kbmMW XML transport format

for kbmMW v. 2.00+

kbmMW have multiple types of transport messages: Request, Response, ServiceCall, Message as the base types, and under the Message type there are several variants: Subscription, Unsubscription, Throttle, Unsolicited, Cache.

This document will concentrate on describing the XML transport format for the message types Request, Response and unsolicited message. The base XML syntax will be valid for the other types too, but they may contain more or less data values.

Each message regardless of type is build with an envelope encapsulating a header and a body part.

XML datatypes

As with all kbmMW transport stream formats, several datatypes are natively supported.

Null values

Null values are not written in the XML stream. Thus values not in the XML document should be percieved NULL or empty.

String values

String values which are of zero length will be written as an endtag like this:

```
<VALUE name='aname' datatype='256' />
```

String values which contains characters outside the ASCII range 32 to 127 will be Base64 encoded. Unless the string is Base64 encoded, the following characters will be converted (HTTP encoded):

& to **&**
' to **'**
“ to **"**
< to **<**
> to **>**

Strings which are encoded with Base64 will have an additional attribute: `encoding='base64'`
Strings which are HTTP encoded, will have an additional attribute: `encoding='http'`

Strings which are less than 71 characters long after any optional encoding will be included as an attribute in an end tag:

```
<VALUE name='aname' datatype='256' value='stringvalue' />
```

Strings which are 71 characters or longer after any optional encoding will be added as a symbol:

```
<VALUE name='aname' datatype='256'>
long string value which exceeds 71 chars. xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
</VALUE>
```

Example of HTTP encoded string less than 71 characters long:

```
<VALUE name='aname' datatype='256' encoding='http' value='It&apos;s cold'/>
```

AName is the logical name of the string.

Datatype **256** represent the value of the constant **varString**.

Unicode string values (WideString)

It follows the same scheme as the standard string, except that the datatype is 8 (varOleStr), and that the string is UTF8 encoded before writing it either raw, http encoded or Base64 encoded depending on the nature of the encoded string.

Eg.

```
<VALUE name='aname' datatype='8' value='utf8encodedstring'/>
```

Datatype **8** represents the value of **varOleStr**.

Integer values

Integer values are written like this:

```
<VALUE name='aname' datatype='3' value='integervalue' />
```

integervalue can be positive and negative values in the range -2147483648..2147483647

Eg.

```
<VALUE name='aname' datatype='3' value='4556' />
```

Datatype **3** represents the value of **varInteger**.

Float values

Float values are stored like this:

```
<VALUE name='aname' datatype='5' value='floatvalue' />
```

floatvalue is the string representation of a floating point value. The exact format is determined by the **FormatSettings** property of the transport on LEVEL7 compilers. On pre LEVEL7 compilers the format follows the default format settings for the **FloatToStr** function.

Eg. (provided the decimal point is .

```
<VALUE name='aname' datatype='5' value='1.56' />
```

Datatype 5 represents the value of varDouble.

Date/Time values

Date/Time values are stored like this:

```
<VALUE name='aname' datatype='7' value='datetimevalue' />
```

datetimevalue is the string representation of a floating point value. The exact format is determined by the **FormatSettings** property of the transport on LEVEL7 compilers. On pre LEVEL7 compilers the format follows the default format settings for the **DateTimeToStr** function.

Eg.

```
<VALUE name='aname' datatype='7' value='13-01-04 12:55:11' />
```

Datatype 7 represents the value of varDate.

Stream data

Stream data is stored like this:

```
<VALUE name='aname' datatype='STREAM' size='asize' encoding='base64'>
base64 encoded streamdata
</VALUE>
```

asize is the size of the stream. However although kbmMW represents the size value correctly kbmMW do not trust it while reading stream data, and will automatically recalculate the size while reading and decoding the Base64 coded data.

Variant

A variant is simply stored as one of the base types. Eg. short, longword etc as integer.

Array of variants

Supports one dimensional arrays which can be nested to any depth.

```
<VALUE name='aname' datatype='adatatype' low='alowbounds' high='ahighbounds' >
  <VALUE .... variant value.... >
...
</VALUE>
```

adatatype is the ordinal type of the array declaration, typically '12' (**varVariant**).
alowbounds is the low bounds of the array, typically '0'.
ahighbounds is the high bounds of the array, typically size of array-1.

Eg. array [0..2] of integer

```
<VALUE name='aname' datatype='3' low='0' high='2' >
  <VALUE name='aname' datatype='3' value='1' />
  <VALUE name='aname' datatype='3' value='2' />
  <VALUE name='aname' datatype='3' value='3' />
</VALUE>
```

Variant byte array

A variant byte array is stored as Base64 encoded data.

```
<VALUE name='aname' datatype='adatatype' low='alowbounds' high='ahighbounds'
encoding='base64'>
... Base64 encoded data ...
</VALUE>
```

adatatype is the ordinal type of the array declaration '17' (**VarByte**).
alowbounds is the low bounds of the array, typically '0'.
ahighbounds is the high bounds of the array, typically size of non encoded data-1.

Object

Objects are handled via the reader and writer handlers registered for that particular type of object.

```
<VALUE name='aname' datatype='OBJECT' classname='aclassname'>
  <VALUE... object defined storage....>
</VALUE>
```

aclassname is the name of the class of the object. Its directly used on read to create the correct type of object. The object class needs to be registered in the receiving end for it to be able to read it.

Eg. if an object have defined a writer (and a compatible reader) which stores an integer and a long string:

```
<VALUE name='aname' datatype='OBJECT' classname='TMyObject'>
  <VALUE name='IntegerProperty' datatype='3' value='1' />
  <VALUE name='StringProperty' datatype='256'>
long string value which exceeds 71 chars. xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  </VALUE>
</VALUE>
```

For sakes of simplicity, its generally considered good practice to keep the same order of the object property values in the XML as in the object reader/writer.

The Request message

The request message is used by clients to request a specific service on the application server, and expect a resulting response back from it.

The syntax for a request message:

```
<KBMMW_REQUEST xmlns:kbmmw='http://www.components4developers.com/2003/kbmMW/2.00/XML/1.00'>

  <kbmmw:Header>

    <SERVICE name='aservicename' [version='aserviceversion'] [stateid='astateid'] />

   [<REQUESTER [token='atoken'] | ([username='ausername'] [password='apassword'])
               [requestid='arequestid'] [location='alocation']) />]

   {<VALUE name='DATA' datatype='adatatype' value='avalue' />]

   [<ATTRIBUTE name='ATTR1' datatype='256' value='avalue' />]
   ...
   [<ATTRIBUTE name='ATTRn' datatype='256' value='avalue' />]

  </kbmmw:Header>

  <kbmmw:Body>
```

```
  (<FUNC name='afunctionname' />)
 |
  (<FUNC name='afunctionname' [ ArgCount='aargcount' ] >
    <VALUE name='ARGn' datatype='adatatype' value='avalue'>
   </FUNC>)

  [<VALUE name='STREAM' datatype='STREAM' size='asize' encoding='base64'>
    ...base64 encoded stream...
   </VALUE>]

  </kbmmw:Body>

</KBMMW_REQUEST>
```

aservicename is the name of the requested service (required).
aserviceversion is the version of the requested service. If no service version is given (empty) the version attribute is optional.
astateid is the ID of the stateful service instance. If a new stateful instance is requested or its not a stateful service, the attribute is optional.
atoken is the token which has earlier been handed to the client from the app server identifying the client. If no token has been given, this attribute is optional.
ausername and *apassword* is the username and password values. The username and password attributes should not be specified if a token has been given. Both attributes are optional.
arequestid is the request id of the request. Its an optional, but usually given, client unique counter. Its of importance to asynchronous clients which will need to know what response to wait for since the responce message will contain the client specified request id.
alocation is an optional string identifying the client location. It can be an ip address or other textual information the client is willing to give.
ATTR1...*ATTRn*: these are optional attribute values which reflects the clients Attribute list. Its not required by kbmMW, but the client and server may use it for transporting additional information between them outside the standard format of the message. The format follows the standard string datatype format, and thus can be multiline.
afunctionname is the name of a function to be called on the requested service/service version.
aargcount is the number of arguments provided for the function call. If no arguments is given the ArgCount attribute is optional.
ARGn is ARG1...ARGn and is the generated name of the arguments given for the function call. The *adatatype* and *avalue* all follows the normal layout as for variant values and thus can be multiline.
asize is the size of the optional STREAM element. If no stream is needed the complete element doesnt need to be given. It follows the normal layout of a stream value.

Notice that the DATA and ARG tags follow the layout for the variant datatype, and thus their exact look may be different from the example (multiline optionally with encoding etc.)

The Response message

The response message is returned to clients after they have made a request.

The syntax for a response message:

```
<KBMMW_RESPONSE xmlns:kbmmw='http://www.components4developers.com/2003/kbmMW/2.00/XML/1.00'>

  <kbmmw:Header>

    <STATUS code='astatuscode' message='amessage' icode='aicode' stateid='astateid' />

   [<REQUESTER [requestid='arequestid'] [token='atoken'] />}

   {<VALUE name='DATA' datatype='adatatype' value='avalue' />]

   [<ATTRIBUTE name='ATTR1' datatype='256' value='avalue' />]
   ...
   [<ATTRIBUTE name='ATTRn' datatype='256' value='avalue' />]

  </kbmmw:Header>

  <kbmmw:Body>

  <VALUE name='RESULT' datatype='adatatype' value='avalue' />

  [<VALUE name='STREAM' datatype='STREAM' size='asize' encoding='base64'>
   ...base64 encoded stream...
   </VALUE>]

  </kbmmw:Body>

</KBMMW_RESPONSE>
```

astatuscode is the status value. 0 for ok, <0 for error and >0 for warning. (required).
amessage is the status message. (required).
aicode is an internal code which in the future for example can request the client to authenticate itself etc. Its required but not in use and an empty value or 0 should be used today.
astateid is the state id returned from the server for the given request. If the request didnt result in holding a state, -1 will be returned.
atoken is the token handed to the client as the result from authenticating and authorizing the client's username and password. If no token is to be returned, this attribute is optional.
arequestid is the request id of the original client request. Its an optional, but usually given, client unique counter. Its of importance to asynchronous clients which will need to know what response to wait for since the responce message will contain the client specified request id.
ATTR1...ATTRn: these are optional attribute values which reflects the clients Attribute list. Its not required by kbmMW, but the client and server may use it for transporting additional information between them outside the standard format of the message. The format follows the standard string datatype format, and thus can be multiline.
asize is the size of the optional STREAM element. If no stream is needed the complete element doesnt need to be given. It follows the normal layout of a stream value.

adatatype and *avalue* all follows the normal layout as for variant values and thus the resulting value can be multiline.

Notice that the DATA and RESULT tags follow the layout for the variant datatype, and thus their exact look may be different from the example (multiline optionally with encoding etc.)

The unsolicited Message message

The response message is returned to clients after they have made a request.

The syntax for a response message:

```
<KBMMW_MESSAGE xmlns:kbmmw='http://www.components4developers.com/2003/kbmMW/2.00/XML/1.00'>

  <kbmmw:Header>


   [<SENDER [token='atoken'] | ([username='ausername'] [password='apassword'])
            [location='alocation']) />]

   {<VALUE name='DATA' datatype='adatatype' value='avalue' />]

   [<ATTRIBUTE name='ATTR1' datatype='256' value='avalue' />]
   ...
   [<ATTRIBUTE name='ATTRn' datatype='256' value='avalue' />]

  </kbmmw:Header>

  <kbmmw:Body>

  (<MESSAGE subject='asubject' />)
 |
  (<MESSAGE subject='asubject' [ ArgCount='aargcount' ] >
     <VALUE name='ARGn' datatype='adatatype' value='avalue'>
   </MESSAGE>)

  [<VALUE name='STREAM' datatype='STREAM' size='asize' encoding='base64'>
    ...base64 encoded stream...
    </VALUE>]

  </kbmmw:Body>

</KBMMW_MESSAGE>
```

atoken is the token which has earlier been handed to the client from the app server identifying the client. If no token has been given, this attribute is optional.
ausername and *apassword* is the username and password values. The username and password attributes should not be specified if a token has been given. Both attributes are optional.
alocation is an optional string identifying the client location. It can be an ip address or other textual information the client is willing to give.
ATTR1...ATTRn: these are optional attribute values which reflects the clients Attribute list. Its not required by kbmMW, but the client and server may use it for transporting additional information

between them outside the standard format of the message. The format follows the standard string datatype format, and thus can be multiline.
asubject is the complete subject hierarchy as a string. Typically starting with 'MSG:'. Please notice that in the asynchronous messaging transports, the subject is also send as a preheader information before the XML package of performance and message filtering reasons. Thus in that case the XML data should be considered the message user payload, rather than the complete message.
aargcount is the number of arguments provided for the function call. If no arguments is given the ArgCount attribute is optional.
ARGn is ARG1...ARGn and is the generated name of the arguments given for the function call. The *adatatype* and *avalue* all follows the normal layout as for variant values and thus can be multiline.
asize is the size of the optional STREAM element. If no stream is needed the complete element doesnt need to be given. It follows the normal layout of a stream value.

Notice that the DATA and ARG tags follow the layout for the variant datatype, and thus their exact look may be different from the example (multiline optionally with encoding etc.)

This concludes the whitepaper about the kbmMW XML transport format.

Kim Madsen
Components4Developers