PCF Processing in JMS - a "gotcha" and the solution

MarkBluemel |Oct 17 2016 Updated

Another in the series of bitesize blog posts about features in MQ V9.

(Note: This is actually applicable to MQ V7, MQ V8 and MQ V9 as the underlying mechanisms are the same in all these cases)

**Introduction**

MQ's Programmable Change Format (PCF) messages are a flexible, powerful way in which to query and modify attributes of a Queue Manager, and the PCF classes provided in the MQ Classes for Java provide a convenient way of accessing their functionality in a Java application.

The functionality can also be accessed from MQ Classes for JMS, but it's important to be aware of a potential "gotcha" in this context.

As I've had 3 people contact me having hit this "gotcha" in the last few weeks, it's probably worth sharing the issue and the solution.

**The common model for processing responses to PCF requests in JMS**

A common approach to processing PCF responses in JMS is to extract the bytes payload of the message, wrap it in a DataInputStream and pass it to the com.ibm.mq.headers.pcf.PCFMessage constructor.

Something like this :-

     Message m = consumer.receive(10000);

    //Reconstitute the PCF response.

    ByteArrayInputStream bais =

 new ByteArrayInputStream(((BytesMessage)m).getBody(byte[].class));

    DataInput di = new DataInputStream(bais);

    PCFMessage pcfResponseMessage = new PCFMessage(di);

See <http://www-01.ibm.com/support/docview.wss?uid=swg21395682>& <https://www.ibm.com/developerworks/community/blogs/messaging/entry/using_pcf_with_mq_jms?lang=en> for some examples

Unfortunately, this is not a totally reliable approach for all platforms - in general it will work for big-endian platforms, but not for little-endian platforms.

**What's the problem here?**

The problem is that in parsing the message headers, the PCFMessage class has to deal with issues of numeric encoding - the headers contain length fields which are in some encoding (big- or little-endian).

If we pass a "pure" DataInputStream to the constructor, PCFMessage has no good indication of the encoding and has to assume a default - quite possibly incorrectly.

If this arises, you'll mostly likely see it manifested as a "MQRCCF\_STRUCTURE\_TYPE\_ERROR" (reason code 3013) in the constructor :-

com.ibm.mq.headers.MQDataException: MQJE001: Completion Code '2', Reason '3013'.

        at com.ibm.mq.headers.pcf.PCFParameter.nextParameter(PCFParameter.java:167)

        at com.ibm.mq.headers.pcf.PCFMessage.initialize(PCFMessage.java:854)

        at com.ibm.mq.headers.pcf.PCFMessage.<init>(PCFMessage.java:156)

This almost invariably means that the encoding has been misinterpreted - we've probably read little-endian data and interpreted it as big-endian.

**The Solution**

The way to avoid this is to pass the PCFMessage constructor something which will tell it the numeric encoding of the data it is working with.

So we need to make a MQMessage from the data we received.

Here's the sort of code to use (Disclaimer: this is an outline and I provide no warranty - in particular you need to add some error handling!):-

      // get a response into a JMS Message

      Message receivedMessage = consumer.receive(10000);

      BytesMessage bytesMessage = (BytesMessage) receivedMessage;

      byte[] bytesreceived = new byte[(int) bytesMessage.getBodyLength()];

      bytesMessage.readBytes(bytesreceived);

      // convert to MQMessage then to PCFMessage

      MQMessage mqMsg = new MQMessage();

      mqMsg.write(bytesreceived);

      mqMsg.encoding = receivedMessage.getIntProperty("JMS\_IBM\_Encoding");

      mqMsg.format = receivedMessage.getStringProperty("JMS\_IBM\_Format");

      mqMsg.seek(0);

      PCFMessage pcfMsg = new PCFMessage(mqMsg);

 **This works both ways...**

The same category of problem can occur with PCF Messages you are putting into a JMS BytesMessage to send to a queue manager via a JMS Producer.

The converse approach will work here, something like the code below  (assume "pcfMessage" is the PCF Message you wish to send and "message" is the JMS BytesMessage that you want to wrap it in) :-

        MQMessage m = new MQMessage();
        m.encoding = MQConstants.MQENC\_INTEGER\_NORMAL;
        m.characterSet = 1208;
        pcfMessage.write(m);

        m.setDataOffset(0);

        byte[] ba = new byte[m.getDataLength()];
        m.readFully(ba);
        message.writeBytes(ba);

**Finally**

I hope this is helpful. If you have questions regarding the use of the PCF classes in JMS, please feel free to reach out to me - my email address is mbluemel@uk.ibm.com