Co-Author [Mohab El-Hilaly]
Originally published on January 08, 2016

The data types to implement the internal buffer and string handling structures in IBM Integration Bus can limit the size of data that can be handled in a single message. IBM Integration Bus has two main data size limits. First, it has a 2-Gb limit on the index of position within a bitstream. Second, it has a 4-Gb limit on the size of internal data buffers. This tutorial demonstrates how to send and receive SOAP messages that have large attachments by using the MTOM and the MIME parser.

The 2-Gb limit

The parser interface uses a signed int data type to record positions within bitstreams. The signed int type has a 2-Gb limit. By extension, the parser interface cannot access any point in the bitstream beyond the 2-Gb mark.

The 4-Gb limit

The second limit is in the data structure to store data buffers that represent the parse tree data in the logical model of a message. This structure uses the unsigned int data type to represent the size of an internal buffer and any position information in this buffer. The unsigned int data type has a 4-Gb limit.

At first, using the additional 2 Gb that is available in the data buffers might seem impossible because an input bitstream is limited to 2 Gb in size. However, where CHARACTER data is stored in the parse tree, it is represented in UCS-2 (a UTF-16 variant). This code page is a double-byte code page. Therefore, each single character in a single-byte code page, such as ASCII or UTF-8, is represented by two characters in a message tree. For example, consider an entire 2-Gb input stream that is represented as a single CHARACTER element in the parse tree.

Similarly, you can create elements directly in the tree by using the extended SQL (ESQL), Java, C, or .NET interfaces. In these cases, element creation is difficult to achieve without rapidly exhausting the memory that is available on the system by using buffer expansion. However, theoretically, you can create elements directly with data buffers that are bigger than 2 Gb. But, the 4-Gb limitation is still in effect.

Consequences of the limits on flow design

The most obvious consequence of these limitations is that the maximum size of a single input message, read by one invocation of an input node, is limited to 2 Gb. Similarly, because the CHARACTER data expands in the message tree, the largest size of a single element in the message tree is 2-Gb characters.

In addition, some types of processing, such as needing to Base64 encode data or to hex encode data, can cause the required space to store the output in the tree to be larger than the original source data. For example, if processing requires hexBinary encoding of UTF-8 data, four times the space is required in the parse tree. Compared to the input data, each single byte is represented as 2 bytes that contain the hex representation. Then, each of these 2 bytes is stored as 2-byte UCS-2 characters. These factors limit the size of an element that undergoes this type of processing to 1 Gb of input data.

In some cases, using streaming transports, such as FileInput nodes or TCP/IP nodes, can allow an entire large message to be split into several smaller records. By using this technique, input messages that are larger than 2 Gb can be processed. However, a single invocation of the message flow is never presented with more than 2 Gb of data.

In other cases, the choice of transport is constrained. Not all transports in IBM Integration Bus support the concept of streaming. In this case, careful usage of the message tree and a knowledge of how data is stored and expanded can lead to improvements in the maximum size of a message that a message flow can handle. For example, binary large object (BLOB) data is not expanded into UCS-2 in the tree. If parts of the message can be represented as binary content, ensuring that this data is treated as binary throughout all processing can allow the processing of larger messages.

Use case overview

Considering the various backend systems that IBM Integration Bus can integrate with, some have limitations with the integration methods. Specifically, the limitations refer to the ability to send and receive a large file by using web services. Transferring large files by using HTTP or web services is not the most efficient method. However, sometimes this integration method is the only one that is available.

You can choose from several methods to send and receive a file by using a web service call:

- Send binary data directly in the SOAP request within a CDATA tag.
- Send a Base64 encoded file within the SOAP request.
- Use SOAP with attachments.
- Use the Message Transmission Optimization Mechanism (MTOM).

For information about the different methods and file transmission for using web services, see File handling in WebSphere Message Broker V6.1. For the use case in this tutorial, the backend service supports the MTOM. The next section explains how to send and receive a file by using the MTOM while bypassing the internal limitation on the file size in IBM Integration Bus.

Implement the MTOM by using an HTTP node with the MIME parser

In a typical scenario, the SOAP request nodes in IBM Integration Bus support the MTOM with the following conditions:

- The MTOM is enabled on the WS-Extensions tab.
- In the preceding input or transformation node, the Validation property is set to **Content and Value**.
- No child exists in the SOAP.Attachments section of the message tree.
- Elements that exist in the output message are defined within the schema as base64Binary.

Without the limitation on the element size in the tree in IBM Integration Bus, this solution might be costly in terms of processing times and memory consumption. A large proportion of this cost might be incurred by validating an enormous SOAP message. In addition, the cost of encoding the data, by using Base64 encoding or hex encoding, increases with the size of the message. The cost might be in validating an enormous SOAP message and the need to encode the file to Base64 before the SOAP request node.

The workaround to this limitation provides a more efficient solution in terms of processing times and memory consumption. This solution avoids validating the SOAP request and encoding the file to Base64 before sending it to the web service.

The following sections show how to send and receive a large file by using the HTTP node and the MIME parser. In this example, the same service expects a file in the request message and returns the file in the response.

Send a file

Using the HTTP node with the MIME parser gives you the ability to bypass the file size limitation in IBM Integration Bus. In essence, an MTOM message is a multipart message on the wire. The first part of the message is the SOAP envelope that references the file by an identifier (ID). The second part of the message is the file that is identified by an ID.

Listing 1 shows an example of the MTOM message on the wire.

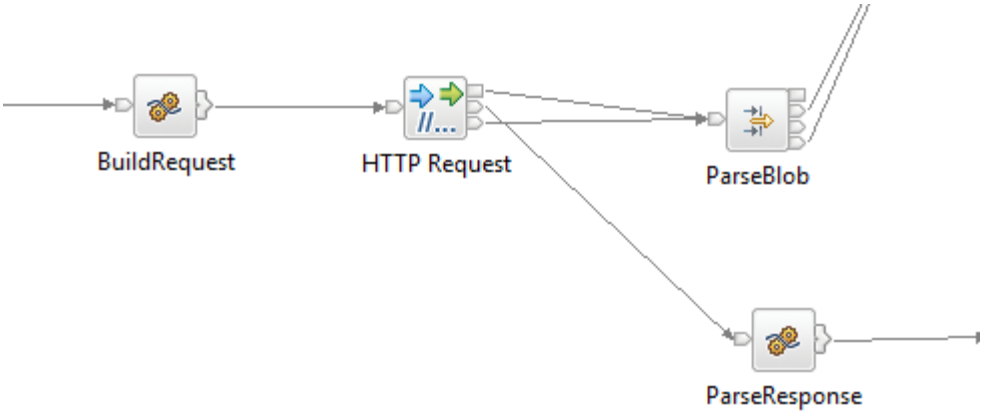
Listing 1. MTOM message on the wire

```
Content-Type: multipart/related; boundary= dwMIMEBoundry
;
type="application/xop+xml"; start="<soap@ibm.com>";
start-info="text/xml"; charset=UTF-8
u
--dwMIMEBoundary
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
   <soap@ibm.com>

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <Request>
      <input>
    <fileName></fileName>
        <file>
          <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
               href="cid:thefile@ibm.com"/>
        </file>
      </input>
    </Request>
   </soapenv:Body>
</soapenv:Envelope>
--dwMIMEBoundary
content-type: text/plain
content-transfer-encoding: binary
content-id:
         <thefile@ibm.com>

... binary data goes here ...
```

As mentioned previously, the MTOM and the MIME parser can help bypass the size limitation.

Figure 1 shows a portion of a subflow that implements the MTOM by using the MIME.

Figure 1. Subflow to implement the MTOM



Building the MIME message occurs in the BuildRequest ESQL Compute node. Listing 2 shows the implementation of this node. In this code, the file exists in the environment before the Compute node. The code has three sections. Each section corresponds to a part of the MTOM message:

- Section 1 defines the boundary of the start sections and any HTTP headers that are needed, such as the URL of the web service.
- Section 2 creates the SOAP request message and defines the "start" part of the message.
- Section 3 defines the file part of the message with the same content ID that is defined in the SOAP envelope.

Listing 2. Node implementation

```
----------------------------------------Section 1----------------------------------------
DECLARE MIMEBoundary CHAR 'dwMIMEBoundary';
DECLARE url char '';
SET OutputLocalEnvironment = InputLocalEnvironment;
SET OutputLocalEnvironment.Destination.HTTP.RequestURL ='theWebService.com';
SET OutputRoot.Properties.ContentType = 'multipart/form-data; boundary=' || '"'||
MIMEBoundary||'"';
SET OutputRoot.HTTPRequestHeader."X-Original-HTTP-URL" ='theWebService.com';
SET OutputRoot.HTTPRequestHeader."Content-Type" = 'multipart/related; type="application/xop+xml";
start="<soapenv@ibm.com>"; start-info="text/xml"; boundary=' || '"'|| MIMEBoundary||'"';
SET OutputRoot.HTTPRequestHeader."SOAPAction"='';
SET OutputRoot.HTTPRequestHeader."MIME-Version"='1.0';
CREATE FIELD OutputRoot.MIME TYPE Name;
DECLARE mm REFERENCE TO OutputRoot.MIME;
CREATE LASTCHILD OF mm TYPE Name NAME 'Parts';
CREATE LASTCHILD OF mm.Parts TYPE Name NAME 'Part';

----------------------------------------Section 2----------------------------------------
Creating the Soap Request message in the environment in order to cast as BLOB later.
CREATE LASTCHILD OF Environment.Variables DOMAIN 'XMLNSC' NAME 'XMLNSC';

DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
DECLARE xop NAMESPACE 'http://www.w3.org/2004/08/xop/include';
CREATE FIELD Environment.Variables.XMLNSC.soapenv:Envelope.soapenv:Body;
SET
Environment.Variables.XMLNSC.soapenv:Envelope.soapenv:Body.ns:Request.input.fileName='theFilename';
SET Environment.Variables.XMLNSC.soapenv:Envelope.soapenv:Body.ns:Request.input.file.xop:Include.
(XMLNSC.Attribute)href='cid:thefile@ibm.com';
DECLARE soapEnv BLOB ;
SET  soapEnv = CAST (ASBITSTREAM( Environment.Variables.XMLNSC) AS BLOB CCSID 1208);

------Doing first part of the message which contains the soap envelope-------------

DECLARE Part1 REFERENCE TO mm.Parts.Part[1];
CREATE FIELD Part1."Content-Type" TYPE NameValue VALUE 'application/xop+xml; charset=UTF-8;
type="text/xml"';
CREATE FIELD Part1."Content-ID" TYPE NameValue VALUE '<soapenv@ibm.com>';
CREATE LASTCHILD OF Part1 TYPE Name NAME 'Data';
CREATE LASTCHILD OF Part1.Data DOMAIN('BLOB') PARSE(soapEnv);

----------------------------------------Section 3----------------------------------------
Adding file in the second part of the message

CREATE LASTCHILD OF mm.Parts TYPE Name NAME 'Part';
DECLARE refPart REFERENCE TO mm.Parts;
DECLARE Part2 REFERENCE TO refPart;
MOVE Part2 TO refPart.Part[2];
CREATE FIELD Part2."Content-Disposition" TYPE NameValue VALUE 'attachment; name='||'"file";
filename='||'"thefile.zip"';
CREATE FIELD Part2."Content-Type" TYPE NameValue VALUE 'application/octet-stream';
CREATE FIELD Part2."Content-Transfer-Encoding" TYPE NameValue VALUE 'binary';

CREATE FIELD Part2."Content-ID" TYPE NameValue VALUE '<thefile@ibm.com>';
CREATE LASTCHILD OF Part2 TYPE Name NAME 'Data';
CREATE LASTCHILD OF Part2.Data DOMAIN('BLOB') PARSE(Environment.Variables.file);
```

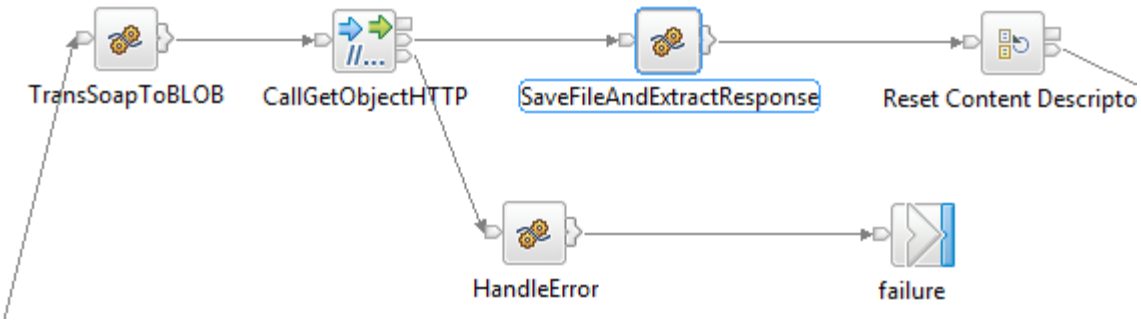
Receive a file

The file size limitation also applies when receiving a file from a SOAP call response. To bypass the size limitation, you use a MIME parser with an HTTP node to receive a large file with the MTOM.

To ensure that the backend service responds by using a multipart message and the MTOM, the service request must be a multipart message. However, the backend service must support the MTOM.

Figure 2 shows a portion of the subflow that implements the MTOM by using the MIME.

Figure 2. Subflow to implement the MTOM



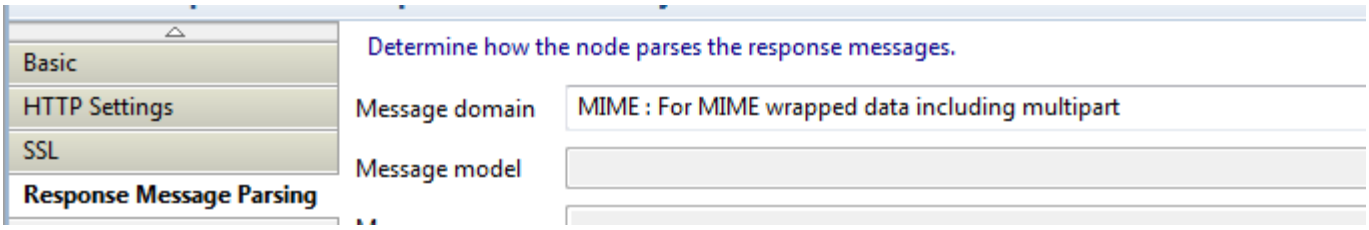
Building the SOAP request message as a multipart message occurs in the TransSoapToBlob Compute node. In Listing 3, a SOAP message is in an environment variable before the Compute node.

Listing 3. TransSoapToBlob Compute node implementation

```
            DECLARE soapString CHAR;
            SET soapString= Environment.Variables.SoapString;
            SET Environment.MIMEBoundary = 'MIMEBoundary';
            SET OutputLocalEnvironment = InputLocalEnvironment;
            SET OutputLocalEnvironment.Destination.HTTP.RequestURL =
                 'theservice.com';
            SET OutputRoot.Properties.ContentType = 'multipart/form-data; boundary='
|| '"'|| Environment.MIMEBoundary||'"';
            SET OutputRoot.HTTPRequestHeader."X-Original-HTTP-URL"
                 ='theservice.com';
            SET OutputRoot.HTTPRequestHeader."Content-Type" = 'multipart/related;
                 type="application/xop+xml"; start="<rootpart@ibm.com>";
                 start-info="text/xml"; boundary='
|| '"'|| Environment.MIMEBoundary||'"';
             SET OutputRoot.HTTPRequestHeader."SOAPAction"='';
             SET OutputRoot.HTTPRequestHeader."MIME-Version"='1.0';
             CREATE FIELD OutputRoot.MIME TYPE Name;
             DECLARE mm REFERENCE TO OutputRoot.MIME;
             CREATE LASTCHILD OF mm TYPE Name NAME 'Parts';
             CREATE LASTCHILD OF mm.Parts TYPE Name NAME 'Part';
             DECLARE soapEnv BLOB ;
             SET soapEnv = CAST (soapString AS BLOB CCSID 1208);
             DECLARE Part1 REFERENCE TO mm .Parts.Part[1];
             CREATE FIELD Part1."Content-Type" TYPE NameValue VALUE
'application/xop+xml; charset=UTF-8; type="text/xml"';
             CREATE FIELD Part1."Content-ID" TYPE NameValue VALUE
                 '<rootpart@ibm.com>';
             CREATE LASTCHILD OF Part1 TYPE Name NAME 'Data';
             CREATE LASTCHILD OF Part1.Data DOMAIN('BLOB') PARSE(soapEnv);
```

The second node of the displayed flow is the HTTP call. Ensure that the parser for Response Message Parsing is set to MIME as shown in Figure 3.

Figure 3. Response Message Parsing setting



If the service returns a valid response, it consists of two parts. The first part of the message is the SOAP response. The second part of the message is the file.

The second Compute node, SaveFileAndExtractResponse, takes the SOAP response that is put on the wire as a BLOB and saves the file in the environment. Listing 4 shows the code to achieve it.

Listing 4. SaveFileAndExtractResponse Compute node implementation

```
            SET OutputRoot.BLOB.BLOB = InputRoot.MIME.Parts.Part[1].Data.BLOB.BLOB;
            Set Environment.Variables.File =
InputRoot.MIME.Parts.Part[2].Data.BLOB.BLOB;
```

The final node in this branch of the flow is the ResetContentDescriptor node, which parses the BLOB (SOAP response) as XMLNSC.

Conclusion

This tutorial explained how to maximize the size of a message that can be handled by a message flow. You learned how to use the MIME parser to ensure that binary parts of the tree are not expanded into character data unnecessarily. Handling a message of this size requires an extremely large amount of memory to be available on the system that is hosting the IBM Integration Bus run time. In general, treat messages of this scale as streaming data. However, where this approach is not possible, use the techniques that are described in this tutorial to send and receive large files from and to a web service by using the MIME parser. When you do, stay within the limitations of the data structures that are available in IBM Integration Bus.

Resources

- For more information about the SOAP Message Transmission Optimization Mechanism (MTOM), see the IBM Documentation.
- Learn more about the different methods and file transmission for using web services in File handling in WebSphere Message Broker V6.1.
- See how to use the SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes in the IBM Documentation.
- Find IBM Integration Bus tutorials in the Technical library in IBM developerWorks® WebSphere zone.

- For the news and information about IBM Integration Bus and related products, see the developerWorks WebSphere zone.
- A downloadable version of this content is attached as a PDF.