Using IBM WebSphere MQ message property APIs in ILE RPG

Kevin Adler February 10, 2014

Handling Java™ Message Service (JMS) message properties from native IBM® WebSphere® MQ applications is now much easier with the addition of the `MQINQMP` and `MQSETMP` application programming interfaces (APIs) in WebSphere MQ. This article highlights how these new APIs can be used within Integrated Language Environment (ILE) RPG programs.

The old way and the way forward

Traditionally, to communicate with a JMS application (say, a web service running in IBM WebSphere Application Server), one would need to use an RFH2 header. This is cumbersome, especially in RPG. The RFH2 headers consist of a set of fixed header fields followed by an arbitrary number of data structures, each containing a length field and a buffer containing XML data that defines the properties. Generating an RFH2 header is not too hard. Just include the copy file, cmqrfh2g, in to your data structure and define as many lengths and data pairs as needed for the number of folders you need to set (properties in different folders must be sent in different pairs). For more freedom and to eliminate sending extra blanks from the fixed length fields, you can also build a buffer dynamically. The real trouble comes in trying to read the properties, which requires parsing an RFH2 header sent by another application. Although RPG is perfectly capable of parsing varying length headers with an arbitrary number of varying length fields, it is much better suited to fixed-length, record-like data.

The other issue with using RFH2 headers in RPG (really any ILE language) is that the XML data must be in Unicode. RPG has the benefit here of supporting UTF-16 natively. Unless you're only dealing with other RPG applications, though, most likely any RFH2 headers you get from other applications will be in UTF-8, since that is the default for Linux, Windows, and Java. This means that your are pretty much guaranteed to have to deal with iconv, which is always a pain.

With the release of WebSphere MQ 7.0, MQ gained new API calls that greatly decrease the amount of work needed to interact with message properties and remove the need to generate or parse RFH2 headers. These are:

- `MQSETMP` – Set a message property
- `MQDLTMP` – Delete a message property

- `MQINQMP` – Inquire about a message property

Along with these APIs, other APIs were added to aid in the use of the message property APIs:

- `MQCRTMH` – Create a message handle
- `MQDLTMH` – Delete a message handle
- `MQBUFMH` – Convert a message buffer into a message handle
- `MQMHBUF` – Convert a message handle into a message buffer

The key to the new message property APIs is a message handle. A message handle (`MQHMSG`) is a 64-bit identifier and is used to allow an application to refer to the properties of a message, similar to how an `MQHCONN` is used to refer to a connection to a queue manager or an `MQHOBJ` is used to refer to a topic or queue. The first step in using these new message property APIs is to create a message handle.

```
dcl-s Hmsg int(20);
dcl-ds MQCMHO Qualified;
    /copy CMQCMHOG
end-ds;

MQCRTMH(HConn : MQCMHO : HMsg : CompletionCode : Reason);
```

You declare two variables, `HMsg` as a 64-bit integer and an `MQCMHO` (Create Message Handle Options) structure. Then, you call `MQCRTMH` to generate our message handle. Assuming that `CompletionCode` is `0`, you now have a message handle with which you can use the new message property APIs.

Using MQSETMP

To see how to use `MQSETMP` to set a message property, let's first look at the parameters of `MQSETMP`:

Example 1: MQSETMP procedure definition from CMQG

```
DMQSETMP          PR                  EXTPROC('MQSETMP')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Message handle
D HMSG                          20I 0 VALUE
D* Options that control the action of MQSETMP (MQSMPO)
D SETOPT                        20A
D* Property name (MQCHARV)
D PRNAME                        32A
D* Property descriptor (MQPD)
D PRPDSC                        24A
D* Property data type
D TYPE                          10I 0 VALUE
D* Length of the Value area
D VALLEN                        10I 0 VALUE
D* Property value
D VALUE                           *   VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CompCode
D REASON                        10I 0
```

For the purposes of this exercise, using the default values for `MQSMPO` and `MQPD` are fine. For more information on these parameters, refer to the WebSphere MQ Information Center.

Let's define some variables to use `MQSETMP`:

```
dcl-ds PropName Qualified;
    /copy QMCHRVG
end-ds;

dcl-s PropertyName varchar(40);
dcl-s PropertyValue varchar(40);

dcl-ds MQPD Qualified;
    /copy CMQPDG
end-ds;

dcl-ds MQSMPO Qualified;
    /copy CMQSMPOG
end-ds;
```

In this example, we will define the usr.format property to inform the receiving application that the message is XML data. First, set up the property name.

```
PropertyName = 'format';
PropertyValue = 'xml';
PropName.VCHRP = %ADDR(PropertyName : *DATA);
PropName.VCHRL = %LEN(PropertyName);
```

Note: If you do not qualify a property name, it will default to the 'usr' folder.

Then, you just need to call `MQSETMP`, specifying all the parameters:

```
MQSETMP(HConn : HMsg : MQSMPO : PropName : MQPD : TYPSTR :
        %LEN(PropertyValue) : %ADDR(PropertyValue : *DATA) :
        CompletionCode : Reason);
```

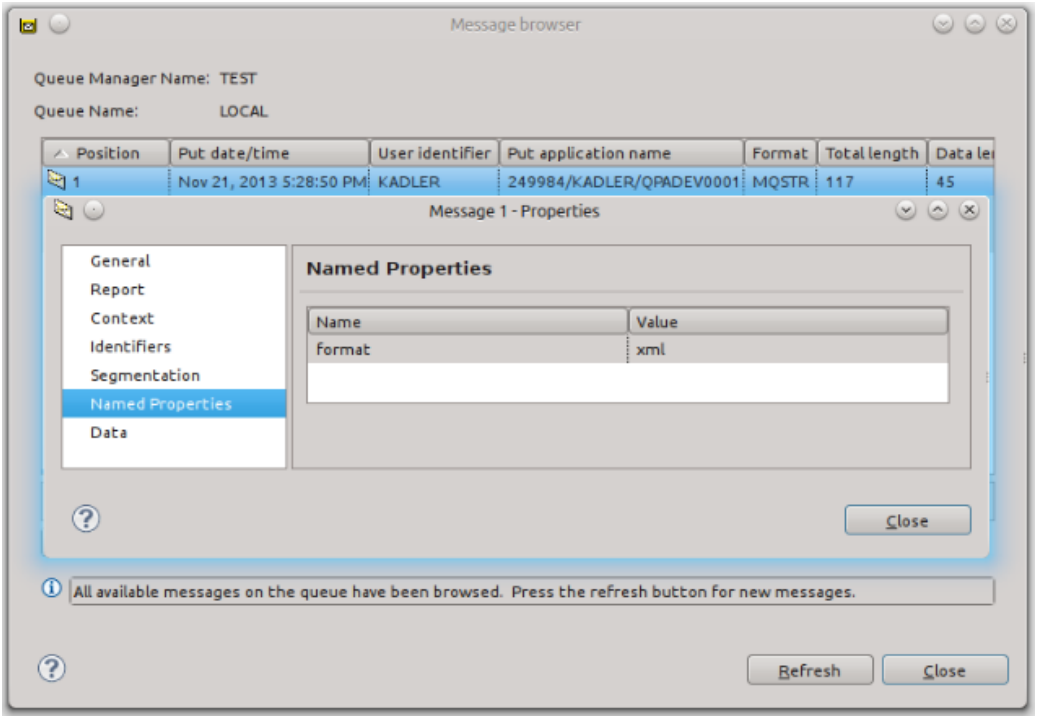
At this point, you've created a message handle and set the `usr.format` property to `xml`, but what message has received this property? The answer is that you haven't actually set this property on an actual message yet. All you've done is allocated a message handle that refers to an area of storage within WebSphere MQ which holds the message properties. When you put a message, you can tell WebSphere MQ to add those properties to the message you are putting. The only thing you must do for this to happen is to set the message handle field to the message handle you got from `MQCRTMH` and tell WebSphere MQ that you are using a version 3 `MQPMO` (so that WebSphere MQ knows that the message handle field is defined).

```
MQPMO.PMVER = PMVER3;
MQPMO.PMOMH = Hmsg;
```

Now, just do an `MQPUT` like normal and the message properties will be added to the message. Under the covers, WebSphere MQ will read the properties from the message handle and generate the RFH2 header for you. You can easily verify that everything worked by using IBM WebSphere MQ Explorer or the `WRKMQMQ` system command. MQ Explorer is an Eclipse-based graphical tool that enables you to explore and configure all WebSphere MQ objects and resources from your Microsoft® Windows® or Linux® PC. It is included with the MQ server installation and is also available separately in the MS0T SupportPac. To view in MQ Explorer, expand the queue manager

and click **Queues**. You can then right-click the queue and click **Browse Messages**. Find the message in the list to inspect and right-click it and then click **Properties**. When you select **Named Properties** in the left pane you can see the properties as shown in Figure 1.

Figure 1: MQ Explorer



Using MQINQMP

Like `MQSETMP`, which makes it easier to set message properties, there is also `MQINQMP` to retrieve the value of a specific message property or even use wildcards to fetch the value of multiple properties. A prototype is shown in Example 2.

Example 2: MQINQMP procedure definition from CMQG

```
DMQINQMP          PR                  EXTPROC('MQINQMP')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Message handle
D HMSG                          20I 0 VALUE
D* Options that control the action of MQINQMP (MQIMPO)
D INQOPT                        72A
D* Property name (MQCHARV)
D PRNAME                        32A
D* Property descriptor (MQPD)
D PRPDSC                        24A
D* Property data type
D TYPE                          10I 0
D* Length in bytes of the Value area
D VALLEN                        10I 0 VALUE
D* Property value
D VALUE                           *   VALUE
D* Length of the property value
D DATLEN                        10I 0
D* Completion code
```

```
D CMPCOD                        10I 0
D* Reason code qualifying CompCode
D REASON                        10I 0
```

The only real difference from `MQSETMP` is that you use an `MQIMPO` instead of `MQSMPO`.

To use `MQINQMP`, you first create a message handle like above and pass it in the `MQGMO`. The only other trick you need is to tell WebSphere MQ that you want the message properties returned in the message handle and not in an RFH2 header.

```
MQGMO.GMVER = GMVER4;
MQGMO.GMMH = Hmsg;

MQGMO.GMOPT += GMPRIH; // return message properties in handle

MQIMPO.IPOPT = IPINQN + // iterate over properties
               IPCTYP + // convert type if necessary
               IPCVAL;  // convert value into native CCSID

PropQuery = 'usr.%';
PropName.VCHRP = %ADDR(PropQuery : *DATA);
PropName.VCHRL = %LEN(PropQuery);

MQIMPO.IPRETNAMCHRP = %ADDR(PropertyName : *DATA);
MQIMPO.IPRETNAMVSBS = PropNameMax;

MQINQMP(HConn : HMsg : MQIMPO : PropName : MQPD : PropertyType :
        PropValueMax : %ADDR(PropertyValue : *DATA) :
        ActPropLength : CompletionCode : Reason);

%LEN(PropertyValue) = ActPropLength;
%LEN(PropertyName) = MQIMPO.IPRETNAMCHRL;
```

Here, you use the query of `usr.%` to retrieve all the properties in the `usr` folder. After the call to `MQINQMP`, `PropertyName` contains the name of the first user property found and `PropertyValue` contains its value. After you have processed this property, you can call `MQINQMP` again to retrieve the next property that matched the query. Eventually, there will be no more properties and WebSphere MQ will set a completion code of 1 and a reason code 2471 (MQRC_PROPERTY_NOT_AVAILABLE).

Conclusion

With the advent of the new message property functions in WebSphere MQ 7.0, it is now much easier to deal with message properties within ILE applications. No longer does an application need to handle parsing RFH2 headers, deal with XML, or deal with character conversion using iconv to get or set message properties. This article has shown how to make use of these new APIs in your applications. Full examples can be found in the AMQ3IQM4 and AMQ3STM4 samples included with WebSphere MQ 7.1. These samples are also included in WebSphere MQ 7.0.1.6 and later fix packs.

Resources

- Find more information about Message Properties, MQSETMP, and MQINQMP in the WebSphere MQ Information Center.

- SupportPac MS0T download
- IBM i developerWorks forum