    8

// Modulo LED is active low
#define LED_OFF                   1
#define LED_ON                    0

#define BLINK_INTERVAL            2000  // 2 seconds


af_lib_t* af_lib = NULL;
bool initializationPending = false; // If true, we're waiting on AF_MODULE_STATE_INITIALIZED
bool rebootPending = false;         // If true, a reboot is needed, e.g. if we received an OTA
firmware update.
bool attr1SyncPending = false;      // If true (e.g. after reboot), we need to set MCU attr 1

volatile long lastBlink = 0;                    // Time of last blink
volatile bool blinking = false;                 // Track whether LED is blinking; represented
by attribute AF_BLINK
volatile bool moduloLEDIsOn = false;            // Track whether the Modulo LED is on
uint16_t prevButtonValue = 1;                   // Track the button value...
uint16_t curButtonValue = prevButtonValue;      // ...so we know when it has changed


void toggleModuloLED() {
  setModuloLED(!moduloLEDIsOn);
```

```
}

void setModuloLED(bool on) {
  if (moduloLEDIsOn != on) {
    int16_t attrVal = on ? LED_ON : LED_OFF; // Modulo LED is active low

    int timeout = 0;
    while (af_lib_set_attribute_16(af_lib, AF_MODULO_LED, attrVal) != AF_SUCCESS) {
      delay(10);
      af_lib_loop(af_lib);
      timeout++;
      if (timeout > 500) {
        // If we haven't been successful after 5 sec (500 tries, each after 10 msec delay)
        // we assume we're in some desperate state, and reboot
        pinMode(RESET, OUTPUT);
        digitalWrite(RESET, 0);
        delay(250);
        digitalWrite(RESET, 1);
        return;
      }
    }

    moduloLEDIsOn = on;
  }
}

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) {

  switch (eventType) {
    case AF_LIB_EVENT_ASR_SET_RESPONSE:
      // Response to af_lib_set_attribute() for an ASR attribute
      break;

    case AF_LIB_EVENT_MCU_SET_REQ_SENT:
      // Request from af_lib_set_attribute() for an MCU attribute has been sent to ASR
      break;

    case AF_LIB_EVENT_MCU_SET_REQ_REJECTION:
      // Request from af_lib_set_attribute() for an MCU attribute was rejected by ASR
      break;
```

```
    case AF_LIB_EVENT_ASR_GET_RESPONSE:
      // Response to af_lib_get_attribute()
      break;

    case AF_LIB_EVENT_MCU_DEFAULT_NOTIFICATION:
      // Unsolicited default notification for an MCU attribute
      break;

    case AF_LIB_EVENT_ASR_NOTIFICATION:
      // Unsolicited notification of non-MCU attribute change
      switch (attributeId) {
        case AF_MODULO_LED:
          // Update the state of the LED based on the actual attribute value.
          moduloLEDIsOn = (*value == 0);
          break;

        case AF_MODULO_BUTTON:
          // curButtonValue is checked in loop(). If changed, will toggle the blinking state.
          curButtonValue = *(uint16_t*) value;
          break;

        case AF_SYSTEM_ASR_STATE:
          switch (value[0]) {
            case AF_MODULE_STATE_REBOOTED:
              initializationPending = true;   // Just rebooted, so we're not yet initialized
              attr1SyncPending = true;    // Set Attribute 1 (happens in loop())
              break;

            case AF_MODULE_STATE_LINKED:
              break;

            case AF_MODULE_STATE_UPDATING:
              break;

            case AF_MODULE_STATE_UPDATE_READY:
              // OTA has been received - reboot needed to use new code
              rebootPending = true;
              break;

            case AF_MODULE_STATE_INITIALIZED:
              // ASR signals that it's ready
              initializationPending = false;
              break;

            case AF_MODULE_STATE_RELINKED:
```

```
            break;
          }
          break;
      }
      break;

    case AF_LIB_EVENT_MCU_SET_REQUEST:
      // Request from ASR to MCU to set an MCU attribute, requires a call to
af_lib_send_set_response()
      switch (attributeId) {

        case AF_BLINK:
          // This MCU attribute controls whether we should be blinking.
          blinking = (*value == 1);
          af_lib_send_set_response(af_lib, AF_BLINK, true, valueLen, value);
          break;
      }
      break;
  }
}

void setup() {

  Serial.begin(9600);

  while (!Serial) {;} // wait for serial port

  Serial.println("Starting sketch: afBlink");

  Serial.println("Teensy resetting Modulo");
  pinMode(RESET, OUTPUT);
  digitalWrite(RESET, 0);
  delay(250);
  digitalWrite(RESET, 1);

  // Start the sketch awaiting initialization
  initializationPending = true;

  Serial.print("Configuring communications...sketch will use SPI");
  Serial.println("SPI");
  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() {
  // Give the afLib state machine some time.
  af_lib_loop(af_lib);

  if (initializationPending) {
    // If we're awaiting initialization, don't bother checking/setting attributes
  } else {

    // If we were asked to reboot (e.g. after an OTA firmware update), make the call here in
loop().
    // In order to make this fault-tolerant, we'll continue to retry if the command fails.
    if (rebootPending) {
      int retVal = af_lib_set_attribute_32(af_lib, AF_SYSTEM_COMMAND,
AF_MODULE_COMMAND_REBOOT);
      rebootPending = (retVal != AF_SUCCESS);
      if (!rebootPending) {
        // Reboot command sent successfully; now awaiting AF_MODULE_STATE_INITIALIZED
        initializationPending = true;
      }
    }

    // After reboot, need to inform service of value of all MCU attributes (in this case just
attr 1)
    if (attr1SyncPending) {
      if (af_lib_set_attribute_bool(af_lib, AF_BLINK, blinking) == AF_SUCCESS) {
        attr1SyncPending = false;
      }
    }

    // Modulo button toggles 'blinking'
    if (prevButtonValue != curButtonValue) {
      if (af_lib_set_attribute_bool(af_lib, AF_BLINK, !blinking) == AF_SUCCESS) {
        blinking = !blinking;
        prevButtonValue = curButtonValue;
      }
    }

    // Flash the LED whenever the 'blinking' value is true
    if (blinking) {
      if (millis() - lastBlink > BLINK_INTERVAL) {
        toggleModuloLED();
        lastBlink = millis();
      }
    }

  }
}
```

Next: *MCU Attributes - Update All*

Updated September 25, 2019