afLib to afLib2 Migration Guide

If you have current projects using afLib, there is no immediate requirement to convert your code to afLib2. However, for new projects we strongly recommend using afLib2 to ensure the latest support is available to you. Converting your project from afLib to afLib2 is not particularly complex and is recommended if you make any larger upgrades to your existing projects.

In general, converting from afLib to afLib2 requires replacing the C include files to reference the new API, and renaming afLib C++ API calls to the new C ones. Follow the steps below:

1 Replace afLib C++ includes with afLib2 includes:

Old

```
#include <iafLib.h>
#include <ArduinoSPI.h>
#include <ArduinoUART.h>
#include <ModuleCommands.h>
#include <ModuleStates.h>
```

New

```
#include "af_lib.h"
#include "af_logger.h"
#include "arduino_logger.h"
#include "arduino_spi.h"
#include "arduino_uart.h"
#include "af_module_commands.h"
#include "af_module_states.h"
#include "arduino_transport.h"
```

2 Create a new af_lib reference:

Old

```
iafLib *aflib;
```

SPI

```
afTransport *arduinoUART = new ArduinoUART(RX_PIN, TX_PIN, &Serial);
aflib = iafLib::create(attrSetHandler, attrNotifyHandler, &Serial,
arduinoUART);
```

UART

```
afTransport *arduinoSPI = new ArduinoSPI(CS_PIN, &Serial);
aflib = iafLib::create(digitalPinToInterrupt(INT_PIN), ISRWrapper,
attrSetHandler, attrNotifyHandler, &Serial, arduinoSPI);
```

New

```
af_lib_t *af_lib;
```

SPI

```
af_transport_t *arduinoSPI = arduino_transport_create_spi(CS_PIN);
af_lib = af_lib_create(attrSetHandler, attrNotifyHandler, arduinoSPI);
arduino_spi_setup_interrupts(af_lib, digitalPinToInterrupt(INT_PIN));
```

UART

```
af_transport_t *arduinoUART = arduino_transport_create_uart(RX_PIN, TX_PIN);
af_lib = af_lib_create(attrSetHandler, attrNotifyHandler, arduinoUART);
```

3 Remove declaration of `ISRWrapper()`:

The original version of afLib required a declaration of ISRWrapper:

```
void ISRWrapper() {
    if (aflib) {
        aflib->mcuISR();
    }
}
```

afLib2 does not require this declaration, so you can remove it from any code you are migrating.

4 Convert old API calls to new:

Old

```
int afLib::getAttribute(const uint16_t attrId)
int afLib::setAttributeBool(const uint16_t attrId, const bool value)
int afLib::setAttribute8(const uint16_t attrId, const int8_t value)
int afLib::setAttribute16(const uint16_t attrId, const int16_t value)
int afLib::setAttribute32(const uint16_t attrId, const int32_t value)
int afLib::setAttribute64(const uint16_t attrId, const int64_t value)
int afLib::setAttributeStr(const uint16_t attrId, const String &value)
int afLib::setAttributeCStr(const uint16_t attrId, const uint16_t valueLen, const
char *value)
int afLib::setAttributeBytes(const uint16_t attrId, const uint16_t valueLen, const
uint8_t *value)
```

New

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

The `af_lib_t` parameter to the new calls is the `af_lib_t` reference created in step 2 above.

5 Modify error constant names.

The original version of afLib used a different format for error constants than does afLib2 (afErrors.h vs. af_errors.h). Briefly, you replace the leading “af” with “AF_”:

Old

```
#define afSUCCESS                    0    // Operation completed successfully
```

```
#define afERROR_NO_SUCH_ATTRIBUTE   -1    // Request was made for unknown attribute
id
#define afERROR_BUSY                -2    // Request already in progress, try again
#define afERROR_INVALID_COMMAND     -3    // Command could not be parsed
#define afERROR_QUEUE_OVERFLOW      -4    // Queue is full
#define afERROR_QUEUE_UNDERFLOW     -5    // Queue is empty
#define afERROR_INVALID_PARAM       -6    // Bad input parameter
```

New

```
#define AF_SUCCESS                   0    // Operation completed successfully
#define AF_ERROR_NO_SUCH_ATTRIBUTE  -1    // Request was made for unknown attribute
id
#define AF_ERROR_BUSY               -2    // Request already in progress, try again
#define AF_ERROR_INVALID_COMMAND    -3    // Command could not be parsed
#define AF_ERROR_QUEUE_OVERFLOW     -4    // Queue is full
#define AF_ERROR_QUEUE_UNDERFLOW    -5    // Queue is empty
#define AF_ERROR_INVALID_PARAM      -6    // Bad input parameter
```

6 Modify command enum names.

The original version of afLib used a different format for module commands than does afLib2 (ModuleCommands.h vs af_module_commands.h). Briefly, you add a leading “AF_”:

Old

```
MODULE_COMMAND_NONE
MODULE_COMMAND_REBOOT
```

New

```
AF_MODULE_COMMAND_NONE
AF_MODULE_COMMAND_REBOOT
```

7 Modify module state enum names.

The original version of afLib used a different format for module states than does afLib2 (ModuleStates.h vs af_module_states.h). Briefly, you add a leading “AF_”:

Old

```
MODULE_STATE_REBOOTED
MODULE_STATE_LINKED
```

```
MODULE_STATE_UPDATING
MODULE_STATE_UPDATE_READY
```

New

```
AF_MODULE_STATE_REBOOTED
AF_MODULE_STATE_LINKED
AF_MODULE_STATE_UPDATING
AF_MODULE_STATE_UPDATE_READY
```

8 Optional use of af_logger.

For Arduino, use of `af_logger` is optional; existing Arduino Serial debugging methods will still work as they always have.

- `af_logger` is new in afLib2; it provides a generic implementation to get debugging output from the Afero MCU Library code.
- Refer to `arduino_logger.cpp` in the library for an example of how we implement the `af_logger` interface on the Arduino platform.
- We provide the following `af_logger` utility calls:

```
af_logger_format_t:
    AF_LOGGER_BIN = 2,
    AF_LOGGER_OCT = 8,
    AF_LOGGER_DEC = 10,
    AF_LOGGER_HEX = 16

void af_logger_print_value(int32_t val);
void af_logger_print_buffer(const char* val);
void af_logger_print_formatted_value(int32_t val, af_logger_format_t format);
void af_logger_println_value(int32_t val);
void af_logger_println_buffer(const char* val);
void af_logger_println_formatted_value(int32_t val, af_logger_format_t format);
```

For example, on Arduino the following two lines of code are functionally identical:

```
Serial.println( int8variable, HEX );
```

```
af_logger_println_formatted_value( int8variable, AF_LOGGER_HEX );
```

For the full afLib2 developer documentation, please refer to [afLib2 for Arduino API](#).

Next: *MCU Coding Tips*

Updated June 6, 2018