M2MXMLC

Last changed on Apr 01, 2005 by Steve Emmons

Overview

M2MXMLC is an Open Source library written in the C language that helps embedded devices parse incoming M2MXML messages, construct outbound M2MXML messages, and manage a "Behavior State Machine" (or BSM) that supports all standard M2MXML commands and on-device behaviors.
M2MXMLC is divided into three modules: 1) **M2MMEM** provides simple memory allocate functions that the other modules use; 2) **M2MXML** provides parsing and formatting functions for M2MXML messages; and 3) **M2MBSM** provides functions to manage a standard implementation of M2MXML commands and on-device behaviors.
In addition, **M2MXMLC** relies on the Open Source XML parsing library called "EXPAT" which it includes for the programmer's convenience.

M2MMEM

M2MXMLC targets embedded environments. Since memory management in these environments often has multiple types of memory, **M2MMEM** allows the other modules of **M2MXMLC** to be written without reference to the ANSI-C memory management library. To do so, it provides simple functions for allocating (but not freeing) space from a fixed size memory buffer. In practice, the **M2MXMLC** modules are able to reuse a small number of buffers over and over, making this type of implementation sufficient for their needs.

M2MMEM only depends on the ANSI-C library defined by "string.h" and uses only "memset()", "strcpy()", and "strlen()" from it.

M2MMEM is defined in "m2mmem.h" and contains the following definitions:

M2MMEM_BUFFER

```
typedef struct {
    char *pStart;
    char *pNext;
    int usedCount;
    int maxSize;
} M2MMEM_BUFFER;
```

The **M2MMEM_BUFFER** maintains the current state of the memory allocated from a beginning fixed-size memory buffer. As a rule, its contents should not be access or modified in any way, since **M2MMEM** provides a complete set of functions for manipulating the **M2MMEM_BUFFER**.

M2MMEM_initializeBuffer

```
void M2MMEM_initializeBuffer(
    M2MMEM_BUFFER *pBuffer,
    char *pStart,
    int maxSize);
```

This function must be used to initialize an **M2MMEM_BUFFER**. It must be given an

M2MMEM_BUFFER (**pBuffer**), the beginning of a fixed-size memory buffer (**pStart**), and the maximum size of the buffer (**maxSize**).

M2MMEM_allocateSpace

```
void *M2MMEM_allocateSpace(
    M2MMEM_BUFFER *pBuffer,
    int size);
```

This function provides the most general form of allocation from **pBuffer** by returning a **void *** pointer to an allocation of **size** characters. If the result of this function is **NULL**, then the space cold not be allocated.

M2MMEM_allocateProperties

```
const char **M2MMEM_allocateProperties(
    M2MMEM_BUFFER *pBuffer,
    const char **ppSource);
```

This function aids in the allocation of a collection of M2MXML Properties or Attributes as prepared by the **EXPAT** XML parser. This involves an array of **char *** pointers terminating with **NULL** but expected in pairs where each pair represents a "key" and "value" combination. This function will allocate the array and the strings defined by **ppSource** out of the **M2MMEM_BUFFER** and will return the allocated copy or **NULL** if the space could not be allocated.

M2MMEM_allocateProperty

```
int M2MMEM_allocateProperty(
    M2MMEM_BUFFER *pBuffer,
    const char **ppProperties,
    const char *pKey,
    const char *pValue,
    int index);
```

This function allocates **pKey** and **pValue** strings from **pBuffer** and adds them to **ppProperties** at the given **index**. If successful, the function returns the index location of the next "key/value" pair; otherwise, it returns a negative number.

M2MMEM_allocatePropertySet

```
const char **M2MMEM_allocatePropertySet(
    M2MMEM_BUFFER *pBuffer,
    int propertyCount);
```

This function allocates an array of **char *** pointers from **pBuffer** that is large enough to hold "key/value" pairs up to **propertyCount**. If successful, it returns the pointer to the array; otherwise, it returns **NULL**.

M2MMEM_allocateString

```
const char *M2MMEM_allocateString(
    M2MMEM_BUFFER *pBuffer,
    const char *pSource);
```

This function allocates a copy of the string **pSource** from **pBuffer** and returns it as the value of the function; otherwise, it returns **NULL**.

M2MMEM_appendString

```
const char *M2MMEM_appendString(
    M2MMEM_BUFFER *pBuffer,
    const char *pSource);
```

This function differs from the "M2MMEM_allocate?" functions in that it assumes that **pBuffer** is intended to grow a contiguous string by appending new strings onto the end of the current buffer. It returns the pointer to the copy of **pSource** when successful, or **NULL** otherwise.

M2MXML

M2MXML can be though of in two basic parts: 1) parsing and 2) formatting. Parsing is the process of taking an incoming M2MXML message, extracting its important elements, and ensuring that they are handled properly. Formatting is the process of building up an outgoing M2MXML message that can be sent to a remote server. From the ANSI-C library, **M2MXML** relies only on "stdio.h", "string.h", and "time.h" and uses the following functions from them: atof(), itoa(), localtime(), memset(), strcmp(), and time().

In addition, **M2MXML** relies on the Open Source XML parser "**EXPAT**." While this is required, it is completely encapsulated by **M2MXML**. However, **EXPAT** has several configuration options that a developer may want to control. For example, unlike **M2MXML**, **EXPAT** relies on ANSI-C memory management by default, but it provides hooks that can be used to replace these functions should that be necessary.

M2MXML is defined in the "m2mxml.h" include file and contains the following:

Common M2MXML Definitions

The following M2MXML definitions are used for both parsing and formatting:

General	Comment
M2MXML_UUIDSIZE	The size of a UUID
M2MXML_NOTIMESTAMP	no timestamp specified
M2MXML_VERSION	current version of M2MXML
M2MXML Element Names	**Comment**
M2MXML_ELEMENT_MESSAGE	inbound/outbound
M2MXML_ELEMENT_COMMAND	inbound/outbound
M2MXML_ELEMENT_EXCEPTION	outbound-only
M2MXML_ELEMENT_PERCEPT	outbound-only
M2MXML_ELEMENT_PROPERTY	inbound/outbound
M2MXML_ELEMENT_RESPONSE	outbound-only

M2MXML Attributes Names	Comment
M2MXML_ATTRIBUTE_ADDRESS	
M2MXML_ATTRIBUTE_EXCEPTIONCODE	
M2MXML_ATTRIBUTE_ENTRYTYPE	
M2MXML_ATTRIBUTE_LABEL	
M2MXML_ATTRIBUTE_MESSAGE	
M2MXML_ATTRIBUTE_NAME	
M2MXML_ATTRIBUTE_PERCEPTTYPE	
M2MXML_ATTRIBUTE_RESULTCODE	
M2MXML_ATTRIBUTE_SEQUENCENUMBER	
M2MXML_ATTRIBUTE_TELEMTRYDEVICE	
M2MXML_ATTRIBUTE_TIMESTAMP	
M2MXML_ATTRIBUTE_TYPE	
M2MXML_ATTRIBUTE_UNITS	
M2MXML_ATTRIBUTE_VALUE	
M2MXML_ATTRIBUTE_VERSION	
M2MXML Command IDs	**Comment**
M2MXML_COMMANDID_UNKNOWN	for custom commands
M2MXML_COMMANDID_QUERYCONFIGURATION	
M2MXML_COMMANDID_REBOOT	
M2MXML_COMMANDID_REQUESTPERCEPT	
M2MXML_COMMANDID_SETANALOGOUTPUT	
M2MXML_COMMANDID_SETCONFIGURATION	
M2MXML_COMMANDID_SETSTRINGOUTPUT	
M2MXML_COMMANDID_TURNOFF	
M2MXML_COMMANDID_TURNON	
M2MXML Command Names	**Comment**
M2MXML_COMMAND_QUERYCONFIGURATION	
M2MXML_COMMAND_REBOOT	
M2MXML_COMMAND_REQUESTPERCEPT	
M2MXML_COMMAND_SETANALOGOUTPUT	
M2MXML_COMMAND_SETCONFIGURATION	

M2MXML_COMMAND_SETSTRINGOUTPUT	
M2MXML_COMMAND_SETUPTRANSDUCER	outbound-only
M2MXML_COMMAND_TURNOFF	
M2MXML_COMMAND_TURNON	
M2MXML Property Names	**Comment**
M2MXML_PROPERTY_DATA	
M2MXML_PROPERTY_DURATION	
M2MXML_PROPERTY_SETPOINT	
M2MXML Percept Type IDs	**Comment**
M2MXML_PERCEPTTYPEID_UNKNOWN	for error processing
M2MXML_PERCEPTTYPEID_ANALOG	
M2MXML_PERCEPTTYPEID_DIGITAL	
M2MXML_PERCEPTTYPEID_STRING	
M2MXML_PERCEPTTYPEID_LOCATION	
M2MXML Percept Type Names	**Comment**
M2MXML_PERCEPTTYPE_ANALOG	
M2MXML_PERCEPTTYPE_DIGITAL	
M2MXML_PERCEPTTYPE_STRING	
M2MXML_PERCEPTTYPE_LOCATION	
M2MXML Percept Type Property Names	**Comment**
M2MXML_PERCEPTPROPERTY_ANALOGIN	
M2MXML_PERCEPTPROPERTY_ANALOGOUT	
M2MXML_PERCEPTPROPERTY_DIGITALIN	
M2MXML_PERCEPTPROPERTY_DIGITALOUT	
M2MXML_PERCEPTPROPERTY_STRINGIN	
M2MXML_PERCEPTPROPERTY_STRINGOUT	
M2MXML_PERCEPTPROPERTY_LOCATIONIN	
M2MXML_PERCEPTPROPERTY_LOCATIONOUT	
M2MXML Entry Type IDs	**Comment**
M2MXML_ENTRYTYPEID_SCHEDULED	
M2MXML_ENTRYTYPEID_REQUESTED	
M2MXML_ENTRYTYPEID_BYEXCEPTION	

M2MXML_ENTRYTYPEID_MANUAL	
M2MXML_ENTRYTYPEID_BYACTUATOR	
M2MXML_ENTRYTYPEID_OTHHER	
M2MXML Exception Codes	**Comment**
M2MXML_EXCEPTIONCODE_HARDWARE	
M2MXML_EXCEPTIONCODE_SOFTWARE	
M2MXML_EXCEPTIONCODE_NOSEQNUM	
M2MXML_EXCEPTIONCODE_BADMESSAGE	
M2MXML Response Result Codes	**Comment**
M2MXML_RESULTCODE_SUCCESS	
M2MXML_RESULTCODE_RECEIVED	
M2MXML_RESULTCODE_SUBMITTED	
M2MXML_RESULTCODE_FAILEDDELIVERY	
M2MXML_RESULTCODE_FAILEDEXECUTION	
M2MXML_RESULTCODE_UNKNOWNCOMMAND	
M2MXML_RESULTCODE_BADARGUMENT	
M2MXML_RESULTCODE_WARNING	

Parsing

The parsing ability in the **M2MXML** module is based on calling a single function, **M2MXML_parseMessage**, that in turn will call one of several function pointers, each of whose arguments represent critical data extracted from the incoming M2MXML message.

M2MXML_HANDLERS

```
typedef int (*M2MXML_HANDLER_LOGERROR)(
    void *pAppData,
    int code,
    const char *pDescription);
typedef int (*M2MXML_HANDLER_UNKNOWN)(
    void *pAppData,
    const char *pUUID,
    const char *pSeq,
    const char *pAddress,
    const char *pCommand,
    const char **ppProperties,
    time_t timestamp);
typedef int (*M2MXML_HANDLER_REQUESTPERCEPT)(
    void *pAppData,
    const char *pUUID,
```

```
    const char *pSeq,
    const char *pAddress,
    time_t timestamp);
typedef int (*M2MXML_HANDLER_TURNONOFF)(
    void *pAppData,
    const char *pUUID,
    const char *pSeq,
    const char *pAddress,
    int sense,
    const char **ppProperties);
typedef int (*M2MXML_HANDLER_SETSTRINGOUTPUT)(
    void *pAppData,
    const char *pUUID,
    const char *pSeq,
    const char *pAddress,
    const char *pValue);
typedef int (*M2MXML_HANDLER_SETANALOGOUTPUT)(
    void *pAppData,
    const char *pUUID,
    const char *pSeq,
    const char *pAddress,
    double setPoint);
typedef int (*M2MXML_HANDLER_REBOOT)(
    void *pAppData,
    const char *pUUID,
    const char *pSeq);
typedef int (*M2MXML_HANDLER_QUERYCONFIGURATION)(
    void *pAppData,
    const char *pUUID,
    const char *pSeq,
    const char *pAddress,
    const char **ppProperties);
typedef int (*M2MXML_HANDLER_SETCONFIGURATION)(
    void *pAppData,
    const char *pUUID,
    const char *pSeq,
    const char *pAddress,
    const char **ppProperties);

typedef struct {
    void *pAppData;

    M2MXML_HANDLER_LOGERROR           pLogErrorHandler;
    M2MXML_HANDLER_UNKNOWN            pUnknownCommandHandler;
    M2MXML_HANDLER_REQUESTPERCEPT     pRequestPerceptHandler;
    M2MXML_HANDLER_TURNONOFF          pTurnOnOffHandler;
    M2MXML_HANDLER_SETSTRINGOUTPUT    pSetStringOutputHandler;
```

```
    M2MXML_HANDLER_SETANALOGOUTPUT    pSetAnalogOutputHandler;
    M2MXML_HANDLER_REBOOT             pRebootHandler;
    M2MXML_HANDLER_QUERYCONFIGURATION pQueryConfigurationHandler;
    M2MXML_HANDLER_SETCONFIGURATION   pSetConfigurationHandler;
} M2MXML_HANDLERS;
```

Each of the function pointers in **M2MXML_HANDLERS**, with the exception of **M2MXML_HANDLER_LOGERROR** and **M2MXML_HANDLER_UNKNOWN**, corresponds to one of the predefined M2MXML standard commands that could be sent by a remote server. As such, a description of their meaning and purpose is deferred to the M2MXML specification document. All the callback functions take **pAppData** as the first parameter. This allows the calling program to pass a reference to important context data through to the callback functions as an aid to implementation.
The special function pointer **M2MXML_HANDLER_LOGERROR** exists for the **M2MXML** module to call in the event of a parsing error. This function will pass an integer and a string which are intended to provide hints as to the nature of the error.
The special function pointer **M2MXML_HANDLER_UNKNOWN** exists as a "catch-all" for custom commands sent by a remote server that are extensions to the M2MXML specification.
All the functions that represent M2MXML commands also pass the **pUUID** of the enclosing message (or **NULL** if not provided), and the **pSeq** that must be used when responding to the command. Those commands that may be sent to a transducer will also have the **pAddress** parameter.

M2MXML_getPropertyString

```
const char *M2MXML_getPropertyString(
    const char **ppProperties,
    const char *pKey,
    const char *pDefault);
```

Since many parsed commands may receive a collection of "key/value" **ppProperties** as part of their definition, the **M2MXML_getPropertyString** function can be used to find a value given a **pKey**. If a match is found, the value will be returned by the function; otherwise, the value of **pDefault** will be returned.

M2MXML_parseMessage

```
void M2MXML_parseMessage(
    M2MXML_HANDLERS *pHandlers,
    const char *pMessageBuffer,
    int messageSize,
    int maxProperties,
    char *pParserBuffer,
    int parserSize);
```

In order to parse an incoming M2MXML message, the **M2MXML_parseMessage** function must be given the collection of **M2MXML_HANDLERS**, the **pMessageBufer** and **messageSize** of the incoming message. Finally, since it will use **M2MMEM** to allocate working storage for the parsing operation, it must be given the **maxProperties** that could be created in the process parsing, as well as a **pParserBuffer** and **parserSize** that it can give to **M2MMEM**.

Formatting

Formatting an outgoing M2MXML message involves the creation of the various M2MXML elements that are to be enclosed within the **<M2MXML></M2MXML>** message tags. Once created, the result may then be formatted into a single string, ready to be sent to the remote server.

M2MXML_createMessage

```
void *M2MXML_createMessage(
    const char *pUUID,
    char *pBuffer,
    int size);
```

This function begins the formatting process by taking a **pBuffer** and **size** that will be used to allocate the elements of the outgoing message. If **pUUID** may be **NULL** to indicate that no "td=" attribute is required in the enclosing **<M2MXML></M2MXML>** message tags. If this function is successful, it will return a **void *** pointer that should be passed to subsequent "M2MXML_create..." functions. If it returns **NULL**, then the operation failed due to insufficient buffer size.

M2MXML_createProperty

```
const char *M2MXML_createProperty(
    void *pMessageData,
    const char *pKey,
    const char *pValue);
```

This function will ensure that a **<Property/>** tag is added to the outgoing M2MXML message with **pKey** and **pValue** as the "key/value" pair. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

NOTE: Before calling this function, room for the property must have been created by calling **M2MXML_createPropertySet**.

M2MXML_createPropertySet

```
const char *M2MXML_createPropertySet(
    void *pMessageData,
    int propertyCount);
```

This function ensures that as many as **propertyCount** number of **<Property/>** tags can be added to an outgoing M2MXML message. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createCommand

```
const char *M2MXML_createCommand(
    void *pMessageData,
    const char *pSeq,
    const char *pAddressName,
    const char *pCommandName,
    const char **ppProperties,
    int requirePropertyValues,
    time_t timestamp);
```

This function will ensure that a **<Command/>** tag is added to the outgoing M2MXML message. If **requirePropertyValues** is set to **XML_FALSE**, then the **ppProperties** enclosed in the command will only contain the "name" attribute. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createException

```
const char *M2MXML_createException(
    void *pMessageData,
    int exceptionCode,
    const char *pOptionalMessage);
```

This function will ensure that a **<Exception/>** tag is added to the outgoing M2MXML message. Valid values for **exceptionCode** can be found in the predefined constants listed above. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createAnyPercept

```
const char *M2MXML_createAnyPercept(
    void *pMessageData,
    const char *pSeq,
    const char *pAddressName,
    const char *pValue,
    const char *pType,
    int entry,
    time_t timestamp);
```

This function will ensure that a **<Percept/>** tag of any type is added to the outgoing M2MXML message. Valid values for **pType** and **entry** are available as predefined constants listed above. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createAnalogPercept

```
const char *M2MXML_createAnalogPercept(
    void *pMessageData,
    const char *pSeq,
    const char *pAddressName,
    double value,
    int entry,
    time_t timestamp);
```

This function will ensure that a **<Percept/>** tag with the given analog **value** is added to the outgoing M2MXML message. Valid values **entry** are available as predefined constants listed above. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createDigitalPercept

```
const char *M2MXML_createDigitalPercept(
    void *pMessageData,
```

```
    const char *pSeq,
    const char *pAddressName,
    int value,
    int entry,
    time_t timestamp);
```

This function will ensure that a **<Percept perceptType="digital" />** tag with the given digital **value** is added to the outgoing M2MXML message. Valid values **entry** are available as predefined constants listed above. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createLocationPercept

```
const char *M2MXML_createLocationPercept(
    void *pMessageData,
    const char *pSeq,
    const char *pAddressName,
    double latValue,
    double longValue,
    int entry,
    time_t timestamp);
```

This function will ensure that a **<Percept perceptType="location" />** tag with the given location **latValue**/**longValue** is added to the outgoing M2MXML message. Valid values **entry** are available as predefined constants listed above. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createStringPercept

```
const char *M2MXML_createStringPercept(
    void *pMessageData,
    const char *pSeq,
    const char *pAddressName,
    const char *pValue,
    int entry,
    time_t timestamp);
```

This function will ensure that a **<Percept perceptType="string" />** tag with the given string **pValue** is added to the outgoing M2MXML message. Valid values **entry** are available as predefined constants listed above. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_createResponse

```
const char *M2MXML_createResponse(
    void *pMessageData,
    const char *pSeq,
    int resultCode,
    const char *pOptionalMessage,
    time_t timestamp);
```

This function will ensure that a **<Response />** tag is added to the outgoing M2MXML message. Valid values **resultCode** are available as predefined constants listed

above. This function returns **NULL** if successful; otherwise, it returns a **const char *** error message.

M2MXML_formatMessage

```
int M2MXML_formatMessage(
    void *pMessageData,
    char *pBuffer,
    int maxSize);
```

This function converts the collected element definitions in **pMessageData** into a string using **pBuffer** of **maxSize**. The function returns the size of the resulting message string, or a negative number if **pBuffer** is not large enough.

M2MXML_setTimezoneOffset

```
void M2MXML_setTimezoneOffset(
    int timezoneOffset);
```

This function allows the application to set a **timezoneOffset** defined in minutes that will be used to adjust timestamps when formatting them for an outbound message.

M2MBSM

The "Behavior State Machine" (or **M2MBSM**) sits on top of **M2MXML** and **M2MMEM**. It provides a complete implementation for the **M2MXML_HANDLERS** called for by **M2MXML** and provides a complete end-to-end encapsulation of the entire message parsing/formatting process, as well as maintaining a model of the logical M2MXML device so that both immediate, scheduled, or event-driven responses to a remote server can be performed by an application with a minimum of programming effort. The definition of **M2MBSM** can be found in "m2mbsm.h".

Logical Device Model

M2MBSM must be given by the application a data structure representing the structure of the device as known to a remote server. This will contain information such as the device's identity (**UUID**), that of each of its transducers (**address**), and other important characteristics (e.g., transducer type, device/transducer properties, etc.). In addition, the application must provide callback functions used to get/set transducer values and perform command operations. The application retains the ability to receive custom commands not handled directly by **M2MBSM**, but has an API that allows the application to tell **M2MBSM** how to perform both predefined and customer behaviors in response to a custom command.

M2MBSM_VALUE

```
typedef struct {
    int size;
    char *pData;
} M2MBSM_VALUE;
```

This type defines a buffer for storing a data value as a string called **pData** of a given **size**. The helper macro **M2MBSM_DEFINEVALUE(x)** can be used to declare an instance of this type in a static data structure.

M2MBSM_PROPERTY

```
typedef struct {
    const char *pKey;
    M2MBSM_VALUE value;
    int show;
} M2MBSM_PROPERTY;
```

This type defines a property by specifying its **pKey** and an **M2MBSM_VALUE** that can be used to store its **value**. In addition, the **show** property can be set to **XML_TRUE** or **XML_FALSE** to control the visibility of the property to the "queryConfiguration" command. It is important to ensure that all properties that may be used during the lifecycle of the device are declared to ensure that they will be found during routine **M2MBSM** processing.

M2MBSM_TRANSDUCER

```
typedef struct {
    void *pAppData;
    int type;
    int inout;
    const char *pAddress;
    const char *pLabel;
    const char *pUnits;
    M2MBSM_PROPERTY *pProperties;
    int propertyCount;
    M2MBSM_VALUE lastValue;
    M2MBSM_VALUE currentValue;
} M2MBSM_TRANSDUCER;
```

This type defines the important attributes of a transducer as it is known to a remote server. Valid values for **type** are defined by **M2MXML**. Valid values for **inout** are defined below. **pAddress** must be unique within the set of transducers defined for a device. **pLabel**, **pUnits**, and **pProperties** are optional and may be set to **NULL**. Note that **pProperties** is not the same as the **EXPAT** static array of pointers to strings. Also note that space must be allocated for both a **lastValue** and separate **currentValue** so that **M2MBSM** can properly manage threshold behaviors.

In/Out Direction Values
M2MBSM_DIRECTION_HIDDEN
M2MBSM_DIRECTION_IN
M2MBSM_DIRECTION_OUT
M2MBSM_DIRECTION_INOUT

M2MBSM_DEVICE

```
typedef struct {
    void *pAppData;
    const char *pUUID;
    M2MBSM_TRANSDUCER *pTransducers;
    int transducerCount;
```

```
    M2MBSM_PROPERTY *pProperties;
    int propertyCount;
} M2MBSM_DEVICE;
```

This type defines the logical device, its transducers, and properties. If a **pUUID** identity must be given to a remote server when sending it messages, it must be provided here; otherwise, it may be **NULL**. The **pAppData** pointer allows the application to include a reference to a context that will be passed to any callback functions during **M2MBSM** processing.

On-Device Behaviors

While most commands from a remote server generate an immediate response, some commands cause the **M2MBSM** to perform scheduled or condition-triggered activities that may result in unsolicited messages sent to the remote server by the application. These commands are said to control "on-device behaviors." Supporting these behaviors is a key purpose of **M2MBSM**.

Standard Behavior Properties

At this time, behaviors are set through device- and transducer-level properties. The following are the standard on-device behavior properties, and when appropriate, they allowed values:

Device Behavior Properties	**Comment**
M2MBSM_PROP_RPTALLABSRUN	
M2MBSM_PROP_RPTALLABSTIME	seconds
M2MBSM_PROP_RPTALLINTRUN	
M2MBSM_PROP_RPTALLINTSTART	seconds
M2MBSM_PROP_RPTALLINTTIME	seconds
M2MBSM_PROP_SETUPDEVICE	
General Transducer Behavior Properties	**Comments**
M2MBSM_PROP_RPTPTABSRUN	
M2MBSM_PROP_RPTPTABSTIME	seconds
M2MBSM_PROP_RPTPTINTRUN	
M2MBSM_PROP_RPTPTINTSTART	seconds
M2MBSM_PROP_RPTPTINTTIME	seconds
Analog Behavior Properties	**Comments**
M2MBSM_PROP_ANALOGCALIBRATION	
- **M2MBSM_ANALOGCALIBRATIONID_10VDC**	
- **M2MBSM_ANALOGCALIBRATIONID_20MA**	
- **M2MBSM_ANALOGCALIBRATIONID_ICTD**	
M2MBSM_PROP_RPTLOW	

M2MBSM_PROP_RPTLOWLOW	
M2MBSM_PROP_RPTHIGH	
M2MBSM_PROP_RPTHIGHHIGH	
Digital Input Behavior Properties	**Comments**
M2MBSM_PROP_RPTLEVELCHANGE	
- **M2MBSM_LEVELCHANGE_OFF**	
- **M2MBSM_LEVELCHANGE_LOW2HIGH**	
- **M2MBSM_LEVELCHANGE_HIGH2LOW**	
Digital Output Behavior Properties	**Comments**
M2MBSM_PROP_PULSELOWRUN	
M2MBSM_PROP_PULSELOWTIME	milliseconds
M2MBSM_PROP_PULSEHIGHRUN	
M2MBSM_PROP_PULSEHIGHTIME	milliseconds
M2MBSM_PROP_TIMERLOWABSRUN	
M2MBSM_PROP_TIMERLOWABSTIME	milliseconds
M2MBSM_PROP_TIMERHIGHABSRUN	
M2MBSM_PROP_TIMERHIGHABSTIME	milliseconds
M2MBSM_PROP_TIMERLOWINTRUN	
M2MBSM_PROP_TIMERLOWINTTIME	milliseconds
M2MBSM_PROP_TIMERHIGHINTRUN	
M2MBSM_PROP_TIMERHIGHINTTIME	milliseconds

M2MBSM_BEHAVIOR

```
typedef struct _M2MBSM_BEHAVIOR {
    void *pAppData;
    M2MBSM_TRANSDUCER *pTransducer;
    int (*pHandler)(M2MBSM_DEVICE *,struct _M2MBSM_BEHAVIOR
*,int,time_t);
    time_t nextFiring;
    time_t firingPeriod;
    struct _M2MBSM_BEHAVIOR *pNext;
} M2MBSM_BEHAVIOR;
```

This type defines the state of a single behavior to be executed. The **pAppData** pointer allows a custom behavior to maintain a reference to whatever data it needs to implement its function. This data will be passed to the **pHandler**, but it is important that only **pAppData**, **nextFiring**, and **firingPeriod** be modified by the function.

M2MBSM_HANDLER_BEHAVIOR

```
typedef int (*M2MBSM_HANDLER_BEHAVIOR)(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_BEHAVIOR *pBehavior,
    int event,
    time_t timestamp);
```

This function pointer prototype defines how behavior callback functions must be defined. The function is passed the target **pDevice**, the **pBehavior** state data in question from which it can obtain the **pTransducer**, if any, that it applies to, the **event** that is firing, and the original **timestamp** when the event occurred (this allows for the current time to be slightly different if necessary).

The following values are valid for the **event** parameter:

Behavior Event IDs
M2MBSM_BEHAVIOREVENTID_MATCH
M2MBSM_BEHAVIOREVENTID_CONFIGURE
M2MBSM_BEHAVIOREVENTID_EXECUTE

For a behavior to be established for ongoing processing, **M2MBSM** goes through a process of finding a match for an in-bound property. To do this, the behavior callback function is called with the **event** of **M2MBSM_BEHAVIOREVENTID_MATCH**. When this occurs, the **pBehavior** argument is actually a **const char *** string to the property "key". If this "key" matches a property needed by the behavior, the function return **XML_TRUE**.

Once a behavior has been matched, **M2MBSM** calls the behavior function again with the **event** of **M2MBSM_BEHAVIOREVENTID_CONFIGURE**. The function must then determine if it has all the elements necessary for establishing the behavior. If so, it returns **XML_TRUE** and the behavior will then be established. Otherwise, if the function returns **XML_FALSE**, the behavior will not be established or will be disabled if it was previously established.

Finally, as determined by the **nextFiring** property of the **M2MBSM_BEHAVIOR** definition, the callback function will be called again with the **event** of **M2MBSM_BEHAVIOREVENTID_EXECUTE**. **M2MBSM** will continue to call the behavior function every time its **nextFiring** time occurs or until the function returns **XML_FALSE**, at which time the behavior will be disabled. This allows for behaviors to terminate themselves after some discrete number of firings, if necessary. However, all standard M2MXML behaviors remain in force until their properties are changed so as to turn them off.

The following are the predefined behavior handlers that **M2MBSM** uses:

Standard Behavior Callback Functions
M2MBSM_reportAllAbsoluteBehaviorHandler
M2MBSM_reportPointAbsoluteBehaviorHandler
M2MBSM_reportAllIntervalBehaviorHandler
M2MBSM_reportPointIntervalBehaviorHandler
M2MBSM_reportDigitalTransitionBehaviorHandler

M2MBSM_reportOutOfBoundsBehaviorHandler
M2MBSM_setupDeviceBehaviorHandler
M2MBSM_timerOffAbsoluteBehaviorHandler
M2MBSM_timerOnAbsoluteBehaviorHandler
M2MBSM_timerOffIntervalBehaviorHandler
M2MBSM_timerOnIntervalBehaviorHandler

Application Processing

Once an application has a logical device model defined and with the backdrop of behaviors, the final step the processing model. **M2MBSM** provides an API that allows the application to have full control over the memory and processing model. It defers to the application to determine when behaviors are processed, as well as when and how messages are accepted from and sent to a remote server. To do this, the application simply initializes the **M2MBSM** state, providing callback functions for accessing physical resources based on the logical model. Next, the application performs its own control flow to read/write from communications sources and permit the **M2MBSM** to process any pending behaviors.

M2MBSM_HANDLERS

```
typedef int (*M2MBSM_HANDLER_REQUESTPERCEPT)(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_TRANSDUCER *pTransducer,
    time_t *pTimestamp);
typedef int (*M2MBSM_HANDLER_TURNONOFF)(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_TRANSDUCER *pTransducer,
    int sense,
    const char **ppProperties);
typedef int (*M2MBSM_HANDLER_SETSTRINGOUTPUT)(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_TRANSDUCER *pTransducer,
    const char *pValue);
typedef int (*M2MBSM_HANDLER_SETANALOGOUTPUT)(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_TRANSDUCER *pTransducer,
    double setPoint);
typedef int (*M2MBSM_HANDLER_REBOOT)(
    M2MBSM_DEVICE *pDevice);
typedef int (*M2MBSM_HANDLER_QUERYCONFIGURATION)(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_TRANSDUCER *pTransducer,
    M2MBSM_PROPERTY *pProperty);
typedef int (*M2MBSM_HANDLER_SETCONFIGURATION)(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_TRANSDUCER *pTransducer,
    M2MBSM_PROPERTY *pProperty);
```

```
typedef void (*M2MBSM_HANDLER_SENDMESSAGE)(
    M2MBSM_DEVICE * pDevice,
    void *pData);

typedef struct {
    void *pAppData;

    M2MXML_HANDLER_LOGERROR           pLogErrorHandler;
    M2MXML_HANDLER_UNKNOWN            pUnknownCommandHandler;

    M2MBSM_HANDLER_REQUESTPERCEPT     pRequestPerceptHandler;
    M2MBSM_HANDLER_TURNONOFF          pTurnOnOffHandler;
    M2MBSM_HANDLER_SETSTRINGOUTPUT    pSetStringOutputHandler;
    M2MBSM_HANDLER_SETANALOGOUTPUT    pSetAnalogOutputHandler;
    M2MBSM_HANDLER_REBOOT             pRebootHandler;
    M2MBSM_HANDLER_QUERYCONFIGURATION pQueryConfigurationHandler;
    M2MBSM_HANDLER_SETCONFIGURATION   pSetConfigurationHandler;

    M2MBSM_HANDLER_SENDMESSAGE        pSendMessageHandler;
} M2MBSM_HANDLERS;
```

This type defines the set of callback functions that **M2MBSM** requires of the application to provide full functionality.
The **M2MXML_HANDLER_LOGERROR** and **M2MXML_HANDLER_UNKNOWN** callbacks are simply passed through to **M2MXML**. If the **M2MXML_HANDLER_UNKNOWN** callback is not provided, **M2MBSM** will respond to any non-standard M2MXML command with the "Unknown Command" response code. If any other command-oriented callbacks are not provided, those commands depending on them will respond with the "Failed Execution" response code.

Key to most of the **M2MBSM** processing are the **M2MBSM_HANDLER_REQUESTPERCEPT** and **M2MBSM_HANDLER_SENDMESSAGE** callback functions. The **M2MBSM_HANDLER_REQUESTPERCEPT** function ensures that the values maintained in the **M2MBSM_TRANSDUCER** reflect the actual values of the physical device. Then, the ** function gives **M2MBSM** a destination for the outbound M2MXML messages that are generated as a consequence of message and behavior processing.

M2MBSM_configureBehavior

```
int M2MBSM_configureBehavior(
    M2MBSM_TRANSDUCER *pTransducer,
    M2MBSM_PROPERTY *pProperty,
    M2MBSM_HANDLER_BEHAVIOR pBehaviorHandler);
```

This function asks **M2MBSM** to consider establishing the given **pBehaviorHandler** for an **M2MBSM_BEHAVIOR** for the specified **pTransducer**. If **pProperty** is not **NULL**, the **pBehaviorHandler** will first be asked if it matches (see description of **events** above). The function will return **XML_TRUE** if the behavior is established, **XML_FALSE** otherwise.

M2MBSM_countBehaviors

```
int M2MBSM_countBehaviors(void);
```

This function returns the number of **M2MBSM_BEHAVIOR** instances that are currently pending. This function can be helpful when deciding to execute the **M2MBSM_processState** function a sufficient number of times in a loop so that all behaviors are guaranteed to have fired at least once.

M2MBSM_findProperty

```
M2MBSM_PROPERTY *M2MBSM_findProperty(
    M2MBSM_PROPERTY *pProperties,
    int propertyCount,
    const char *pKey);
```

This function helps the application find an **M2MBSM_PROPERTY** within the given collection **pProperties** of size **propertyCount**. The function will return the **M2MBSM_PROPERTY** that matches the given **pKey** or **NULL** if not found.

M2MBSM_initializeState

```
M2MBSM_HANDLERS *M2MBSM_initializeState(
    M2MBSM_DEVICE *pDevice,
    M2MBSM_BEHAVIOR *pBehaviorBuffer,
    int maxBehaviors,
    int maxProperties,
    char *pPropertyBuffer,
    int propertySize,
    char *pParserBuffer,
    int parserSize,
    char *pFormatBuffer,
    int formatSize);
```

This function initializes the **M2MBSM** state for the application and should only be called once before any other **M2MBSM** functions are called. This is when the application provides the **pDevice** logical model that is used throughout subsequent message/behavior processing. In addition, the application must provide a set of fixed-size buffers that **M2MBSM** (or indirectly, **M2MXML**) will use as working storage for its function.

M2MBSM_nextStartTime

```
time_t M2MBSM_nextStartTime(
    int hhmmss,
    int mustBeFuture);
```

This function creates a time values offset from "now" by **hhmmss**. If **mustBeFuture** is true and the calculated time has already occurred today, the value is shifted by 24 hours.

M2MBSM_parseMessage

```
void M2MBSM_parseMessage(
    const char *pMessageBuffer,
```

```
    int messageSize);
```

This function ensures that the incoming message in **pMessageBuffer** of size **messageSize** is parsed and processed by **M2MBSM**, resulting in the appropriate immediate messages and established behaviors.

M2MBSM_processState

```
time_t M2MBSM_processState(void);
```

This function finds the next behavior that is ready to be fired, if any, and calls its function with **M2MBSM_BEHAVIOREVENTID_EXECUTE**. The function returns the next time when a behavior should be fired. If this result is used to optimize the application's control logic, the programmer should be careful to consider the potential for incoming messages to change this "next firing time."

APPENDIX I: testmain.c

Included with **M2MXMLC** is a test application called "testmain.c" that accepts M2MXML commands from **stdin** or a filename provided as a command-line argument and writes any resulting messages to **stdout**. In addition, it accepts some simple non-XML inputs that help control behavior processing. The file "unittestin.txt" represents a full set of incoming messages that both require immediate response and establish various on-device behaviors. When provided as input the "testmain" the output should exactly match the contents of "unittestout.txt". As changes are made to **M2MXMLC**, these testing tools will be maintained in order to ensure the correctness of the library.