Using the IBM Integration Bus EmailInput node in your ESB

Alasdhair M. Buchanan August 20, 2014

Starting with WebSphere Message Broker V7, the EmailInput node has enabled you to provide input to a message flow via an e-mail, and the functionality remains the same in IBM Integration Bus V9. This article provides a simple example with the e-mail consisting of either text or a single attachment, and describes possible issues when you are processing more complex e-mail messages. It also shows you how to configure IBM Integration Bus to use the EmailInput node.

Design considerations

Choosing your protocol

The EmailInput node uses one of two protocols: Internet Message Access Protocol (IMAP) or Post Office Protocol 3 (POP3).

IMAP supports both online and offline modes of operation. E-mail clients using IMAP generally leave messages on the server until the user explicitly deletes them. This and other characteristics of IMAP operation allow multiple clients to manage the same mailbox. Most e-mail *clients* support IMAP in addition to POP to retrieve messages, but fewer e-mail *services* support IMAP. IMAP offers access to the mail storage. Clients can store local copies of messages, but this storage is considered a temporary cache.

POP supports simple download-and-delete requirements to access remote mailboxes (termed maildrop in the POP Request for Comments (RFC)). Although most POP clients have an option to leave mail on the server after download, e-mail clients using POP generally connect, retrieve all messages, store them on the user's PC as new messages, delete them from the server, and then disconnect.

The choice of which protocol to use may be very simple: the e-mail service may support only POP3. However, if it also supports IMAP, then you must choose between the two.

A major consideration is that each mailbox may only be read by one message flow. Typically, this message flow provides the single point of processing for an e-mail message. It takes the input

message, transforms it in some way, and outputs it for further processing using a different protocol. In this case, the multiple client capability of IMAP is irrelevant.

In the project that forms the basis of this article, initially IMAP was used. However, it was discovered that as e-mails and their attachments got larger (into the multiple megabytes), processing time to read the e-mail message increased. While a small e-mail (~ 1KB) was read in under a second, when the e-mail plus attachments exceeded 6MB, the time to read the message increased to over five minutes, because IMAP does not supply the message in one piece, but in 16KB chunks, as they are requested by the client. A number of tuning parameters may reduce this read time, but since there was a tight deadline, POP3 was chosen and the message read time for the 6+ MB message dropped to 2 seconds. This reduction was due to the message and its attachments being retrieved as a single request, rather than with a large number of small requests. The project had a complex infrastructure involving firewalls, routers, mail servers, and a specialist e-mail security device, which may have increased the read time.

Nevertheless, unless there is a requirement for, or a foreseeable future use of, the additional facilities of IMAP, experience from this project suggests using POP3 as the protocol.

Relating message flows to mailboxes

Each mailbox should only be accessed by a single message flow or thread. If two message flows or threads access the same mailbox, there is a strong risk that an e-mail may be processed twice, because the e-mail will be available to the second message flow while the first message flow is processing it.

A message flow can only access a single mailbox, so if you have fifteen mailboxes, you'll have fifteen message flows. In theory, if you have multiple EmailInput nodes in a message flow, each pointed at a different mailbox, one message flow could handle multiple mailboxes, but this configuration would require the message flow to be multi-instance, unless you were prepared to serially process the e-mails. If you use a multi-instance message flow, there is a risk, as mentioned above, that a second thread could try to process an e-mail that was already being processed.

Configuring the message broker

To configure the message broker instance for use with the EmailInput node, either create a security identity for the mailbox and specify a URL or create a security identity and a configurable service linked to the security identity. To create a security identity, provide the e-mail userid and password details for the mailbox Use the mqsisetdbparms command to set up the identity. For example, for a message broker named MB8BROKER, an e-mail user broker.example@mailserver.co.uk, and a password of easyone, the command would be:

```
mqsisetdbparms MB8BROKER -n email::example.identity
  -u broker.example@mailserver.co.uk -p  easyone
```

The value example.identitywould be used for the Security Identityparameter in the EmailInput node configuration. The URL is of the form `protocol://hostname:port`, where:

- `protocol` is either IMAP or POP3.

- `hostname` is either the Internet Protocol Version 4 (IPV4) TCP/IP address, or the DNS-resolvable host name of the e-mail server.
- `port` is the port number on which the e-mail server is listening for either POP3 or IMAP traffic.

For example: `POP3://mailserver.co.uk:110`

To create a configurable service, use the mqsicreateconfigurable service command. For the URL and security identity above, the command would be:

```
mqsicreateconfigurableservice MB8BROKER -c EmailServer -o myEmailConfigurableServiceName
  -n serverName,securityIdentity -v POP3://mailserver.co.uk:110,example.identity
```

When configuring the Email Server parameter on the EmailInput node, specify either a URL or a configurable service name. If a URL is specified, enter the Security identifyparameter.

Determining your message format and how to process it

The sample program supplied with the toolkit demonstrates the processing of an e-mail message that will contain either a message body or an attachment. Reality may be slightly more complex. In deciding the design of the message flow, consider the following questions:

- How many different sources will there be for the input e-mails? If the input e-mails are coming from a single source (only.user.sending@singlesource.com), the design may be much simpler than if the message flow must process from many sources, because the exact format of the message that will be received is dependent on not only on the message body and, if present, the attachments, but also on how the message body was constructed, what e-mail application was used, and the options selected by the sending user.
- What application will create the message received?. If the message is programmatically generated from a single application, there is a high degree of confidence that only messages in specific formats will be received. If the message is being generated by a variety of e-mail applications, either e-mail clients or browser based e-mail, the message flow may have to process a large variety of input structures.
- Are the messages generated by humans? If not, the structure of the messages will probably be of a defined format. If a human was involved, assume that the message will come in one of the many possible formats. The exact format will depend on the human and the options they have selected on their e-mail application.

Revisit your requirements

Before starting to design the message flow, revisit the requirements based on the issues above.

If the requirements give a very bounded source for the messages, such as single application generated messages with a limited number of possible formats, proceed to design the message flow. If the requirement is to simply process the message from a defined mailbox, it may be advisable to analyse the requirements more rigorously. If the requirement is to process any message that arrives in a mailbox, the processing will be a great deal more complex, and require substantially more effort in development and testing, than if the requirements can be limited to particular attachment types and a message body. Performing this review, or being aware of these

issues at the outset, will allow a much more accurate estimation of the effort required to process the input messages, and ensure that the correct expectations have been set.

Input structures

The input message will have three sections in the Root logical tree:

- `Properties` -- Normal IBM Integration Bus properties values
- `EmailInputHeader`-- Set by the EmailInput node -- see below
- `MIME` -- Set by the MIME parser -- see below

The EmailInput node provides a basic set of information in the Root.emailInputHeader logical tree structure, as shown in Table 1:

Table 1. emailInputHeader fields

Element Location	Element Data Type	Description
Root.EmailInputHeader.To	CHARACTER	Comma-separated list of e-mail addresses of primary recipients
Root.EmailInputHeader.Cc	CHARACTER	Comma-separated list of e-mail addresses of cc'd recipients
Root.EmailInputHeader.From	CHARACTER	Comma-separated list of e-mail addresses of senders of the e-mail
Root.EmailInputHeader.ReplyTo	CHARACTER	Comma-separated list of e-mail addresses of reply targets
Root.EmailInputHeader.Subject	CHARACTER	Summary of e-mail content or subject
Root.EmailInputHeader.Size	INTEGER	Size of e-mail, including any attachments, in bytes
Root.EmailInputHeader.SentDate	CHARACTER	Sent date of e-mail

The properties in **bold** in Table 1, will always be present with at least one value in each list. The Cc and ReplyTo lists may be NULL. The contents of the MIME section in the message tree is determined by the factors discussed above: presence (and type) of attachments, e-mail program, user options, and so on.

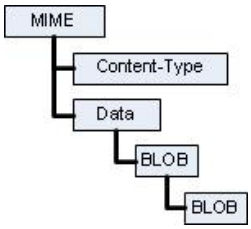
Types of message bodies

Simple text message body

In this section, the field names are given with the first letter of each individual word capitalised. Do *not* trust that this will be the case in every message. While the field name is constant, how it is capitalised depends on the generating application. Therefore, extract the field name and convert to uppercase before testing the field name.

At its simplest, the MIME section of an input message consists solely of a simple text message body, as shown in Figure 1:

Figure 1. Simple text MIME body

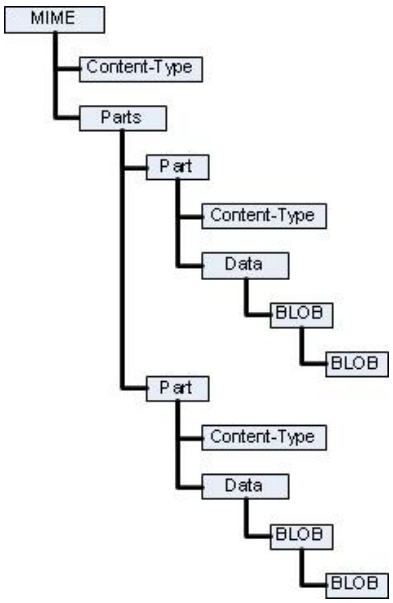


The text data is held in MIME.Data.BLOB.BLOB, as a BLOB. To convert to readable text, extract the character set that is used to encode the text, which is held in the Content-Type field, which is of the format contentType,charset="CharacterSet". contentType in this case will be text/plain and charset will be a character set such as ISO-8859-1. If the requirement is to output the message body as readable text, convert the charset value to a CCSID value that can be used by the CAST function. For information on the translations that must be performed, see Supported code pages in the IBM Integration Bus Knowledge Center.

Simple Text + HTML message body

If the sending user has selected that the message body will also be sent as HTML, the MIME section will be as in Figure 2.

Figure 2. Text + HTML MIME body



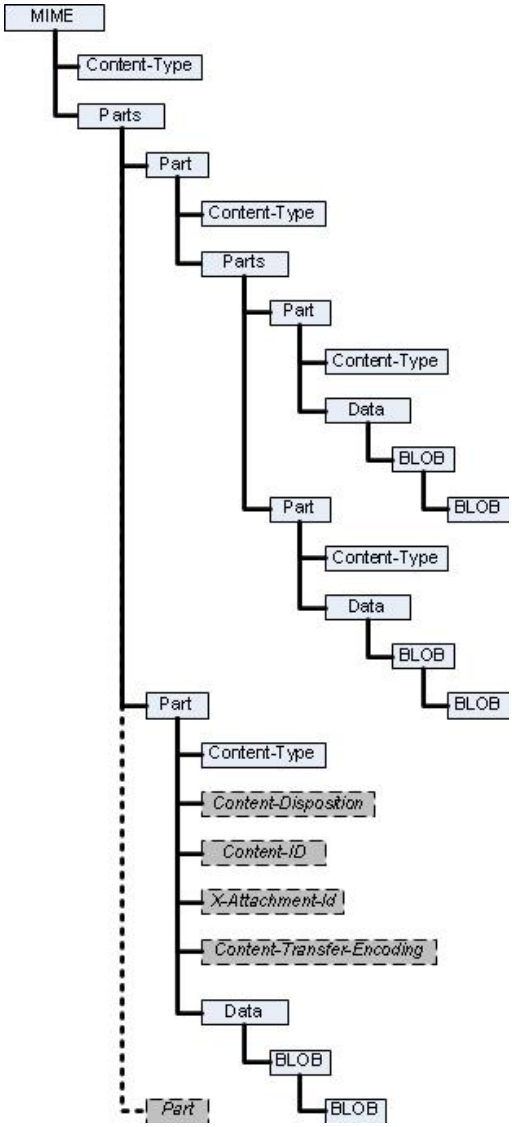
The simple text version will be in the MIME.Parts.Part[1] section and the HTML version will be in the MIME.Parts.Part[2] section. Again examine the Content-Typefield to determine the correct character set to be used.

In all the following examples, it is assumed that a message body, consisting of a text portion and an HTML portion, is present in the input message.

Attachment files or inline image files

If inline images or attachments are present, the MIME section format becomes increasingly complex, as Figure 3 illustrates:

Figure 3. Message body + attachments or inline images MIME body



The MIME.Parts.Part[1].Parts.Part[] section is made up of the message body. Thus, while in a message with only a text and HTML message body, the HTML text is referenced as MIME.Parts.Part[2], once an attachment or inline image is present, the HTML text is now referenced as MIME.Parts.Part[1].Parts.Part[2]. Attachments or inline images are referenced by MIME.Parts.Part[2] and subsequent Part sections in the MIME.Parts section.

Attachments

The Content-Disposition field in a particular Part section relating to an attachment contains details of the associated file name. The first part of the field will be Attachment and the second part of the

field will be name="filename.ext". The filename can be extracted for subsequent processing as required.

If there are multiple attachments with the same name in an e-mail, the X-Attachment-Id field will be present in the Part section for each attachment with the same name, which enables you to differentiate the individual files.

An attachment can also be Base64 encoded. In this instance, a field Content-Transfer-Encoding will be present in the Part section for endcoded attachment and will have the value Base64. This encoding requires that the result of CASTing the contents of the BLOB.BLOB field as CHARACTER (using the appropriate CCSID derived from the associated Content-Type field), to be passed into the BASE64DECODE function before the attachment data can be treated as readable text.

Inline images

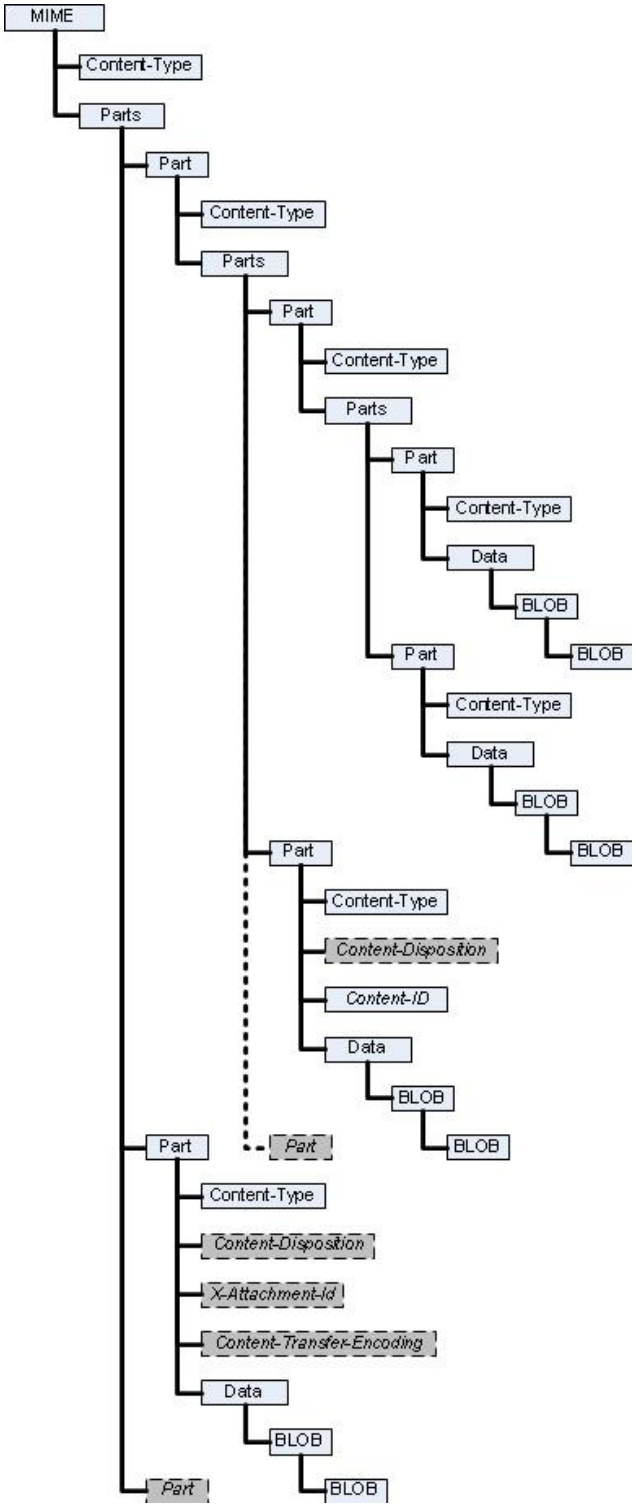
The Content-Disposition field in a particular Part section relating to an attachment contains details of the associated file name. The first part of the field will be Image/ and the second part of the field will be name="filename.ext". The filename can be extracted for subsequent processing as required.

When inline images are present, an associated Content-Id field will contain a unique identifier for each image. The format of the Content-Id field value is <contentIDValue>. This identifier is used in the text body to identify where the inline image is to be placed. The location is marked by the text cid:contentIDValue. If the processing of the message flow involves saving the inline images as separate files, and if there is a requirement to display the body text in its original state (with the inline image in place), then search the body text and replace each occurrence of cid:contentIDValue with the associated cid:{file_path}/file_name value. If the file containing the body text and the inline image files are output to the same directory, there is no need to specify {file_path}/.

Attachment files and inline image files

If both attachments and inline images are present, the MIME section format becomes even more complex, as shown in Figure 4:

Figure 4. Message body + attachments and inline images MIME body



When both attachment files and inline image files are present, the processing of the individual attachment files or inline image files is the same as when either attachments or inline images are present, as described above. The difference in processing arises from the change in where the individual component's data is located in the MIME body section.

The text message body is now located at MIME.Parts.Part[1].Parts.Part[1].Parts.Part[1], and the HTML message body at MIME.Parts.Part[1].Parts.Part[1].Parts.Part[2].

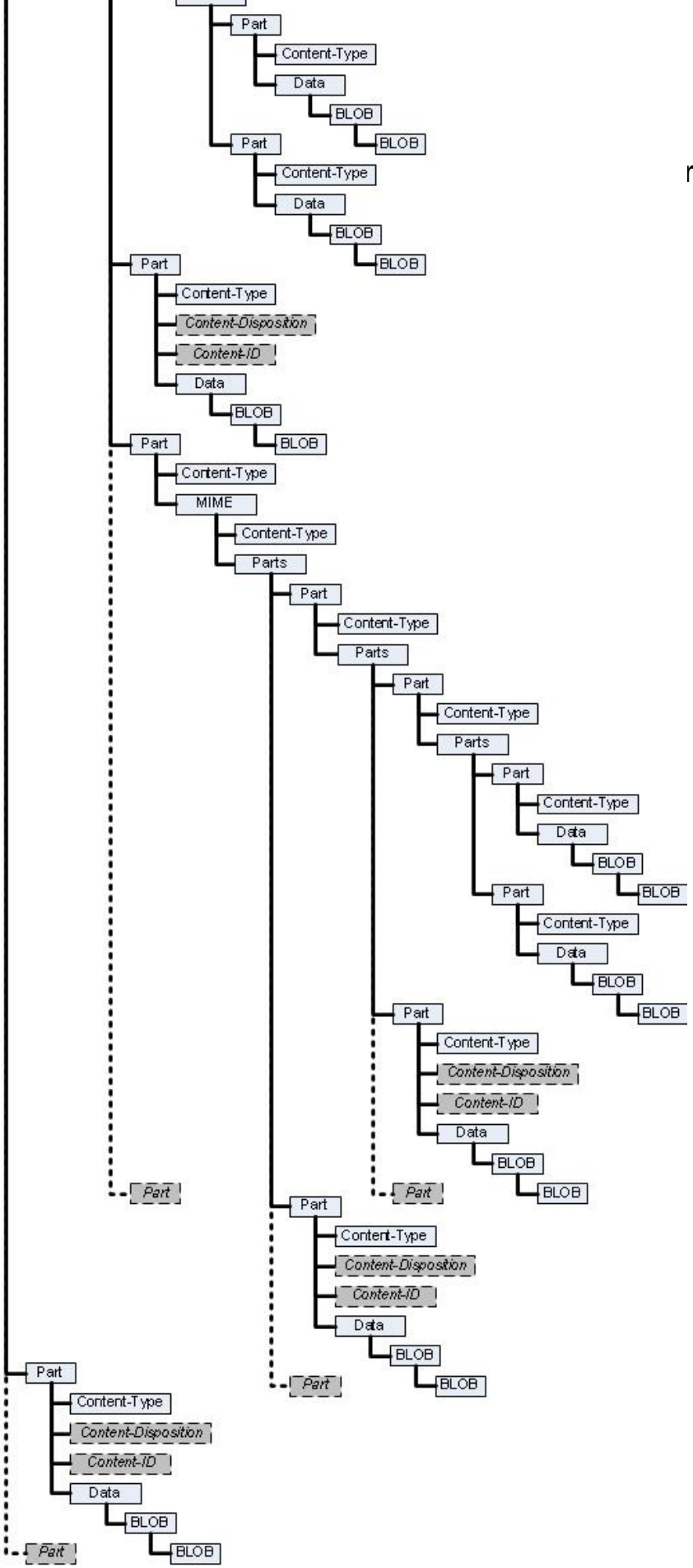
Attachments are referenced by MIME.Parts.Part[2] and subsequent Part sections in the MIME.Parts section.

Inline images are referenced by MIME.Parts.Part[1].Parts.Part[2] and subsequent Part sections in the MIME.Parts.Part[1].Parts section.

Embedded e-mail files

This section deals with a situation outside of the requirements for the project that is the basis of this article. It is included as a cautionary example of how complex the parsing of an e-mail message can become. You can save a message as an e-mail file (.eml extension), and then include this file in a new e-mail, yielding the structure shown in Figure 5:

mages with embedded e-



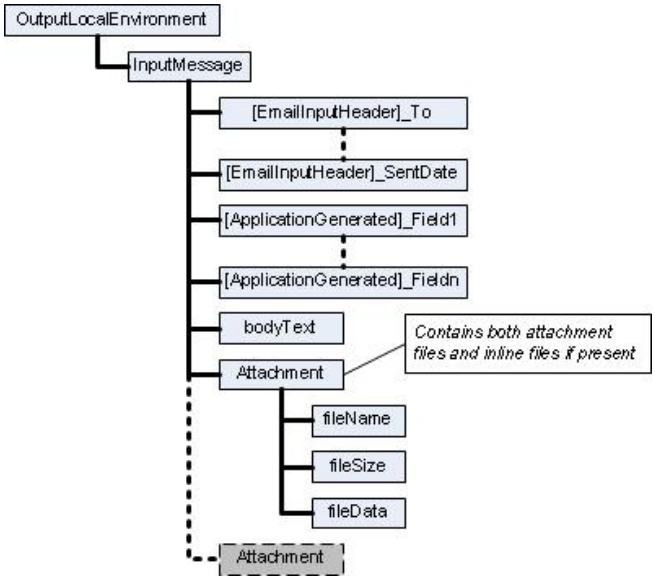
Embedded e-mails can themselves have embedded e-mails, greatly increasing the possible complexity. The solution for processing these messages will almost certainly be recursive.

Processing the e-mail message

When processing the message, if you have only one of the above formats to deal with (ignoring embedded e-mails that are outside the scope of this article), you can code the message flow to handle the message in a single pass as it traverses the MIME section in the tree

When multiple formats must be handled, a dual pass process is recommended. The first pass traverses the MIME section and writes the data from the various formats into a structure in the OutputLocalEnvironment that was developed for the particular processing requirements, as shown in Figure 6:

Figure 6. OutputLocalEnvironment message data structure



After the input message is processed into the standardised LocalEnvironment structure, the required processing can be readily achieved in one additional pass. The processing logic is much simpler and easier to understand compared to doing all of the required formatting and processing in a single pass.

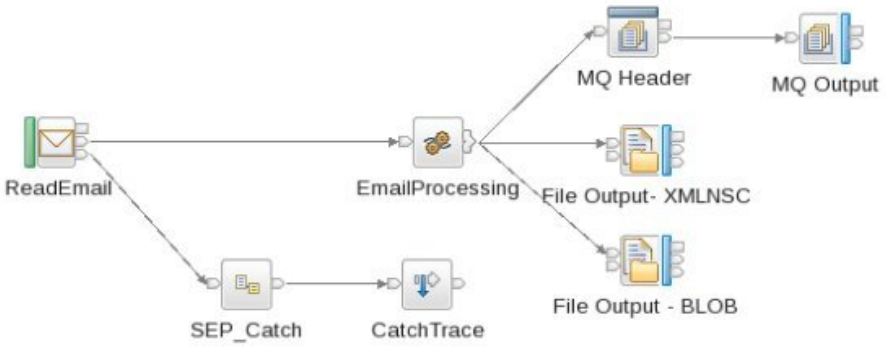
www Sample message flow

The message flow that was developed has the structure shown in Figure 7 below, with the following processing:

1. Read an e-mail.
2. Assign a unique identifier.
3. Write a manifest file containing the header parameters, the message body in readable format, and additional generated data in XML format.
4. Write each attached file, renaming it with the addition of the unique ID.

5. Write out each inline file, renaming it with the addition of the unique ID and the associated Content-Id. When inline files are present, adjust the message body so that the inline files are correctly inserted in the output manifest file.
6. Write a log message to a queue.
7. Write start, finish, and error status messages to a status queue.

Figure 7: Sample message flow



Parameters for the processing were supplied as user-defined properties. To process different mailboxes, the only change required to the message flow code is to change the Security Identify parameter value on the EmailInput node (ReadEmail). Here is the structure of the processing performed by the Compute node (EmailProcessing):

```
Common_EmailProcessor
|
|---SaveMessage()
| |
| SET attachments='True|False'
| |---ProcessMessageBodyText()
| | |
| | |---xlateUnprintableChars(inBLOB,[xlateBLOB])
| | |
| | SET emailCharSet from email
| | |---xlateCharSetToCCSID(emailCharSet,[CCSID])
| |
| |---ProcessAttachedFiles()
| | |
| | SET fileName
| | |---SaveFile(procPtr,tgtPtr,fileName)
|
|---GenerateLog()
| |
| |---CopyMessageHeaders()
| |
| |---CopyCommonText(srcPtr,outPtr)
| |
| |---CopyAttachmentText(srcPtr,outPtr)
|
|---OutputToQueue(logQueue)
| |
| SET TerminalOut = 'out'
| PROPAGATE-->TerminalOut
|
SET Status='ESB Started'
|---GenerateStatus(Status)
| |
| |---CopyMessageHeaders()
```

```
 |
 |---OutputToQueue(statusQueue)
 | |
 | SET TerminalOut = 'out'
 | PROPAGATE-->TerminalOut
 |
 |---OutputAttachmentFiles()
 | |
 | SET outFile='{UUID}.{attachment}'
 | |---OutputToFile(outFile,'BLOB')
 |  |
 |  SET TerminalOut = 'out2'
 |  PROPAGATE-->TerminalOut
 |
 |---FormManifestXML()
 | |
 | |---CopyMessageHeaders()
 | |
 | |---CopyCommonText(srcPtr,outPtr)
 | |
 | |---CopyAttachmentText(srcPtr,outPtr)
 |
 SET outFile='{UUID}.xml'
 |---OutputToFile(outFile,'XMLNSC')
 | |
 | SET TerminalOut = 'out1'
 | PROPAGATE-->TerminalOut
 |
 |---FormManifestXML()
 | |
 | |---CopyMessageHeaders()
 | |
 | |---CopyCommonText(srcPtr,outPtr)
 | |
 | |---CopyAttachmentText(srcPtr,outPtr)
 |
 |---OutputToQueue(triggerQueue)
 | |
 | SET TerminalOut = 'out'
 | PROPAGATE-->TerminalOut
 |
 SET Status='ESB Complete'
 |---GenerateStatus(Status)
 |
 |---OutputToQueue(statusQueue)
   |
   SET TerminalOut = 'out'
  PROPAGATE-->TerminalOut
```

The majority of this code is either specific to the required application processing, or else it sperforms functions that are standardised in a given environment, such as logging.

The two functions common to every implementation using the EmailInput node are: traversing of the MIME portion of the message tree in the ProcessMessageBody routine, and translating the character set into a CCSID in the xlateCharSetToCCSID routine. Here is the relevant code:

Traversing the MIME message Tree

```
…...
DECLARE bodyTextPtr REFERENCE TO InputRoot.MIME;
/*************************************************************/
/* Parse to the right Content Type & set pointer to body text     */
/*************************************************************/
/* Try to move to 'Parts' - assumes attachments may exist         */
```

```
/*************************************************************/
MOVE bodyTextPtr FIRSTCHILD NAME 'Parts';
IF LASTMOVE(bodyTextPtr)
THEN
 MOVE bodyTextPtr FIRSTCHILD NAME 'Part';
 /*********************************************************/
 /* MOVE must work - if not message MIME is wrong...      */
 /*********************************************************/
 IF NOT LASTMOVE(bodyTextPtr)
 THEN
  THROW USER EXCEPTION CATALOG 'BIPmsgs' MESSAGE 2951
   VALUES('MIME.Parts structure incorrect.');
 END IF;
 /*********************************************************/
 /* Try to move to 'Parts' - for Gmail type header        */
     /*********************************************************/
 MOVE bodyTextPtr FIRSTCHILD NAME 'Parts';
 IF LASTMOVE(bodyTextPtr)
 THEN
 /*****************************************************/
 /* Move to first 'Part' tag - now standardised       */
 /*****************************************************/
  MOVE bodyTextPtr FIRSTCHILD NAME 'Part';
  /*****************************************************/
  /* MOVE must work - if not message MIME is wrong...  */
  /*****************************************************/
  IF NOT LASTMOVE(bodyTextPtr)
  THEN
   THROW USER EXCEPTION CATALOG 'BIPmsgs' MESSAGE 2951
   VALUES('MIME.Parts.Part.Parts structure incorrect.');
  END IF;
  /*****************************************************/
  /* Try to move to 'Parts' - email contains inline    */
  /*****************************************************/
  MOVE bodyTextPtr FIRSTCHILD NAME 'Parts';
  IF LASTMOVE(bodyTextPtr)
  THEN
  /*****************************************************/
  /* Move to first 'Part' tag - now standardised       */
  /*****************************************************/
   MOVE bodyTextPtr FIRSTCHILD NAME 'Part';
   /*****************************************************/
   /* MOVE must work - if not message MIME is wrong.  */
   /*****************************************************/
   IF NOT LASTMOVE(bodyTextPtr)
   THEN
    THROW USER EXCEPTION CATALOG 'BIPmsgs'
    MESSAGE 2951
     VALUES('MIME.Parts.Part.Parts.Part.Parts structure incorrect.');
   END IF;
  END IF;
 END IF;
END IF;
….
```

At this point bodyTextPtr points to the section of the MIME message tree that contains the message body text, ready for subsequent processing:

Converting the e-mail character set to a CCSID

```
CREATE PROCEDURE xlateCharSetToCCSID(IN emailCharSet CHARACTER, OUT CCSID INTEGER)
BEGIN
/*************************************************************/
/* Test the supplied character set                           */
/*************************************************************/
```

```
 CASE UPPER(emailCharSet)
  WHEN 'ISO-8859-1'
      THEN
       SET CCSID = 819;
      WHEN 'ISO-8859-2'
      THEN
       SET CCSID = 912;
      WHEN 'ISO-8859-3'
      THEN
       SET CCSID = 913;
      WHEN 'ISO-8859-4'
      THEN
       SET CCSID = 914;
      WHEN 'ISO-8859-5'
      THEN
       SET CCSID = 915;
      WHEN 'ISO-8859-6'
      THEN
       SET CCSID = 1089;
      WHEN 'ISO-8859-7'
      THEN
       SET CCSID = 813;
      WHEN 'WINDOWS-1252'
      THEN
   SET CCSID = 1252;
  WHEN 'US-ASCII'
  THEN
   SET CCSID = 367;
  WHEN 'UTF-8'
  THEN
   SET CCSID = 1208;
  WHEN 'UTF-16'
  THEN
   SET CCSID = 1204;
  WHEN 'UTF-32'
  THEN
   SET CCSID = 1236;
  WHEN 'SCSU'
  THEN
   SET CCSID = 1212;
  WHEN 'BOCU-1'
  THEN
   SET CCSID = 1214;
  WHEN 'CESU-8'
  THEN
  SET CCSID = 9400;
  ELSE
  /*******************************************************/
  /* Not found:                                          */
  /* indicates unsupported character set supplied        */
  /* Required action - complain bitterly                 */
  /*******************************************************/
          THROW USER EXCEPTION CATALOG 'BIPmsgs' MESSAGE 2951
             VALUES('Unsupported character set supplied:', emailCharSet);
 END CASE;
END;
```

Conclusion

Configuring IBM Integration Bus to support the EmailInput node is straightforward. Choosing the between IMAP and POP3 for the protocol can be critical for performance if large messages will be processed. For the example in this article, POP3 is probably the best choice.

The format of the input e-mail message depends on the source application, user options, presence of attachments and/or inline data files, and other factores. Therefore it is vital to determine which formats will be handled by a message flow. If a single format will be handled, processing can be done in a single pass within the flow, but if multiple formats will be processed, a dual-pass design is preferred, with the input message written in a standardised format in the LocalEnvironment on the first pass, and then processed on the second pass.

Acknowledgments

The author would like to thank Tony J Cox, from the WebSphere MQ Level-3 Support team, for reviewing this article.

Related topics

- **IBM Integration Bus resources**
  - IBM Integration Bus V9 Knowledge Center
    A single portal to all IBM Integration Bus documentation, with conceptual, task, and reference information on configuring, migrating to, and using IBM Integration Bus.
  - IBM Integration Bus developer resources page
    Downloads, tutorials, education, product info, forums, how-to articles, and other resources to help you use IBM Integration Bus to enable connectivity and transformation in heterogeneous IT environments for businesses of any size using a wide range of platforms, including cloud and z/OS.
  - IBM Integration Bus product family page
    Product features, use cases, and resources.
  - Video: What's new in IBM Integration Bus
    A short YouTube video showing key IBM Integration Bus features.
  - Download IBM Integration Bus Developer Edition
    A lightweight edition that you can use for evaluation, development, unit test, and other scenarios.
  - Follow IBM Integration Bus on Twitter
    Latest IBM Integration Bus news and announcements,
  - IBM Integration Bus forum
    Forum on mqseries.net for user questions, answers, and tips.
  - Track IBM Integration Bus user requirements
    Create, view, and track IBM Integration Bus user requirements.
- **WebSphere resources**
  - developerWorks WebSphere
    Technical information and resources for developers who use WebSphere products. developerWorks WebSphere provides downloads, how-to information, support resources, and a free technical library of more than 2000 technical articles, tutorials, best practices, IBM Redbooks, and product manuals.
  - Most popular WebSphere trial downloads
    No-charge trial downloads for key WebSphere products.
  - WebSphere forums
    Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.
  - WebSphere demos
    Download and watch these self-running demos, and learn how WebSphere products can provide business advantage for your company.
  - WebSphere-related articles on developerWorks
    Over 3000 edited and categorized articles on WebSphere and related technologies by top practitioners and consultants inside and outside IBM. Search for what you need.
  - WebSphere-related books from IBM Press
    Convenient online ordering through Barnes & Noble.
  - WebSphere-related events

Conferences, trade shows, Webcasts, and other events around the world of interest to WebSphere developers.

- **developerWorks resources**
  - Trial downloads for IBM software products
    No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.
  - developerWorks business process management developer resources
    BPM how-to articles, downloads, tutorials, education, product info, and other resources to help you model, assemble, deploy, and manage business processes.
  - developerWorks blogs
    Join a conversation with developerWorks users and authors, and IBM editors and developers.
  - developerWorks tech briefings
    Free technical sessions by IBM experts to accelerate your learning curve and help you succeed in your most challenging software projects. Sessions range from one-hour virtual briefings to half-day and full-day live sessions in cities worldwide.
  - developerWorks podcasts
    Listen to interesting and offbeat interviews and discussions with software innovators.
  - developerWorks on Twitter
    Check out recent Twitter messages and URLs.
  - IBM Education Assistant
    A collection of multimedia educational modules that will help you better understand IBM software products and use them more effectively to meet your business requirements.