WebSphere MQ Integrator

MRM Primer

Version 2.1

IC00-IA7A-00

WebSphere MQ Integrator

MRM Primer

Version 2.1

IC00-IA7A-00

Note!
Before using this information and the product it supports, be sure to read the general information under Appendix L, "Notices" on page 123.

First edition (August 2002)

This edition applies to Version 1.0 of SupportPac™ IA7A MRM Primer and to all subsequent releases and modifications until otherwise indicated in new editions. The use of this SupportPac is in conjunction with IBM® WebSphere MQ Integrator Version 2.1 or WebSphere MQ Integrator Broker Version 2.1.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

User Technologies (MP095)
IBM United Kingdom Laboratories
Hursley Park
Hursley
Hampshire, SO21 2JN
England

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you. You may continue to use the information that you supply.

Contents

Figures vii

Tables ix

About this SupportPac xi
Who this document is for xi
What you need to know to understand this document xi
General assumptions xii
C header files and Cobol copybooks xiii
Error messages during Check In and Deployment xiii
WebSphere MQ Integrator flexibility xiv
SupportPac contents xiv
Completing the exercises xiv
Testing the exercises xiv

Chapter 1. Introduction to messaging . . 1
WebSphere MQ 1
WebSphere MQ Integrator 2
High level overview 3
Summary 4

Chapter 2. Message processing concepts 5
Data interpretation 5
Physical format 6
Logical format 6

Chapter 3. Introduction to the MRM . . . 9
The logical message model 10
The buildtime and runtime environments 11
Buildtime environment 11
Runtime environment 11
The message repository 12
Creating message models 12
How the broker processes a message 12

Chapter 4. Importing C or Cobol data structures into the MRM 15
Reviewing assumptions for this exercise 15
Creating a message set 15
Adding physical format layers 16
Importing the data structure 17
Creating a logical message 19

Chapter 5. Creating and testing a basic message flow 21
Reviewing assumptions for this exercise 21
Creating the WebSphere MQ resources 22
Creating a basic message flow 22
Defining a broker in the domain 24
Assigning the message flow to the execution group 26
Deploying the message flow to the broker 27
Testing the basic message flow 29
Using WebSphere MQ Explorer 29
Using IH03 SupportPac 31

Chapter 6. The Trace node 35
Reviewing assumptions for this exercise 35
Use of the Trace node 35
Adding a Trace node to a message flow 36
Demonstrating trace 37

Chapter 7. Converting CWF to TDS . . 41
Reviewing assumptions for this exercise 41
Creating and assigning the message set 41
Creating the message set 42
Adding physical format layers 42
Importing the data structure and creating the associated message 42
Mapping the TDS physical layer 43
Delimiters 45
Assigning a message set to a broker 45
Create a CWF to TDS message flow 46
Adjusting the Input node properties for the message set 46
Adding a Compute node 47
Setting the Compute node to convert CWF to TDS 48
Defining the broker in the domain 51
Assigning the message flow to the broker . . . 51
Saving all the changes 51
Deploying the configuration to the broker . . . 52
Testing the CWF to TDS message flow 52
Error processing in a message flow 53

Chapter 8. Further CWF input processing 55
Reviewing the assumptions for this exercise . . . 55
Creating additional message sets 56
Adding physical format layers 56
Importing additional data structures 56
Creating the logical messages 57
Mapping additional TDS layers to the logical layers 57
Assigning the message sets to the broker 58
Creating the additional WebSphere MQ resources . 59
Creating the additional message flows 59
Setting the additional input and output node properties for CWF to TDS 60
Setting the additional compute node properties for CWF to TDS 61
Defining the broker in the domain 61
Assigning the message flows to the execution group 61
Deploying the configuration to the broker 62
Confirm error processing 62
Testing the additional message flows 62

Chapter 9. Further message transformation 65
Review the assumptions for this exercise 65
Create the message set. 66
Create a message set based on another message set 66
Add the message and message types to the message set workspace 67
Add an XML physical layer 67
Add an element to a message set 67
Create the message flow 69
Create the additional WebSphere MQ resources 70
Copy an existing message flow 70
Update the message flow nodes 70
Assign, deploy, and test the message flow 72
Assign the message set to the broker 72
Assign the message flow to an execution group 72
Deploy the configuration to the broker 72
Test the message flow 72

Chapter 10. Basic message manipulation 75
Reviewing assumptions for this exercise 75
Creating the message set 75
Create the message set and add the message and types 76
Copy a message set type 77
Create an element based on the new Type . . . 77
Create a type and add an element 78
Add a new message to the message set 79
Removing an element from a compound type . . 79
Create and add an element to a compound type 80
Creating the message flow 81
Create the additional WebSphere MQ resources 81
Copy an existing message flow 81
Update the message flow nodes 81
Assigning, deploying, and testing the message flow 84
Assign the message set to the broker 84
Assign the message flow to an execution group 84
Deploy the configuration to the broker 84
Test the message flow 85

Appendix A. Example C header files . . 87
Example 1: PaxData1.h 87
Example 2: PaxData2.h 87
Example 3: PaxData3.h 88
Example 4: PaxData4.h 89

Appendix B. Example Cobol copybook files 91
Example 1: PaxData1.cpy 91
Example 2: PaxData2.cpy 91
Example 3: PaxData3.cpy 92
Example 4: PaxData4.cpy 93

Appendix C. Example input message files 95
Example 1: PaxData1.ipt 95
Example 2: PaxData2.ipt 95
Example 3: PaxData3.ipt 95
Example 4: PaxData4.ipt 95

Appendix D. Example TDS output . . . 97
Example 1: From PaxData1.ipt 97
Example 2: From PaxData2.ipt 97
Example 3: From PaxData3.ipt 97
Example 4: From PaxData4.ipt 97

Appendix E. .XML trace file 99

Appendix F. .log trace file 101

Appendix G. Example CWF import report file 105

Appendix H. Variable conversion input files 107
Example 1: CWF input file (PaxData4_TR1_CWF.ipt) 107
Example 2: TDS input file (PaxData4_TR1_TDS.ipt) 108
Example 3: XML input file (PaxData4_TR1_XML.ipt) 108
Example 4: OTH (Other) input file (PaxData4_TR1_Other.ipt) 109

Appendix I. Variable conversion output files 111
Example 1: From CWF input file (PaxData4_TR1_CWF.ipt) 111
Example 2: From TDS input file (PaxData4_TR1_TDS.ipt). 111
Example 3: From XML input file (PaxData4_TR1_XML.ipt) 112
Example 4: From OTH (Other) input file (PaxData4_TR1_Other.ipt) 113

Appendix J. Manipulated output files 115
Example 1: From CWF input file (PaxData4_TR1_CWF.ipt) 115
Example 2: From TDS input file (PaxData4_TR1_TDS.ipt). 115
Example 3: From XML input file (PaxData4_TR1_XML.ipt) 116
Example 4: From OTH (Other) input file (PaxData4_TR1_Other.ipt) 117

Appendix K. Delimiter examples for PaxData4 files 119
PaxData4.h 120
PaxData4.cpy 120
A break down of the TDS output showing delimiter separation 121

Appendix L. Notices 123
Trademarks 125

Bibliography 127

Index . . . . . 129 **Sending your comments to IBM . . . 131**

Figures

1. WebSphere MQ Integrator Overview 3
2. MRM processes in the Control Center, the Configuration Manager, and the broker . . . 9
3. The Control Center on completion of Message Set creation 16
4. Typical error message generated on deploying to broker. 17
5. The Control Center after adding the PaxData1 C header to it. 19
6. MQSeries Explorer after adding input and output queues 22
7. The Control Center after adding MQInput and MQOutput nodes 24
8. The Control Center after adding the broker: topology view. 25
9. The Control Center after adding the broker: Assignments view 26
10. The Control Center after assigning the message flow to the broker 27
11. The Control Center Log view 28
12. The Control Center Operations view 28
13. WebSphere MQ Explorer showing MQSI_SAMPLE_QM queue manager queues . 29
14. WebSphere MQ Explorer Put Test Message dialog 30
15. WebSphere MQ Explorer Message Browser dialog 30
16. WebSphere MQ Explorer Properties for Message dialog 31
17. IH03 (rfhutil.exe) initial screen 32
18. IH03 (rfhutil.exe): Data tab 32
19. The Control Center and message flow to be modified 36
20. The Control Center showing the added Trace node 37
21. The Control Center after importing the message. 43
22. Add an existing Compound Type dialog 44
23. The Control Center after assigning a message set to the broker 46
24. A Compute node properties dialog. 49
25. The IH03 dialog after reading the test message. 53
26. The WebSphere MQ Integrator Control Center after assigning the message sets. 58
27. MQSeries Explorer after adding the additional local queues 59
28. The Control Center after assigning all the resources to the broker. 62
29. The Check Out List dialog 69
30. Screen shot of the Type tree that has just been created 79
31. The input and output message 81

Tables

1. Message set names to create 56
2. CWF header files: physical format layer parameters 56
3. CWF header files to import 57
4. Message, Message IDs, Message Type names to use in creating the messages 57
5. Type names requiring modification. 58
6. Message sets to assign to the broker (MQSI_SAMPLE_BROKER) 58
7. Input and output queues needed for the message flows 59
8. Message flow and node names 60
9. Input node properties needed for the message flows. 60
10. Output node properties needed for the message flows 60
11. Input and output queues needed for the message flows 61
12. Message flows to assign to the broker. . . . 61
13. Input and output queues needed for the message flows 63

About this SupportPac

This primer gives users who are new to WebSphere MQ Integrator and the Message Repository Manager (MRM) a basic understanding of how to use the MRM to create, deploy, and test a message set through a message flow. It is not intended to be a document that stands on its own as a reference guide, but must be used in conjunction with the existing documentation and help supplied with WebSphere MQ and WebSphere MQ Integrator to assist you to gain a quicker understanding of the MRM.

The primer and associated files:

- Provide an introduction to messaging
- Provide an introduction to message processing concepts
- Provide an introduction to the MRM
- Show how to import a data structures of increasing complexity (such as one defined in a C header file or Cobol copybook) and create the associated message set
- Show how to create a basic message flow and test it
- Provide an introduction to the Trace node
- Show how to create message sets and message flows to:
  - Convert a custom wire format (CWF) message into a tagged delimited string (TDS) message
  - Convert a message to a format specified in the input message
  - Convert a message to a format specified in the input message, remove the format element and add a date time stamp to the outgoing message

Where further information is available, this document points a referenceto it. The "Bibliography" on page 127 refers to the relevant manuals and other information supplied with WebSphere MQ Integrator.

Who this document is for

This document is for users who are unfamiliar with the MRM in WebSphere MQ Integrator Version 2.1. It is not intended as a comprehensive guide on the use of the MRM, but to provide the basic knowledge on which to develop further experience.

What you need to know to understand this document

You should be familiar with the concepts of WebSphere MQ and with the use of MQSeries Services and MQSeries Explorer.

You should be familiar with the concepts of WebSphere MQ Integrator and with message set and message flow design using the Control Center.

The term *local error log* is used within this document to mean the Event Log on Windows NT.

General assumptions

If you download and use the files in this SupportPac, the following are assumed:

- You have installed WebSphere MQ Integrator Version 2.1. Your installation options include the broker, the Configuration Manager, and the Control Center. Other components are optional for this purpose.
- You have set up WebSphere MQ, WebSphere MQ Integrator, and DB2 with the sample names used in the *WebSphere MQ Integrator Installation Guide for Windows NT and Windows 2000*. These are the names that are used within this SupportPac. If you use other names (where the sample names are quoted in this document) you must substitute your names for the sample ones. Examples of these names are:
  - Broker queue manager: MQSI_SAMPLE_QM
  - Broker: MQSI_SAMPLE_BROKER
  - Broker database: MQSIBKDB
  - Configuration repository database: MQSICMDB
  - Message repository database: MQSIMRDB

  Where an object is described in any of the processes and has a name in brackets after it (for example message set (MRMP_MS1)) the name in brackets is the one that has been used in the procedure and all following procedures and chapters that use that object. This is to enable you to more easily cross reference the text with screen shots.
- The queue manager (MQSI_SAMPLE_QM) has a dead letter queue (dead.letter.queue) for handling any undeliverable messages. As a minimum for WebSphere MQ and WebSphere MQ Integrator a dead letter queue should be defined for each queue manager. See the WebSphere MQ documentation for details on how to create a dead letter queue.
- You have created and started the Configuration Manager on a Windows NT system. The name of the queue manager is not assumed. However, if your broker and your Configuration Manager do not share the same queue manager, you must set up MQSeries communications between the two queue managers. The *WebSphere MQ Integrator Installation Guide for Windows NT and Windows 2000* provides instructions to help you to do this.
- You have created and started a broker on a Windows NT or Windows 2000 system that specifies queue manager MQSI_SAMPLE_QM. The *WebSphere MQ Integrator Installation Guide* provides instructions to help you to do this. You can use a different queue manager if you choose: if you do so, you must modify the scenario resources after you have imported them as indicated in the appropriate descriptions.
- You are using WebSphere MQ as the transport protocol between your application and your broker.
- You have created and initialized the DB2 databases and tables required.
- You have updated your broker domain topology to define the broker to which you intend to deploy the message flows. This document assumes that you are assigning message flows to the broker's default execution group: if you want to use another execution group, you can do so during the assignment step. For more information about updating your topology, see the *WebSphere MQ Integrator Using the Control Center* book.
- You have installed and are familiar with the SupportPac IH03: MQSeries Integrator V3 - Message display, test and performance utilities.

The *WebSphere MQ Integrator Administration Guide* contains detailed information about using the WebSphere MQ Integrator commands for creating brokers, starting brokers, and other related objects.

C header files and Cobol copybooks

The exercises in this primer have used the C header files PaxData1.h, PaxData2.h, PaxData3.h, and PaxData4.h as the primary source of data structures that have been imported into the MRM for you to work with.

The equivalent Cobol copybooks are also included with this primer and are PaxData1.cpy, PaxData2.cpy, PaxData3.cpy, and PaxData4.cpy. These files have the same structure and labels as the C header file and if you imported these instead of the C header files, they would create an almost identical set of elements, element values and compound types as the C header files.

There are two main differences that will affect the exercises you will be working through:

1. Labels from a Cobol copybook will import with all the labels in uppercase. You will need to make the appropriate substitutions for this in the exercises in this primer if you work with the supplied Cobol copybooks.
2. A character definition in a Cobol copybook (for example, `PaxSurname PIC X(20).`) will import as an MRM STRING type. It will have the padding character in the custom wire format (CWF) layer set to a SPACE and not the NUL that is set as a padding character if a C header file is imported.

   This will strip any trailing blanks from the input message fields that are written to the logical message tree and these blanks will not be in the output messages that can be seen in the appendices. If you need to keep these trailing blanks then the padding character can be updated to a NUL after importing the Cobol copybook.

Whether you use the C header files or the Cobol copybooks, once you have imported the data structure, the steps to complete each of the exercises are the same.

Error messages during Check In and Deployment

During the development and testing of message sets and message flows, you will almost certainly receive error messages when you check in information to the repository or deploy changes to the broker. For example, you might see a message dialog, or need to view information in a log file.

These error messages are self explanatory and provide information that indicates why the specified action has failed. Read the messages and make the appropriate changes to ensure completion the next time you submit this action.

It is not the purpose of this document to provide the answers to all the error messages that you will encounter. Where it is known that you could or will receive an error message, it is briefly discussed.

WebSphere MQ Integrator flexibility

The methods that are in this SupportPac give you a starting point from where you can build further experience. This document is not intended to provide a comprehensive solution to all the ways that you can accomplish the processes that are covered here. In many cases, there is more than one way that you can complete a process or a step within a process. These alternate ways are part of what gives WebSphere MQ Integrator its flexibility.

SupportPac contents

This SupportPac is supplied in a zip file that contains all the files that you need to use the scenarios described here:

- IA7A-00.pdf (this document)
- Files required for the exercises in this document
  - PaxData1.h
  - PaxData2.h
  - PaxData3.h
  - PaxData4.h
  - PaxData1.cpy
  - PaxData2.cpy
  - PaxData3.cpy
  - PaxData4.cpy
  - PaxData1.ipt
  - PaxData2.ipt
  - PaxData3.ipt
  - PaxData4.ipt
  - PaxData4_TR1_CWF.ipt
  - PaxData4_TR1_Other.ipt
  - PaxData4_TR1_TDS.ipt
  - PaxData4_TR1_XML.ipt

Completing the exercises

The exercises in this SupportPac increase in complexity. They often require earlier exercises to be completed before the later ones can be started. Where this is true, the exercise will state what you must have completed before you can complete the current exercise.

Testing the exercises

The exercises are provided as a guide to allow you to import data structures, create message sets, and create and test message flows. Each scenario includes some information on how you can test its operation. Each later chapter tends to rely on procedures finished in an earlier chapter: you must complete the procedures in that chapter. Where possible, procedures in earlier chapters have been identified to ensure that testing should complete successfully.

You will find the following helpful in testing:

- The IH03 SupportPac *MQSeries Integrator V2 - Message display, test and performance utilities*

Testing the exercises

This SupportPac is a GUI based program that assists in the testing of WebSphere MQ Integrator applications. It can display messages in a variety of formats, including XML, and COBOL copybook representations. It can read data from and write data to files and WebSphere MQ queues.

- The WebSphere MQ Explorer

  This program allows you to view the contents of messages on a queue. WebSphere MQ Explorer displays a tree view of the queue managers and their resources. Select the queue manager that hosts your broker, and look for the queue or queues you are interested in.

  Although you cannot use WebSphere MQ Explorer to put messages to a queue for testing all these scenarios, because it cannot use a file as input, it is useful for other actions. For example, creating queues.

Testing the exercises

Chapter 1. Introduction to messaging

The last few years have seen an increasing growth in the market for the consolidation of applications or hardware. The reasons for this can vary from the merger and acquisition of companies to the need for a company to integrate disparate or legacy software to work with new applications on various hardware platforms or operating systems. As technology progresses, the need for integration of systems becomes greater due to companies looking to keep costs down through linking systems together to make a complete solution.

As an example, a customer purchases a computer from a store. The sale through the till triggers a series of messages that are routed through financial systems for accounting purposes and to warehousing systems to order a new computer for the store. If the warehouse does not have any in stock, the manufacturing process logs the need for a computer to be built and ensures that all the items needed to build it are in stock or are ordered.

You can see from this relatively simple example how complex the process can become in linking all the systems together to create the complete solution. From this need, a number of companies developed software to allow applications to communicate with each other using messages.

WebSphere MQ

The WebSphere MQ family of products from IBM enables companies to develop program-to-program communications. Applications can now communicate by writing and retrieving application-specific data (messages) to or from queues, without having a private, dedicated, logical connection that must be developed and maintained by programmers.

Messages are the means by which the data is arranged, addressed, and sent between applications. Applications no longer need to talk directly to each other, but can place a message on a WebSphere MQ queue. Applications now have the ability to run asynchronously. This is where the sending application places a message on a queue and continues with its own processing and does not wait for a response from the receiving application. The retrieving application gets the message in its own time and might not even be running at the time the message was placed. An example of this is where an application is started at a specific time for handling the messages as a batch process.

The sending application does not need to know where the receiving application is. This is handled by a system administrator who configures the queue manager to ensure the message is routed for the next application. This allows the application to be on the same system, another system, even one running a different operating system, and routing and handling of the message is transparent to the sending application.

Administrators now have the ability to move applications from one system to another: by changing the parameters and definitions within the queue manager, the messages can be routed to the new system. This removes the costly need for any programming changes to any of the applications. This message and queuing process became a part of what has become known as middleware.

WebSphere MQ Integrator

Just passing the data from one application to another, particularly where they are on different machines and operating systems, left companies with the problem where diverse applications need to understand the data that is being processed. This often needed separate software to be developed to ensure that applications processing shared data were effectively speaking the same language.

WebSphere MQ Integrator is an application that has been written to enable this translation to be done for the application, removing the need for the often difficult process of updating legacy software. Applications continue to send the message to a queue in the standard way, but a *message broker* is now monitoring the queue. It retrieves the message and processes it through a *message flow*, placing it on an outgoing queue for the next application to process.

A message flow is a general purpose, reusable application. It is made up of a sequence of message processing nodes or subflows. This allows a single processing node or subflow to be used in multiple instances where the processing is common. An example of this is error handling.

This flow contains a sequence of operations that are performed on the message by a series of message processing nodes. These can:

- Transform it in terms of structure, layout, or code tables. An example of this would be from ASCII (American Standard Code for Information Interchange) to EBCDIC (Extended Binary Coded Decimal Interchange Code) formats
- Route it dependant on message content
- Deliver copies to multiple destinations
- Modify or add to the content of a message, for example adding a date or time stamp
- Read or update a database dependant on message content
- Perform virtually any operation using custom nodes

For a given application scenario, the message flow describes all the possible outcomes of processing for that message. Using the earlier example, this includes routing additional messages to an audit application for a financial transaction, sending a message to the warehouse for restocking, or handling an invalid message.

WebSphere MQ Integrator includes a range of message processing nodes called primitives that provide most of the basic functions such as input (for example: MQInput) to the message flow and output (MQOutput) from the flow.

Message flow nodes provide the individual processing steps that make up a message flow, and each node defines a single operation on a message. You can think of each node, including primitives, as a reusable component in an integration library. A node is joined to its neighbor by connectors, and it is this combination of nodes and connectors that make up a message flow.

On completion of the message flow, the flow can be assigned for execution to one or more brokers for the processing of messages.

High level overview

WebSphere MQ Integrator comprises the following two integrated components:

- The administrative and modeling environment and this consists of the Control Center and Configuration Manager. Here the message flows and message sets are modeled and deployed to the brokers for execution.
- The broker, where the message flows execute.

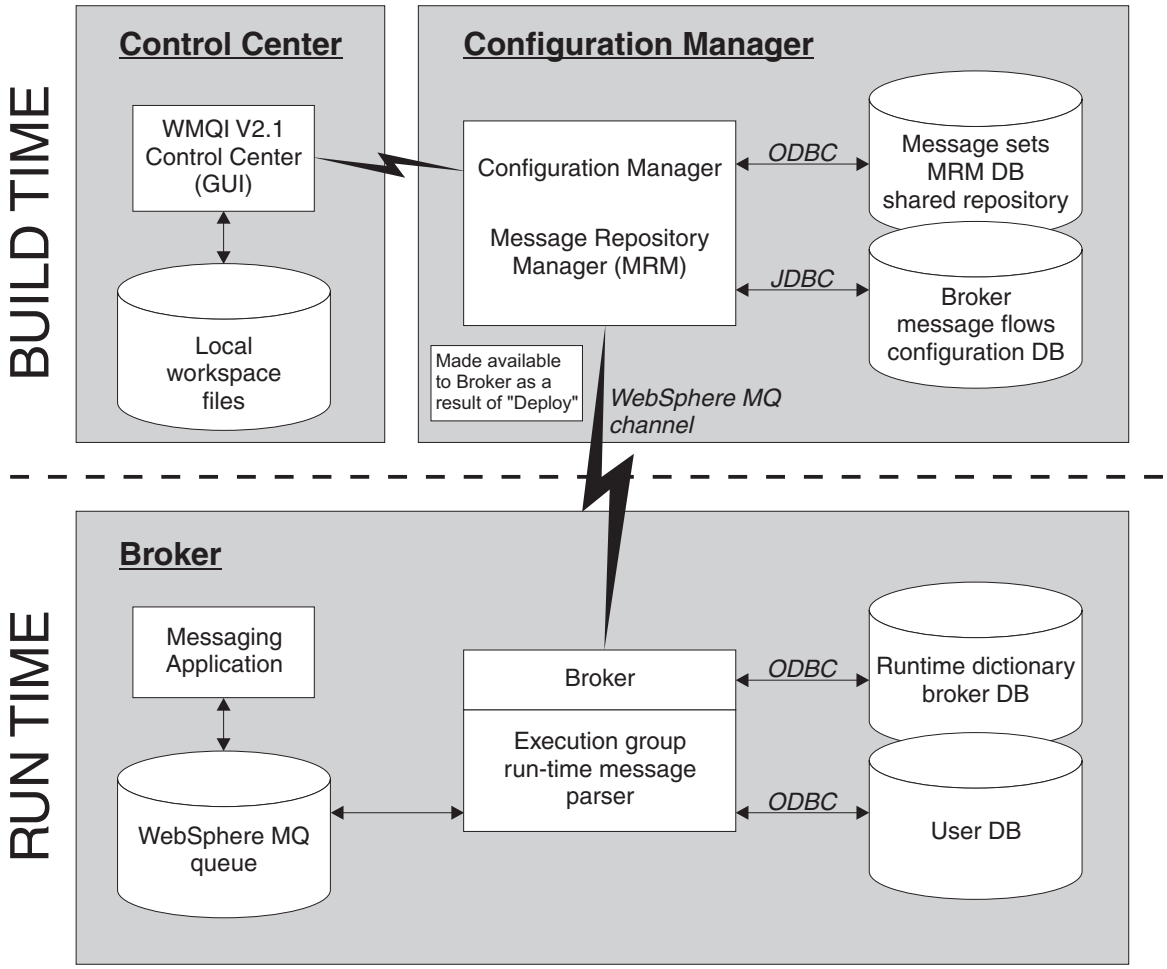

Figure 1. WebSphere MQ Integrator Overview

Configuration Manager and Control Center

The Configuration Manager and the Control Center form the buildtime environment.

The Configuration Manager is the central component of the WebSphere MQ Integrator environment and is required to manage the broker domain. The Configuration Manager provides a service to the other components in the broker domain, giving them configuration updates in response to activities that you initiate from the Control Center. It serves three main functions:

- It maintains the database tables that provide a central record of the broker domain components in the configuration repository
- It manages the initialization and deployment of brokers and message processing operations in response to actions that you initiate through the Control Center

- It checks the authority of defined user IDs to initiate those actions

The *Control Center* is the graphical development and administrative tool that interacts with the Configuration Manager. Using the Control Center, you can:

- Configure and control the broker domain
- Create, manipulate, and deploy broker domain resources. Example of these are message flows and message sets for the broker
- Monitor and manage the operational state of a broker

Message Broker

The message broker exists on the operating system where the message flows are to execute and are the runtime environment. The broker has three main functions:

- Manage the Administrative Agent and execution groups as they are deployed. It also restarts failed processes to increase resilience.
- Manage configuration changes through updates to the broker database tables. It also monitors the brokers input queues for Control Center requests and returns messages indicating success or failure.
- Manage the execution groups and their message flows.

Summary

Using the Control Center, Configuration Manager, and the broker, WebSphere MQ Integrator provides you with the framework that supports the supplied, basic functions, along with plug-in enhancements, to enable the rapid construction and modification of business processing rules that are applied to messages within a messaging system.

The Control Center and Configuration Manager form the build environment and are where messages are modeled, routed and transformed according to the business rules that the user has defined.

The message broker is the runtime environment and are where these rules are applied and the messages are processed.

For further details of data processing using WebSphere MQ Integrator see:

WebSphere MQ Integrator Introduction and Planning

WebSphere MQ Integrator Working with Messages

WebSphere MQ Integrator Using the Control Center

Chapter 2. Message processing concepts

To understand how the MRM handles the data that it receives in the form of messages placed on a WebSphere MQ queue, it is necessary to understand how the data is received and broken down into its separate components.

Data that an application sends to a WebSphere MQ queue has control information attached to it. This includes routing and descriptive information about the message and how it should be handled by WebSphere MQ. The combination of this control information and the application data is what forms the message. For the purpose of this concepts overview, reference is made to the application data only.

Data interpretation

When an application exchanges data with another application, it is essential that both interpret the data in the same way. If one of the applications interprets all or part of the message in a different way, it might process the data incorrectly, produce wrong results, and perhaps pass invalid data to a further application. An example would be a date being in the position an application is expecting a price. There needs to be clear understanding of not only the layout of the data, but the type of data each section represents.

Look at the following:

```
1 New Court
```

This looks like an address, but it can be broken down and interpreted in a number of different ways. Here are some of them:

- `'1 New Court'`: where the whole item is a text field
- `'1'` and `' New Court'`: where `'1'` is one text field and the remainder is another
- `'1 '` and `'New Court'`: where `'1 '` is one text field and the remainder is another (note: the difference with the one above is that the space has moved from the beginning of the second field to the end of the first field)
- `'1'` and `' New Court'`: where `'1'` is a binary field of 49 (Hex 31 and displaying as an ASCII 1) referring to house number 49 and the remainder is a text field
- `'1'`, `'New'`, and `'Court'` are all separate text fields that are separated by a space delimiter
- X'31204E657720436F757274' data that just happens to display as `1 New Court`!

These are just a few of the many examples of how one short block of data can be interpreted. This clearly demonstrates that unless the applications are interpreting the data in the same way, they are not able to share it.

Physical format

To enable an application to interpret the incoming message, the physical format of the data has to be defined. Using the above example, it has now been changed to read:

```
X'0001''New Court '
```

Here is a full description of the data: it comprises of two fields (elements). The first is a two byte field containing a binary number. The second field is a 10 byte field containing text characters. In a C header file this would be something like:

```
struct _Address_Line_1
     {
         short   HouseNumber;
         char    StreetName [10];
     } Address_Line_1;
```

You could use a Cobol copybook to define it the same way. What you are doing is describing the physical format of the data, not its content. The physical format of the data consists of such information as field or element size, and the type of data (for example binary, character, or hexadecimal) that it holds. In whatever format the data is physically held, it must be described in a way that is clear and unambiguous to any of the applications that will make use of it.

If all the applications understand and recognize the physical layout of the data, it is necessary only to pass the data between the applications. Passing the data from one application to another can be handled by applications such as WebSphere MQ and the message queueing process described earlier. The data requires no further processing other than the sending application placing the message on a queue and the receiving application retrieving it from a queue.

Logical format

If communicating, applications can require the same data, but in a different physical format. For example, an application could be processing data using a C data structure and has to pass this data on to an application that processes the data in extensible markup language (XML). If multiple applications require this information, each application is likely to require the data in a different format. The easiest way to handle this transformation is put the data into a common logical format from where each of the physical formats can be mapped.

This logical model of the data is a description of the data that is devoid of its physical representation, and is independent of the platform and the way that the message is physically constructed. You can view the logical model as the business meaning of the data.

For example, the letter 'A' is still the letter 'A' regardless of whether the physical format of the data is in ASCII (X'40') or EBCDIC (X'C1'). They are just different physical manifestations of the same business data.

If two applications were processing common data, one with the data in a C data structure and the other in XML, it would be necessary to write an application to convert this data from one physical format to the other. If data needs to be returned to the original application, the ability to convert from XML to the C data structure will be required. If a third application requires this data, such as SWIFT (a TDS format), it is possible the following conversions could be required:

- C data structure to XML
- XML to C data structure
- C data structure to TDS
- TDS to C data structure
- XML to TDS
- TDS to XML

Adding a third physical format has required another four conversion processes and if you add a fourth physical format, this new format would need to be mapped to each of the other existing physical formats. This is a difficult and time consuming task.

If the data received is translated into a logical format, the mapping of the data to and from any of the related physical formats becomes much easier and maintainability is greatly improved. Using a common logical format ensures that when another application's physical format is added, that physical format only needs mapping to the existing logical format. This application will now be able to communicate with any other application that has already been mapped to the logical format.

When the data is in this logical model, it can be processed in any way that the controlling application wishes to handle it. The simplest would be to move the data from the logical model to a different physical model. This is where the receiving application is expecting the information in a different order or format than the sending application is sending it in.

The application could need a date and time stamp in the message. This could be added to the information in the logical model and sent on as a new message to the next application. Perhaps one of the fields needs updating with pricing information. The process could interrogate the data in the logical model for the item, interrogate a database for its price, multiply it by the number of items in the data and add this information as a total cost to the logical model. This is now sent on to the next application for further processing.

If you process the data in a logical format, it means that whatever processing needs to be done, the process is only coded once and is carried out on the business data in the logical model.

The data for any new application only needs its physical format to be mapped to the current logical one to allow its data to be processed in the same way as an existing application that has already been mapped. If the physical data does not quite map to the current logical one, it could be pre-processed to match that layout reducing the need for coding for each message that is received.

You can see from this approach that there is a great flexibility in using the middleware to develop solutions for linking applications together.

Logical format

Chapter 3. Introduction to the MRM

The Message Repository Manager (MRM) is a component of WebSphere MQ Integrator that allows messages to be modeled in the Control Center, and have their definitions stored in a central message repository. This model is the logical format discussed in Chapter 2, "Message processing concepts" on page 5.

The MRM also provides a parser that works in the broker to parse incoming messages, and a message writer to construct outgoing messages that conform to the message models you have created.

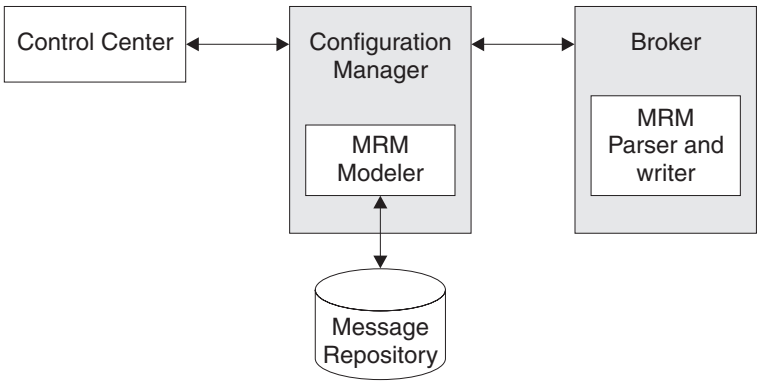

Figure 2. MRM processes in the Control Center, the Configuration Manager, and the broker

The Control Center interface to the MRM modeler (within the Configuration Manager) can be used to define the messages that WebSphere MQ Integrator is to handle. The Control Center works with the Configuration Manager to provide limited checks to ensure the model is valid, and to store the model in the message repository. A message set (containing a set of one or more related messages) can be assigned to a broker. The definitions of this set are retrieved and deployed to the broker in the form of a runtime dictionary, and describe the message model associated with the message set.

When a message is received by the broker, it is parsed. If it is identified as an MRM message, the broker involves the MRM and this interprets the data using the runtime dictionary, and produces a tree structure to represent the message. This structure is the broker's view of the logical message model. It is from this structure that the message can be manipulated to add, change, or transform the layout of the data in the message. This structure consists of some or all of the Message tree, LocalEnvironment tree, Environment tree, and ExceptionList tree. This structure and the trees created are discussed in detail in the *WebSphere MQ Integrator Working with Messages* book.

The runtime dictionary is also used by the MRM message writer to construct the output message, mapping its structure to the output physical message format required by the next application. The output message might be a copy of, or derived from, or bear no resemblance to, the input message.

The logical message model

The logical message model is built up of at least the following entities:

Message set
A message set is a collection of one or more related message definitions. Message sets can be assigned to the broker. When it is assigned to the broker and deployed, it becomes a runtime dictionary in the broker database.

Message
A message definition is a depiction of the information that your applications want to exchange (the actual message). Logically it has one or more element definitions that represent different kinds of business information.

Element

An element is a piece of business information. It is always based on a type and its contents give meaning to the type.

For example: an element with contents such as *Downing Street* or *Baker Street*, might have an element definition called *Street Name* that is of type STRING. The name and type are some of the properties that define the element.

When an element is created, it can have one or more element values associated with it. This value defines the way an the element will be constrained. The element value might define a default value for the element, or perhaps set the maximum length of the element. Each element value defines a single value for an element and one or more element values can be associated with each element.

The value constraints are:

- Minimum inclusive value
- Maximum inclusive value
- Minimum length
- Maximum length
- Enumeration
- Default value
- Null permitted
- Scale
- Datetime template
- Pattern

For further information on element values see the *WebSphere MQ Integrator Working with Messages* book.

Type There are two kinds of types, simple types and compound types.

- Simple types are predefined as BINARY, BOOLEAN, DATETIME, DECIMAL, FLOAT, INTEGER, and STRING.
- Compound types are those that you have defined yourself. These contain a group of elements along the lines of a C structure, that create a substructure within a message. A compound type can also contain other compound types.

This list only covers the minimum set of entities that are required to build a valid logical model that you can deploy to the broker. Further information on entities can be found in the *WebSphere MQ Integrator Working with Messages* book.

The buildtime and runtime environments

As stated earlier, the MRM can be looked at in terms of having a buildtime environment and a runtime environment. These terms are only used in this SupportPac to help with the understanding of the MRM: they are not used within the main documentation for WebSphere MQ Integrator.

Buildtime environment

The buildtime environment is analogous to the environment that a software developer writes and builds their code in. It consists of such things as code, libraries, macros, and the ability to build their code into a working program and test it. The MRM is where the systems administrator performs the equivalent functions for such things as message sets, elements, and types.

The buildtime environment is where:

- You save configuration data in the message repository
- You build messages in the Control Center
  - including the logical message format
  - including the physical message format
- You design the message manipulation in the Control Center with message nodes and flows. These are stored in the message repository
- You import messages structures to and export messages structures from the message repository
- Message sets and message flows are assigned to the broker
- You deploy such things as the message sets and message flows to the broker. Messages sets are deployed to the broker as a runtime dictionary and the message flows become libraries that are invoked by the broker to process an incoming message
- You control the management of messages through:
  - Locking
  - Versioning
  - Migration

Runtime environment

The runtime environment is analogous to the environment that the programs created by software developers are run in. It is where the system receives a message on a queue, the broker retrieves it from the queue, parses it, and does any necessary processing, and finally puts it back on to a queue for the next application. It consists of:

- Message parsers
- Runtime dictionaries
- Physical format descriptors
- Message flows

The message repository

The message repository database that you create is where the metadata that represents the logical message models is kept. This is also where the layers of additional data, used for mapping the logical model to the physical model, are stored. The definitions are made up of reusable components and are organized into message sets.

Creating message models

You create an MRM message definition within, and as a member of, a message set. A message set is an organizational grouping, and includes the definitions of one or more related messages, often those used by a single application. You construct each message using a set of building blocks called message components, some of these are supplied by WebSphere MQ Integrator (the simple types) and some of these you define using the Control Center (the compound types).

You can select from the following options to:

- Define your logical message model through the definition of elements and types.
- Import a message model from an external source to create the logical message model. Most legacy applications have their data defined in some way. Examples of this are a C header or Cobol copy book. The MRM provides a series of importers that you can use to take an existing definition and produce the appropriate entities in the MRM. A message set can be imported from another message set that has been exported from the same, or another, message repository.

When you have defined a message set, you must assign it to the brokers that need access to the message definitions. You do this using the Assignments view in the Control Center. When you deploy the message set to the broker, the MRM constructs a message dictionary from each message set and sends this information to each of the brokers you have assigned it to.

How the broker processes a message

When you have created the message sets, message flows, and all necessary definitions, you must assign and deploy them to the broker. The broker processes message data by:

- Receiving the raw data. The data is received on to a WebSphere MQ queue that the broker is monitoring.
- Parsing the data into the logical format. In the buildtime environment, the physical layout of the data is defined and deployed to the broker. Based on the physical layout, the MRM parses the data it has received in to the logical format that the broker processes. A message in the MRM logical format can be handled by all the IBM-supplied message processing nodes except the New Era of Networks nodes.
- Processing the data. Any necessary processing is done to the logical format of the data within the message flow nodes that have been defined and deployed by the Control Center.
- Writing (parsing) the data to the physical format. Following the processing, the data must be put back into a physical format for passing on to the next application.
- Sending on the processed data. Here the data, back in its raw state again, is placed on a queue for the next application to pick up and process.

How the broker processes a message

Through the definition and mapping of physical models to logical models, the processing of the data and mapping the data back to the same or a different physical model is what makes the MRM such a powerful and flexible application development tool.

Chapter 4. Importing C or Cobol data structures into the MRM

In this chapter you will look into how to import a C data structure to create a logical model and to add a custom wire format (CWF) physical format definition to the logical model. You will do this using a series of exercises and will be using the file PaxData1.h that is included in the SupportPac and shown in Appendix A, "Example C header files" on page 87.

The process for importing the equivalent Cobol data structure is virtually identical to that used to import a C data structure. The PaxData1.cpy file (shown in Appendix B, "Example Cobol copybook files" on page 91) is included with this SupportPac and you can import this file instead of the PaxData1.h file.

In this chapter you will look at:

- "Reviewing assumptions for this exercise"
- "Creating a message set"
- "Adding physical format layers" on page 16
- "Importing the data structure" on page 17
- "Creating a logical message" on page 19

Reviewing assumptions for this exercise

In addition to the "General assumptions" on page xii, the following apply:

- You have the default queue manager running
- You have the Configuration Manager running
- You have the default broker running
- You have started the Control Center
- You have access to the files in Appendix A, "Example C header files" on page 87 or Appendix B, "Example Cobol copybook files" on page 91
- If you are importing a Cobol copybook you have reviewed "C header files and Cobol copybooks" on page xiii

If you not have used the default names, you will need to substitute the names that you have used.

Creating a message set

Before you can import a data structure into the MRM, you must define a message set to import it into. Message sets are built using the Control Center. Use the steps below to create the message set that you will import your formats into.

1. In the Control Center, select the **Message Sets** tab.
2. Right-click **Message Sets** in the left pane.
3. Select **Create —> Message Set**.
4. A **Create a new Message Set** dialog is displayed. Enter a name for the message set, and leave the remaining settings to assume their default values. The name used here is MRMP_MS1
5. Click **Finish**. This creates the message set and can now be populated with messages.

In Figure 3 you can see the field **Identifier** at the bottom of the right pane. This message set identifier is automatically generated when the message set is created. The number that you see in Figure 3 (DPQ898C072001) cannot be changed. The one generated when you create yours will almost certainly be different.

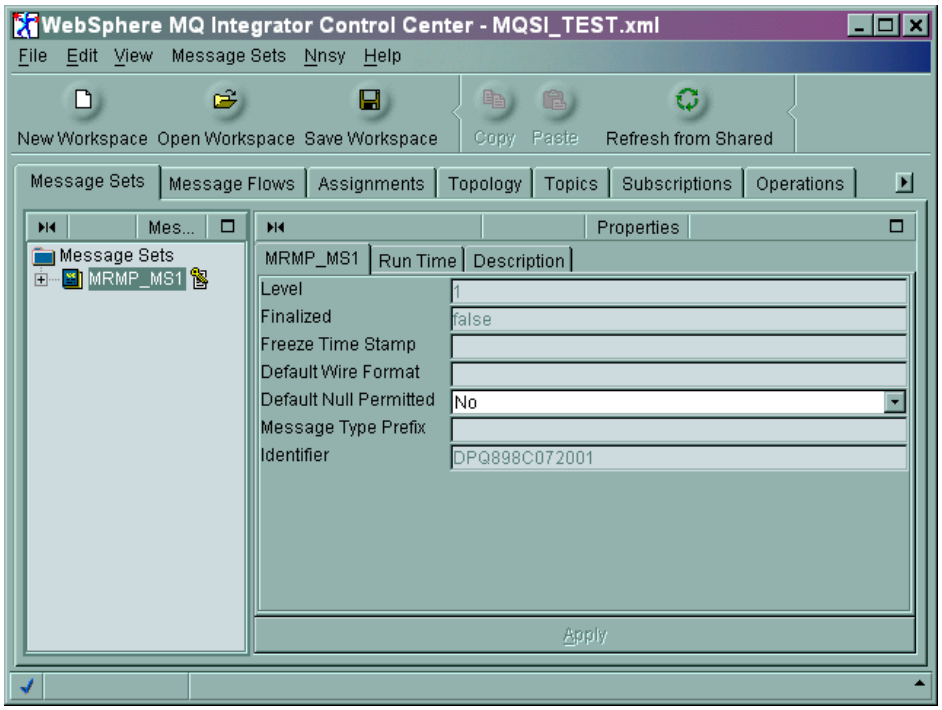

Figure 3. The Control Center on completion of Message Set creation

Adding physical format layers

You must add a physical format layer to the message set to allow the broker to map the data that is received to the logical format. When you import a C or Cobol data structure, the physical layer to add is a custom wire format (CWF) layer. You can add a physical format layer before or after importing the data structure if you are creating a logical layer. It is **recommended** that you add this layer before importing a data structure.

If you add a CWF physical layer before you import a structure, the Control Center ensures that most elements and element values are correctly set. If you import a data structure before adding the CWF physical layer, the Control Center is unable to populate properties that do not yet exist.

For example, if you import a C structure before you add a CWF physical layer:

- An UNSIGNED CHAR maps to BINARY and the element value for the length is set to 0.

  You might not notice this until you try to either check in the object or deploy the object to the broker after it has been checked in, and an error message is generated. Figure 4 on page 17 is an example of the type of error message generated. Read the message to understand the error that has been generated and investigate and correct it.

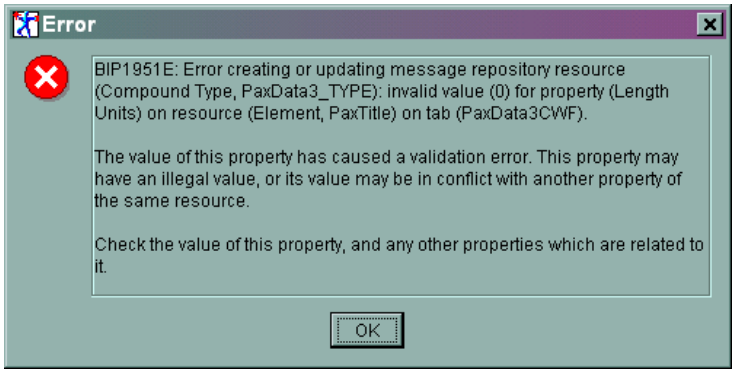

Figure 4. Typical error message generated on deploying to broker.

- A SHORT type maps to INTEGER, but the element value for its length is set to 4 for the element, but the C structure should have an integer length of 2.
- Repeating elements might only have values for the number of repeats set in some of the required fields and not all of them. If this occurs the incoming message is not parsed correctly.

To add a CWF physical format layer:

1. In the Control Center, select the **Message Sets** tab.
2. Right-click the message set (MRMP_MS1) you are adding the physical format to, and select **Check Out**.
3. Right-click the message set (MRMP_MS1) again and select **Add -> Physical Format... -> Custom Wire Format...**. This brings up the **Add a Custom Wire Format** dialog.
4. Enter the name for the physical format, for example CWF or PaxData1CWF.
5. Click **Finish**.

Importing the data structure

You now need to create the components in the message set that define the structure of the message to the MRM. In this example, rather than create them all manually, you will import a C header file or Cobol copybook that the MRM will use to define these components for you.

You have added the required message set and the physical format layer prior to importing a data structure. To import the data structure:

1. In the Control Center, select the **Message Sets** tab.
2. Select one of the following options:
   - For importing a C header file.

     Right-click the message set that you are to import into (for example MRMP_MS1)

     Select **Import to message set -> C...**. (You can also do this by selecting the message set that you wish to import to (for example MRMP_MS1) and selecting **Message Sets -> Import to message set -> C...** from the menu bar.)

     The **C Language Importer** dialog is displayed.

     The name of the C header file to use is **PaxData1.h**.
   - For importing a Cobol copybook.

     Right-click the message set that you are to import into (for example MRMP_MS1)

Select **Import to message set -> Cobol...**. (You can also do this by selecting the message set that you wish to import to (for example MRMP_MS1) and selecting **Message Sets -> Import to message set -> Cobol...** from the menu bar.)

The **Cobol Importer** dialog is displayed.

The name of the Cobol copybook to use is **PaxData1.cpy**.

Note: You should note that all labels from the data structure will be imported into the MRM in uppercase. You will need to make the appropriate substitutions for these labels in the following exercises.

3. Either type the full pathname and filename (PaxData1.h or PaxData1.cpy) of the source file in the **Import Source File** field, or use the **Browse** button to search for and select the file that you are to import. You can generate a report only at this point by selecting the **Report Only** check box. This tells you if there are any errors in the header file but does not create any resources in the Control Center. The report is also generated when the data is imported. See Appendix G, "Example CWF import report file" on page 105 for an example of a report file.
4. Click **Finish**.

   A dialog is displayed indicating if the operation was successful. It gives the name of the report file that it has generated and the location of this file. It also states if any user operation is required.
5. Select **OK** to continue.

When a data structure is imported, the MRM parses the source file and isolates the data structure definitions. From this information, it creates the logical definitions that correspond to the source file data structures and sets the appropriate wire format (CWF) properties that define the mapping between the logical definitions that are being created in the message set and the physical format of the original message.

Importing a data structure into a message set creates the compound types, the elements, and the element values that are associated with the data structure. Although these are created and added to the message repository, they do not display in the Control Center until you have added them to your workspace. This is because there might be a large number of elements, types, and messages and this allows you to add only those items to the workspace that you require to work with.

The report that is generated (see step 4 above) describes all the definitions that have been created, and includes information about any errors or conflicts within those definitions. By selecting the **report only** check box (see step 5 above) you can generate this report without making any changes to the message set. You are recommended to use this option if you are unsure of the validity of the data descriptions that you are importing.

If you added the CWF physical layer after importing the data structure, you will see the following lines at the beginning of the report:

```
Warning: The Custom Wire Format Plugin is not installed and registered>>
>> for the project.*
  No Custom Wire Format values will be set as a result
```

*The line has been split at >> for the purposes of displaying in this document.

Creating a logical message

Before you can use the data structure that you have just imported, you need to create the message for it:

1. In the Control Center, select the **Message Sets** tab.
2. Right-click the message set (MRMP_MS1) that you have imported into.

   Select **Create -> Message...**. This will open the **Create a new message** dialog.
3. Give the message a **Name** (PaxData1_Msg) and an **Identifier** (PaxData1ID).
4. Use the **Type** drop-down selector to list the available message types. The data structures that you have imported will be listed there. Select **PaxData1Msg_TYPE**.
5. Click **Finish**.
6. From the menu select **File -> Check In -> All (Saved to Shared)**.

The message has now been added to the message set and checked in. By expanding the MSMP_MS1 message set (click the **+** to the left of it) and expanding the Messages folder within that message set structure, you will see PaxData1_Msg listed. See Figure 5 for an illustration of this.

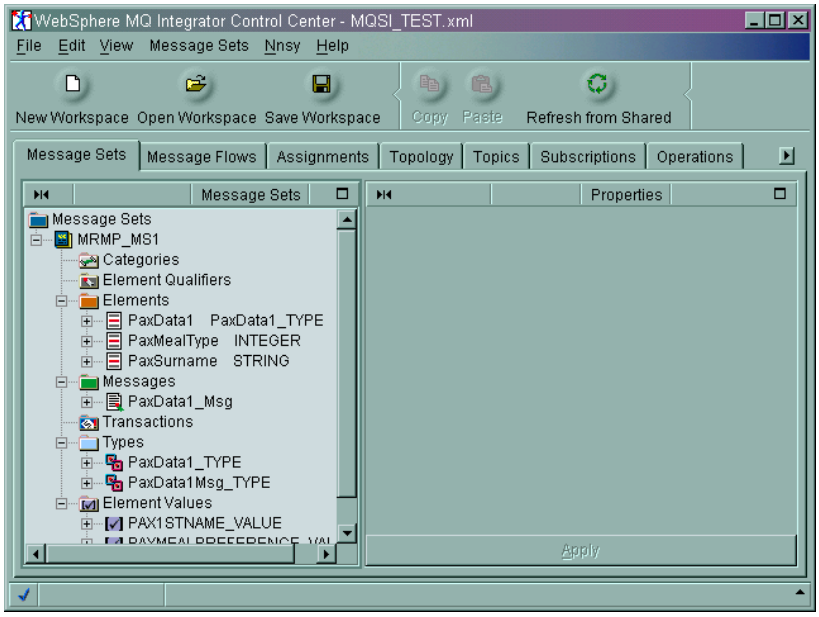

Figure 5. The Control Center after adding the PaxData1 C header to it.

Chapter 5. Creating and testing a basic message flow

In the previous chapter you created a message set. Message sets and message flows must not be confused with each other. A message set is the definition of the data, a message flow is how that data is to be processed. They are independent of each other and a message set can be used with multiple message flows. This chapter will just address message flows and you will see how to use message sets and message flows together in a later chapter.

This chapter gives an understanding of creating and testing a basic message flow. The message flow that you will create performs no transformation or changes to the message. It only routes the message from an input queue to an output queue and performs no processing of the message data.

It is important to understand the basic concepts in creating a message flow before moving on to message manipulation.

This chapter covers:

- "Reviewing assumptions for this exercise"
- "Creating the WebSphere MQ resources" on page 22
- "Creating a basic message flow" on page 22
- "Defining a broker in the domain" on page 24
- "Assigning the message flow to the execution group" on page 26
- "Deploying the message flow to the broker" on page 27
- "Testing the basic message flow" on page 29

Reviewing assumptions for this exercise

In addition to the "General assumptions" on page xii, the following apply:

- You have the default queue manager running
- You have the Configuration Manager running
- You have the default broker running
- You have the Control Center running
- You have the MQSeries Explorer running
- You have access to the files in Appendix C, "Example input message files" on page 95
- You have installed and understand the use of IH03 SupportPac.

If you have not used the default settings during the installation and setup of WebSphere MQ Integrator, you will need to substitute the names that you have used.

Creating the WebSphere MQ resources

This basic verification test requires two queues, one for input and the other for output. This section shows how to use the MQSeries Explorer to create these definitions.

1. In MQSeries Explorer, expand the tree under the queue manager MQSI_SAMPLE_QM. (Click the + to the left of the queue manager name.)
2. Right-click the queues folder and select **New -> Local Queue**.
3. Enter the queue name MQSI_INQ in the **Queue Name** field on the **Create Local Queue** dialog. You can accept the default values for all the other properties of the queue.
4. Click **OK**.
5. Repeat steps 3 and 4 to define the output queue MQSI_OUTQ. Again you can accept the default value for all other properties.

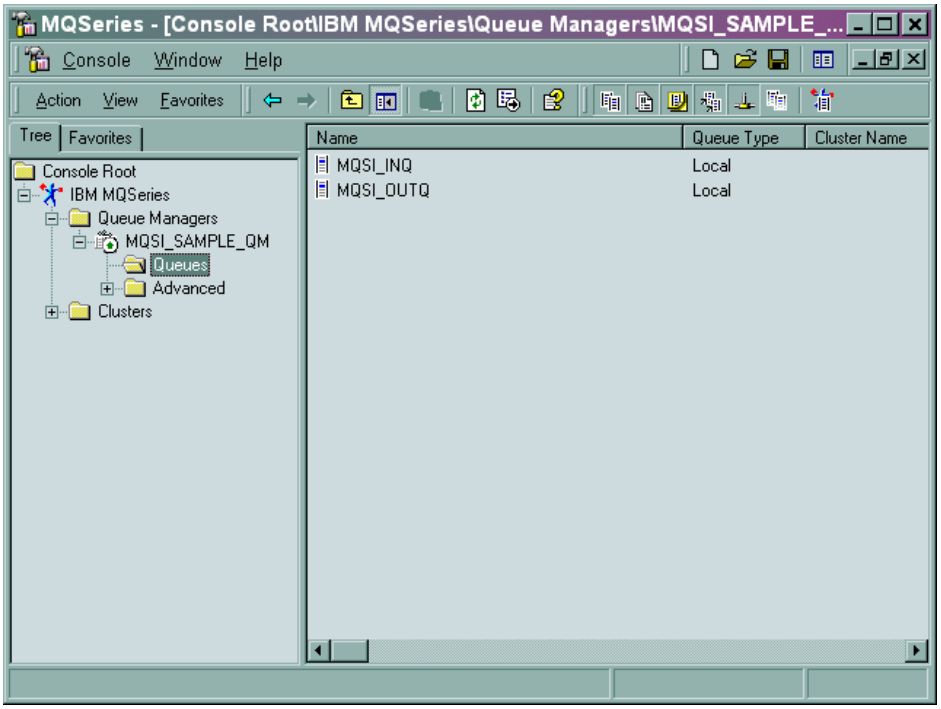

Figure 6. MQSeries Explorer after adding input and output queues

Creating a basic message flow

Now you will create the message flow that will process the messages you put to your input queue. The message flow is very basic: the only processing it does is to retrieve the message from the input queue and put it to the output queue.

1. In the Control Center, select the **Message Flows** tab.

   Right-click the **Message Flows** in the left pane and select **Create -> Message Flow**.

   Enter the name MQSI_TEST in the **Create new Message Flow** dialog.

   Click **Finish**. The new message flow appears in the tree view in the left pane.
2. Expand the IBM Primitives tree (click the + to the left of it) to display the supplied nodes.

Select the **MQSI_TEST** message flow in the left pane. Drag and drop an **MQInput** node from the list of primitives on to the right pane. This creates an MQInput node called MQInput1.

MQInput1 is the default name assigned to the MQInput node when it is added to the message flow. If you add a second MQInput node to the message flow it would become MQInput2, and so on. You can change the default names of any of the nodes you add, to names that are more appropriate to their use or context.

You can create message flows that have nodes in any place on the right pane. For ease of use and understanding, you are recommended to place nodes in a consistent order. By providing this level of consistency when you start to design message flows, you can make maintenance and transfer of ownership a much easier task. A suggestion is to have inputs come in from either the left or top of the pane and outputs to the right or bottom of the pane.

3. Right-click the MQInput node (MQInput1) in the right pane and select **Properties**.

   On the **Basic** tab, type the MQSeries input queue name of your input queue (MQSI_INQ).

   Click **OK**.

4. Drag and drop an **MQOutput** node from the list of primitives on to the right pane.

   Right-click the MQOutput node (MQOutput1) in the right pane and select **Properties**.

   On the **Basic** tab, type in the queue manager name (MQSI_SAMPLE_QM) and the queue name (MQSI_OUTQ) for the output queue.

   Click **OK**.

5. Right-click the **MQInput** node (MQInput1) and select **Connect -> Out**. This gives you a connector attached to your mouse pointer. Drag this to the **MQOutput** node (MQOutput1) and add the connector to it by left-clicking over the node. The connector attaches itself to the input terminal of the MQOutput node.

6. You have now completed your first message flow. Select **File -> Check In -> All (Save to Shared)**. This checks in all the resources to the configuration repository and saves the local copy of the workspace file. If you created a new workspace for this message flow, you will be prompted to give the workspace a name when you save it.

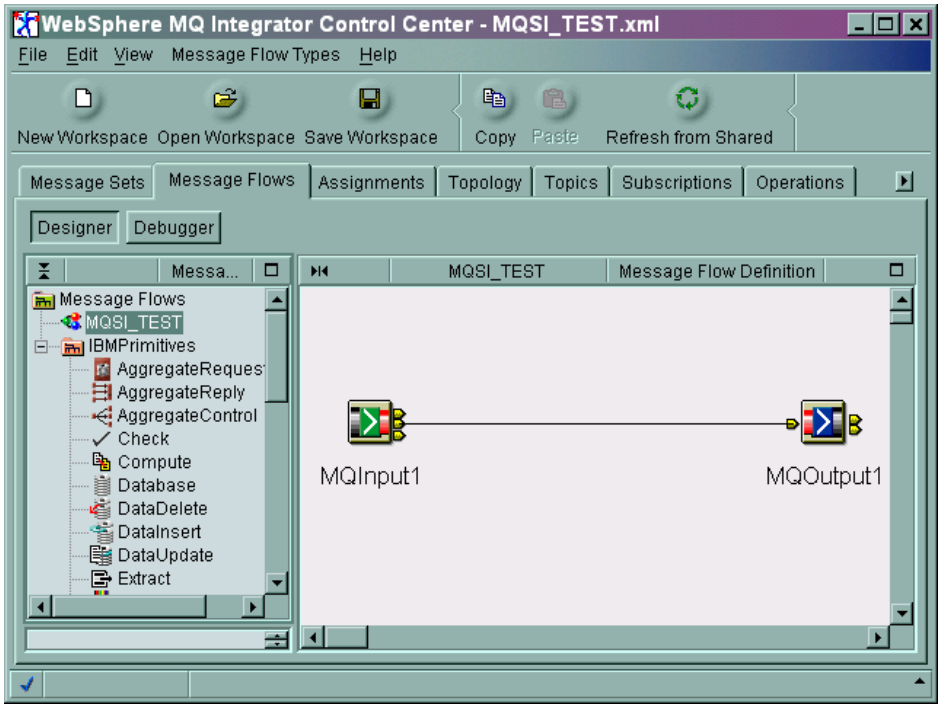

Figure 7. The Control Center after adding MQInput and MQOutput nodes

Defining a broker in the domain

If you have completed the installation process, you will have created a default broker. (If you have not done this, refer to the *WebSphere MQ Integrator Installation Guide* for details on creating a broker.) If you used the default broker name in the installation guide, this will be MQSI_SAMPLE_BROKER. Although the broker has been created it is possible that it has not been added to the workspace. You must add a reference to it within the workspace to be able to work with it:

1. In the Control Center, select the **Topology** tab.
2. In the left pane, right-click the **Topology** root and select **Check Out**. The topology has to be checked out to create the broker reference in the workspace. The checked out topology has a key symbol displayed next to it.
3. Right-click the **Topology** root again and select **Create -> Broker**.
4. To use the broker that was created during the installation, specify the same broker name (MQSI_SAMPLE_BROKER) and queue manager name (MQSI_SAMPLE_QM) that you used during that process.
5. Click **Finish**.
6. Right-click the **Topology** root and select **Check In**.

The Topology view now displays the broker that you have just added to the workspace.

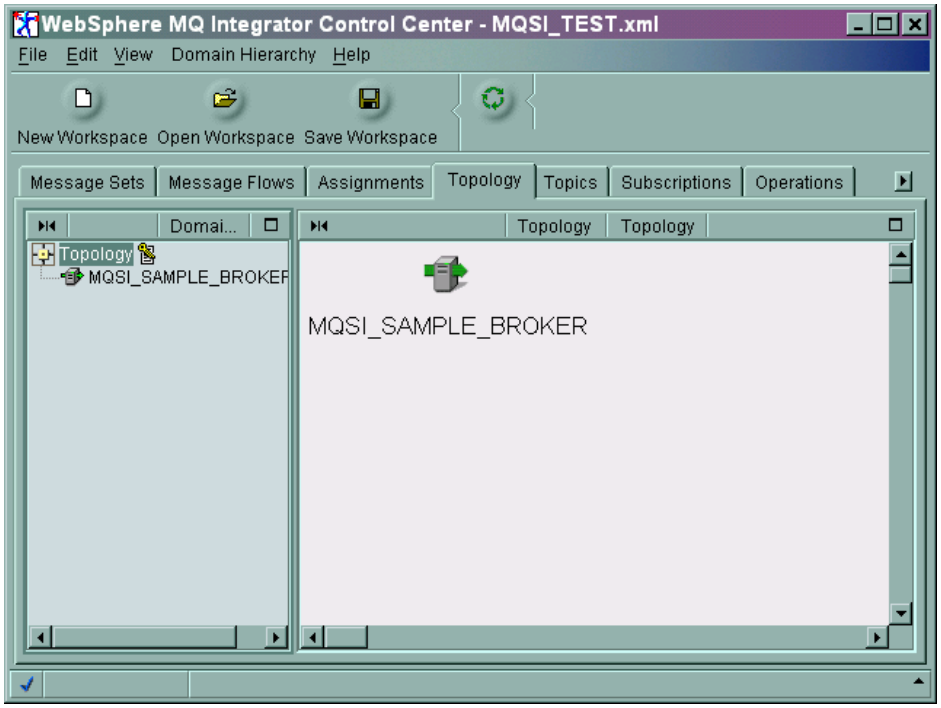

Figure 8. The Control Center after adding the broker: topology view

Assigning the message flow to the execution group

Having created a message flow, you need to define where you want to run it by assigning the flow to a execution group:

1. In the Control Center, select the **Assignments** tab.

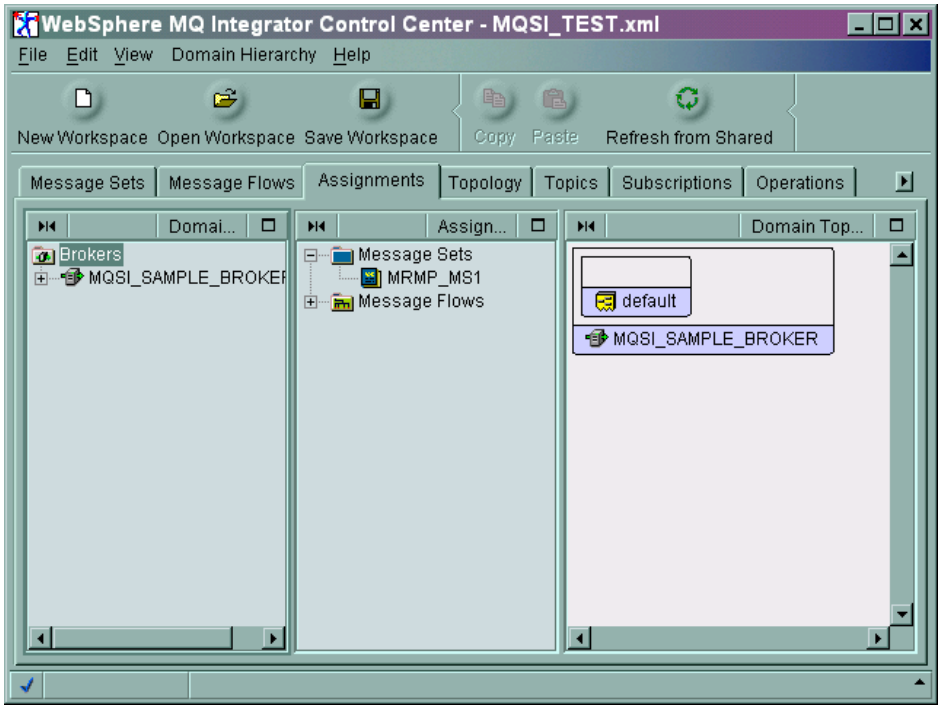

Figure 9. The Control Center after adding the broker: Assignments view

2. Expand the broker name (MQSI_SAMPLE_BROKER) in the left pane (click the + to the left of the name) to display the execution groups for the broker. The sample broker currently has one execution group (called **default**). This execution group is always created when you create a broker.
3. Right-click the **default** execution group and select **Check Out**. This locks the execution group for you. If this is the first time that you have worked with the execution group, you might get a message stating that this has never been checked in. If so, click **OK** and continue. When any object is checked out a key symbol appears to the right of the object name in the left pane.
4. Select your message flow (MQSI_TEST) and drag and drop it on the **default** execution group in the right pane, where you can see a graphical representation of the broker and default execution group. You can only drop a message flow on an execution group and not on the broker itself.
5. Right-click the **default** execution group in the left pane and select **Check In**. You might see a **Check in Confirmation** dialog showing the other elements (such as broker and topology) that also need to be checked in. Click **OK** and all necessary elements will be checked in, including those listed in the **Check In Confirmation** dialog.

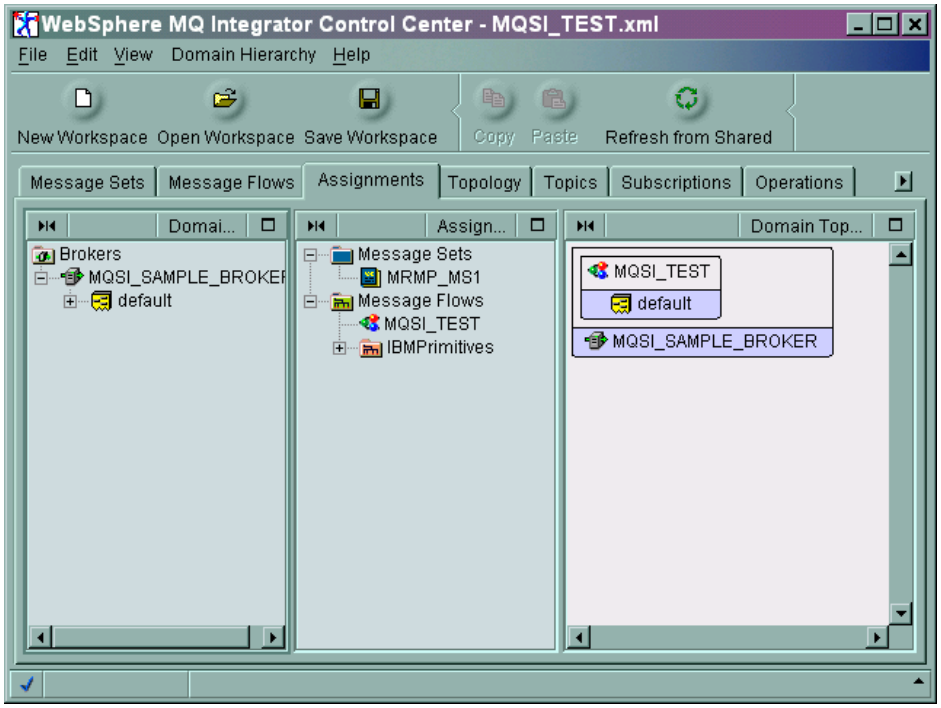

Figure 10. The Control Center after assigning the message flow to the broker

Deploying the message flow to the broker

An assignment makes a connection between a message flow and a broker within the Control Center. This connection instructs the Control Center to send the message flow to the broker when a deployment occurs. It is only when you make a deployment that the Configuration Manager updates the broker with the configuration stored in the configuration repository.

1. Before you can deploy any changes, all the updated resources must be checked in. If you have followed the instructions in this section, all the relevant resources are checked in. If you are in any doubt, you can check everything in by selecting **File -> Check In -> All (Saved to Shared)** from the menu.
2. In the Control Center, select the **Assignments** view.
3. Right-click the broker name (MQSI_SAMPLE_BROKER) in the left pane and select **Deploy -> Complete Assignments Configuration**. When the Configuration Manager receives this request from the Control Center, it sends messages to the broker to give it the updated information it needs to be able to support the new message flow.
4. Check the deploy by selecting the **Log** tab and clicking the refresh button (the green icon above the log pane). Check for error messages or success messages. (There might be a brief time delay before the messages appear.)

Deploy the message flow to the broker

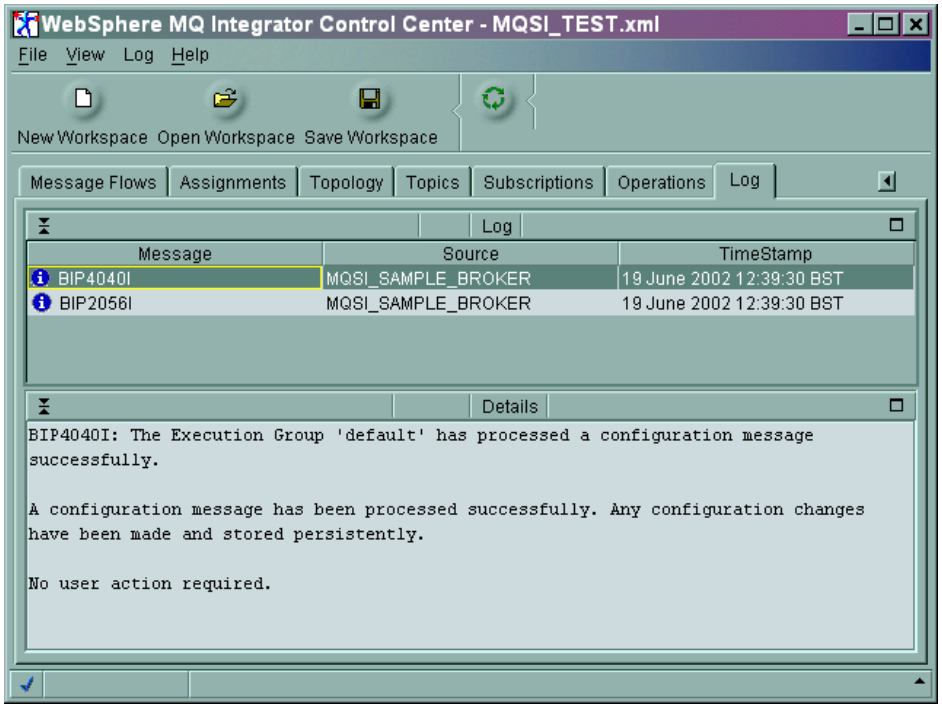

Figure 11. The Control Center Log view

Note: If you receive an error message, it will provide details of what you need to change to be able to complete this action.

5. View the deployed configuration graphically in the **Operations** view. When you refresh this view, the broker, execution group and message flow are displayed with green lights to show they are active.

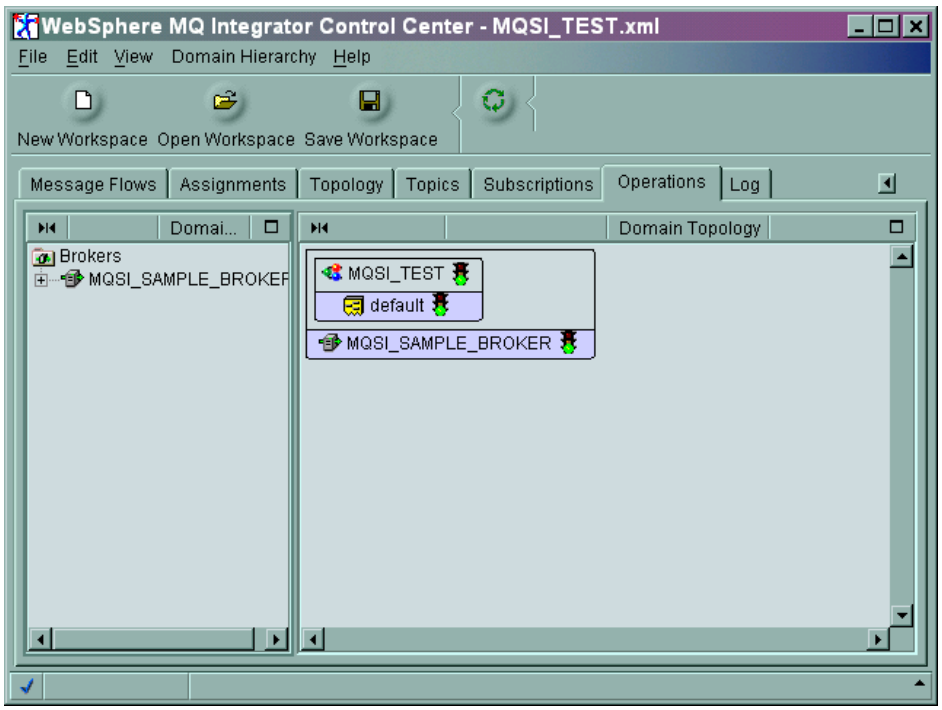

Figure 12. The Control Center Operations view

Testing the basic message flow

Before you can test a message flow, you must start the broker and then place a message on the input queue for the broker to process.

To test the message flow, you can use one of the following programs:

- MQSeries Explorer
- The SupportPac IH03

Using WebSphere MQ Explorer

You can use MQSeries Explorer to place a message on the input queue because all that is required is a text message. You cannot use MQSeries Explorer if the information in the message contains binary data, or if your messages are large.

This section shows how to use WebSphere MQ Explorer to test your message flow.

1. In WebSphere MQ Explorer, expand the queue manager and the queues folder.

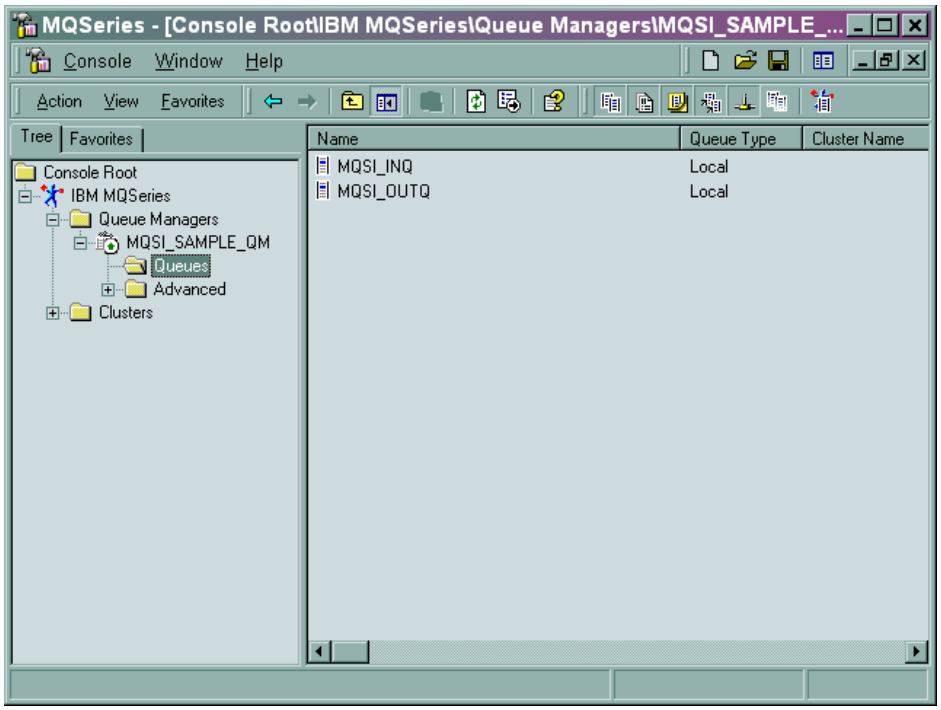

Figure 13. WebSphere MQ Explorer showing MQSI_SAMPLE_QM queue manager queues

2. Right-click the input queue (MQSI_INQ) and select **Put Test Message** from the menu.
3. Enter the text of your message (it can be any text string). Click **OK** to put the message on the input queue.

Figure 14. WebSphere MQ Explorer Put Test Message dialog

4. A confirmation dialog appears to confirm that the message has been placed on the queue. Click **OK**.
5. Right-click the output queue name (MQSI_OUTQ) and select **Browse Messages** from the menu.
6. Your message is displayed. The text of the message is displayed to the right of the screen. You might have to scroll the pane to the right to see it.

   If the message does not appear on the output queue, see "Error processing in a message flow" on page 53 for further information on error processing.

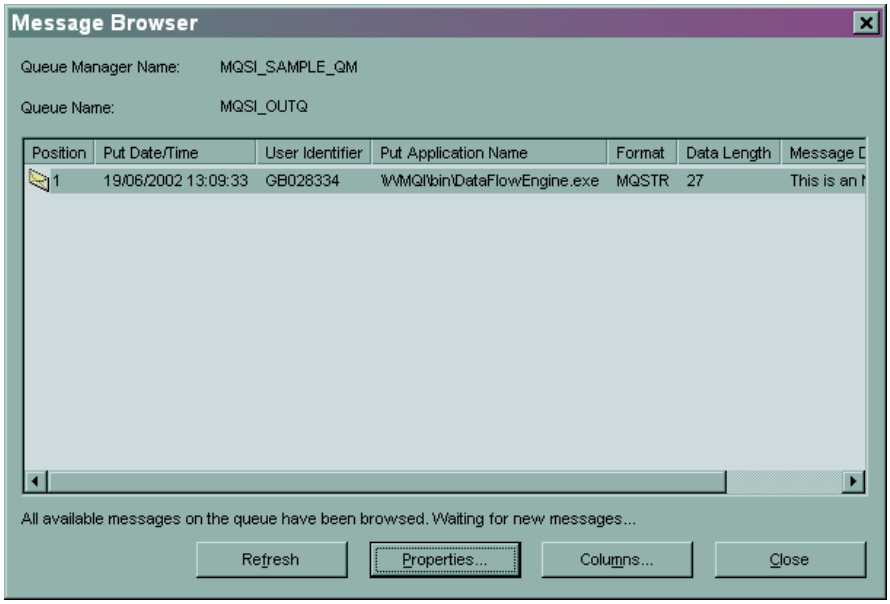

Figure 15. WebSphere MQ Explorer Message Browser dialog

7. From the Message Browser dialog, you can also select the message, click **Properties**, and select the **Data** tab to view the message data. The other tabs show further information about the message.

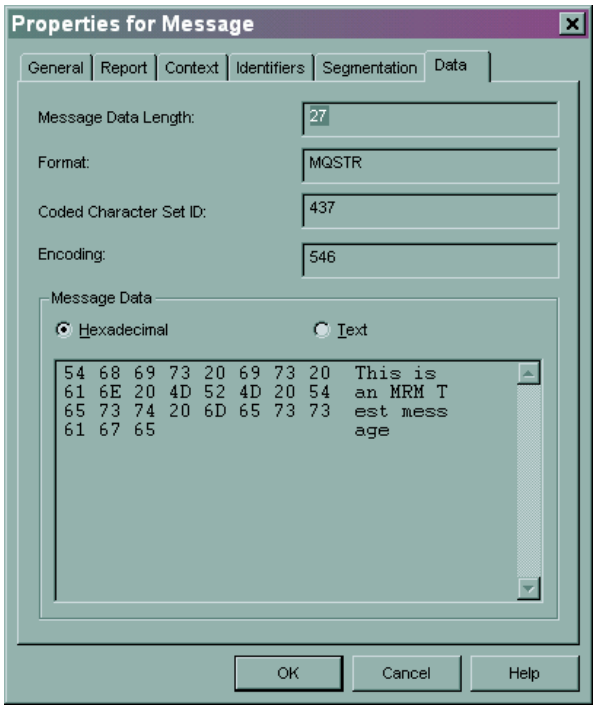

Figure 16. WebSphere MQ Explorer Properties for Message dialog

The test is complete.

Using IH03 SupportPac

You can also use the SupportPac IH03 (rfhutil.exe). This SupportPac reads a file (whatever the data) and places it on the queue of your choice. It also has the ability to read or browse messages from a queue. It is therefore, a flexible interface to your queues.

This section shows you how to us the SupportPac IH03 to test your message flows.

1. Start the **rfhutil** application. (This is the name of the application that comes as part of IH03).
2. Under **Queue Manager Name (to connect to)**, enter the name of the queue manager (MQSI_SAMPLE_QM).
3. Under **Queue Name**, enter the name of the input queue (MQSI_INQ) that you are placing the message on to.
4. Click **Read File**, and use the dialog to select the file **PaxData1.ipt** (supplied with this SupportPac) and click **Open**.

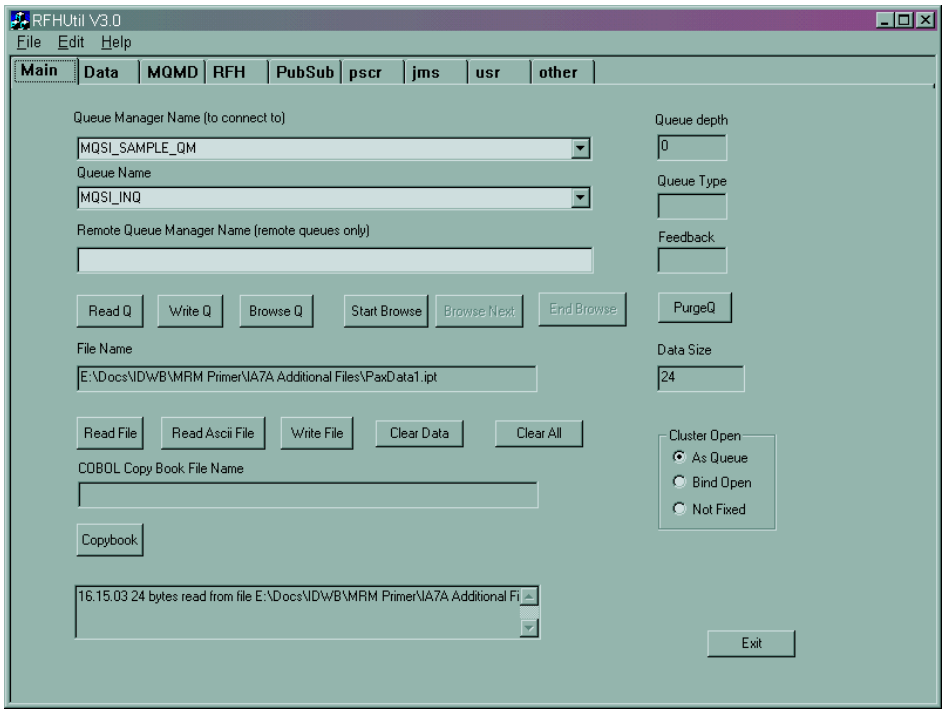

Figure 17. IH03 (rfhutil.exe) initial screen

5. A message confirming that the file has been opened appears at the bottom of the dialog. Select the **Data** tab at the top of the utility: this shows the data it has read from the file.

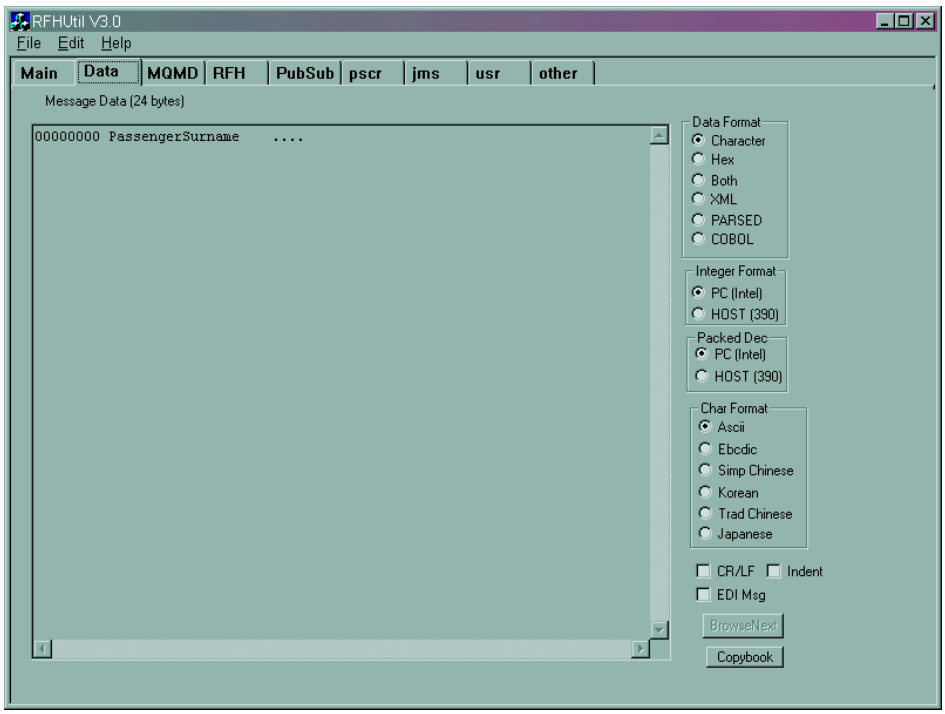

Figure 18. IH03 (rfhutil.exe): Data tab

6. Click **Write Q**. This places the data that has been read from the file onto the specified queue. A confirmation message (for example: 'Message sent to MQSI_INQ') will be displayed at the bottom of rfhutil dialog.
7. Click **Clear Data** to clear all the fields including that in the Data tag. Because the data that is being written to and read from queue is identical in this exercise, it could be difficult to see what has been retrieved from the queue. Clearing the data from the fields will make it clearer that, after reading the data, a message has actually been retrieved.
8. Change the queue name to the output queue (MQSI_OUTQ).
9. Click **Read Q**. This will retrieve the message that was put above and generate a message stating that a message has been read (for example: 'Message read from MQSI_OUTQ') at the bottom of the dialog.
10. Select the **Data** tag: the data shown in this instance will be identical to that when you select the Data tag in step 4 above. (See Figure 18 on page 32 for an example).

This test is complete.

Note: It is possible to select **Browse Q** in step 9 above. This reads the data from the message on the queue, but leaves the message on the output queue. For further details on rfhutil, read the associated documentation in IH03 SupportPac.

Using the IH03 SupportPac

Chapter 6. The Trace node

The aim of this chapter is to give an introduction to the Trace node. This chapter covers:

- "Reviewing assumptions for this exercise"
- "Use of the Trace node"
- "Adding a Trace node to a message flow" on page 36
- "Demonstrating trace" on page 37

Reviewing assumptions for this exercise

In addition to the "General assumptions" on page xii, the following apply:

- You have the default queue manager running
- You have the Configuration Manager running
- You have the default broker running
- You have the Control Center running
- You have created the message flow in Chapter 5, "Creating and testing a basic message flow" on page 21
- You have access to the files in Appendix C, "Example input message files" on page 95
- You have installed and understand the use of *IH03*.

If you have not used the default settings during the installation and setup of WebSphere MQ Integrator, you will need to substitute the names that you have used.

Use of the Trace node

The Trace node is one of primitive nodes that is supplied with WebSphere MQ Integrator. You can use this node during the development of a message flow to check that the behavior of a message flow is as expected, or to assist in the diagnosis of errors that could occur. A Trace node can be added at any point in the message flow and the data captured can incorporate text, message content, and date and time information.

You can use the Trace node to write out some or all of the contents of the Message tree, LocalEnvironment tree, Environment tree, and ExceptionList tree to file. This file can be a user file, a trace log, or a local error log. See Chapter 3, "Introduction to the MRM" on page 9 for further details on message trees.

Trace nodes can be invaluable in assessing and analyzing what is happening in a message flow, but this data gathering also adds an overhead to the message flow processing. If a high throughput is desired, do not leave Trace nodes in message flows or have trace switched on when you are satisfied that the message flows have been found to be working correctly. Trace nodes are independent of Trace and you should be aware that Trace nodes will still gather information even if user trace is inactive.

For more information on the Trace Node see the Control Center online help and the *WebSphere MQ Integrator Problem Determination Guide*.

Adding a Trace node to a message flow

This exercise will add a Trace node to the message flow that was built in Chapter 5, “Creating and testing a basic message flow” on page 21. If you are using a message flow other than this one, the principle remains the same.

If you are adding a Trace node to a new message flow that you are creating, you can do so in exactly the same way as you added the nodes to your first flow in “Creating a basic message flow” on page 22.

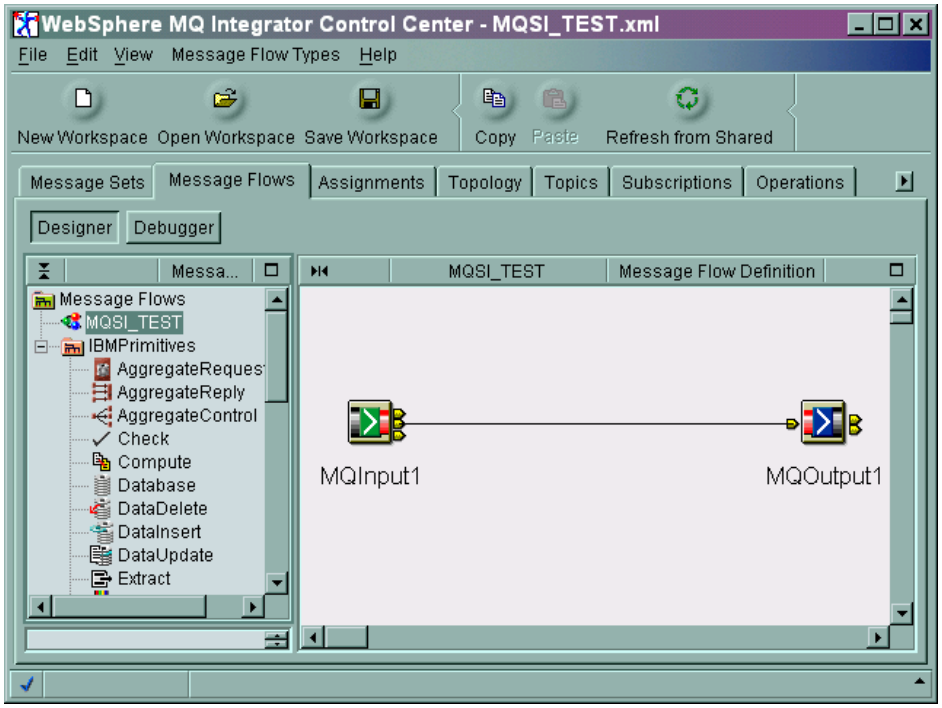

Figure 19. The Control Center and message flow to be modified

1. In the Control Center, select the **Message Flows** tab.
2. In the left pane right-click the message flow (MQSI_TEST) to be amended and select **Check Out**.
3. Right-click the connector line and select **Delete**. Click **Yes** to confirm the deletion.
4. Expand the IBMPrimitives in the left pane and scroll down to show the Trace node.
5. Select the Trace node and drag and drop it into the right pane between the MQInput1 node and the MQOutput1 node. This creates a Trace node with the default name Trace1.
6. Right-click the Trace node (Trace1) and select **Properties**.

   In the **Pattern** field type **${Root}**. This will make the Trace node (Trace1) trace the whole message tree passing through it, from the root downwards
7. Right-click the MQInput1 node and select **Connect -> Out**.

   Left-click the Trace node to make the connection from the MQInput1 node.
8. Right-click the Trace node and select **Connect -> Out**.

   Left-click the MQOutput1 node to make the connection from the Trace node.

9. From the **File** menu, select **Check In —> All (Saved to Shared)**.

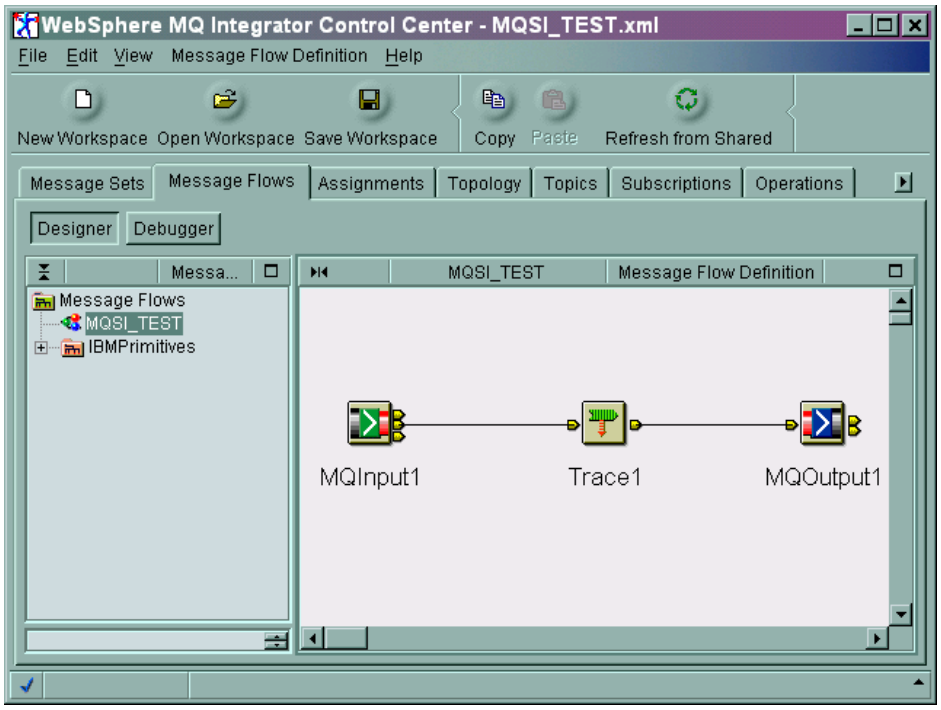

Figure 20. The Control Center showing the added Trace node

10. In the **Assignments** view, right-click the broker name (MQSI_SAMPLE_BROKER) in the left pane.

    Select **Deploy -> Complete Assignments Configuration**. When the Configuration Manager receives this request from the Control Center, it sends messages to the broker to give it the updated information it needs to be able to support the new message flow.

11. Check the deploy by selecting the **Log** tab and clicking the refresh button (the green icon above the log pane). Check for success messages. (There might be a brief time delay before the messages appear.) See Figure 11 on page 28.

12. View the deployed configuration graphically in the **Operations** view. Refresh this view and the broker, execution group and message flow are displayed with green lights to show they are active. See Figure 12 on page 28.

This has added a Trace node to the message flow and deployed it to the broker. The Trace node has been added with the default setting of user trace. The information that you request is written to the system generated user trace file in the log sub-directory. This combines the user trace information with the contents of the message tree.

Demonstrating trace

In this exercise you will use the trace nodes to create a log file and look at the data within that file.

For this exercise you will use the message flow (MQSI_TEST) that you added the Trace node to in “Adding a Trace node to a message flow” on page 36. (The traces that are shown in this chapter also had the input node updated as in “Adjusting the Input node properties for the message set” on page 46. This allows the input

node to interpret the incoming message correctly and the trace file shows this as the message has been correctly parsed. Further details of this are covered in Chapter 7, “Converting CWF to TDS” on page 41. The message placed on the queue to allow this is the one in the PaxData1.ipt file supplied with this SupportPac.)

1. Place a message on the input queue (MQInput1). Follow the instructions in either “Using WebSphere MQ Explorer” on page 29 or “Using IH03 SupportPac” on page 31.
2. This will create a file in the `log` sub-directory of the location where WebSphere MQ Integrator is installed (for example: `C:\IBM\WMQI\log`). The default file name is made up of the broker name, the broker UUID (Universal(ly) Unique Identifier), and a suffix of `userTrace.bin.` and a number (For example: `MQSI_SAMPLE_BROKER.46163417-ee00-0000-0080-df695815836b.userTrace.bin.1).` You are also able to give the log file a name that you specify. See the Trace node in the Control Center online help for information on how to do this.
3. The file has been recorded in a binary form and you will not be able to read it directly. The log file needs to be converted and formatted for you to read it.

   Open a system Command prompt and change to the log subdirectory. (For example: `cd c:\ibm\wmqi\log`).
4. Enter `mqsireadlog MQSI_SAMPLE_BROKER -u -e default -o mrmptrace.xml` where:
   - **MQSI_SAMPLE_BROKER** is the name of the broker you are using.
   - **default** is the name of the execution group being traced.
   - **mrmptrace.xml** is the name of the XML output file to be used and is created in the directory that the command is run from

   The **mqsireadlog** command takes the log file and converts it into an XML format. This can now be read using an XML file reader (an example would be Microsoft Internet Explorer). You can find further details on the **mqsireadlog** command in the *WebSphere MQ Integrator Administration Guide*. The file created can be seen in Appendix E, “.XML trace file” on page 99.

   The output files from this command are created within the directory the command is run from. You are recommended to make this the log subdirectory to keep all the trace files together.
5. If you want to use a text editor to view the trace contents, use the **mqsiformatlog** command. Enter `mqsiformatlog -i mrmptrace.xml -o formatmrmptrace.log`

   where:
   - **mrmptrace.xml** is the input file
   - **formatmrmptrace.log** is the output file and is created in the directory that the command is run from

   This can now be read by a text editor: you can see an example in Appendix F, “.log trace file” on page 101. This is an example of a trace where the input node has been modified to parse the incoming message. As each node processes the message, trace information is written to the log file. The trace can show:
   - The process of the message being received

- The message structure where the input node has the expected message format defined and the broker is able to parse the message. For example:

```
(0x1000021)MRM        = (
  (0x1000013)PaxData1 = (
    (0x300000B)PaxSurname  = 'PassengerSurname    '
    (0x300000B)PaxMealType = 1
  )
```

- The message that will be processed (although there is no processing).
- The output message from the message flow.

Demonstrating trace

Chapter 7. Converting CWF to TDS

This chapter will look at the processes needed to take an input message with a physical format of Custom Wire Format (CWF) such as a C data Structure and convert it to an output message with a physical format of Tagged Delimited String (TDS). Some of these procedures have been covered in earlier chapters and will only be referenced from this chapter.

This chapter covers:

- "Reviewing assumptions for this exercise"
- "Creating and assigning the message set"
- "Create a CWF to TDS message flow" on page 46
- "Testing the CWF to TDS message flow" on page 52
- "Error processing in a message flow" on page 53

Reviewing assumptions for this exercise

In addition to the "General assumptions" on page xii the following apply:

- You have the default queue manager running
- You have the Configuration Manager running
- You have the default broker running
- You have the Control Center running
- You have created the message set in Chapter 4, "Importing C or Cobol data structures into the MRM" on page 15
- You have created the message flow in Chapter 5, "Creating and testing a basic message flow" on page 21
- You have access to the files in Appendix A, "Example C header files" on page 87 or Appendix B, "Example Cobol copybook files" on page 91
- You have access to the files in Appendix C, "Example input message files" on page 95
- You have installed and understand the use of IH03
- If you are importing a Cobol copybook you have reviewed "C header files and Cobol copybooks" on page xiii

If you have not used the default settings during the installation and setup of WebSphere MQ Integrator, you will need to substitute the names that you have used.

Creating and assigning the message set

This section covers:

- "Creating the message set" on page 42
- "Adding physical format layers" on page 42
- "Importing the data structure and creating the associated message" on page 42
- "Mapping the TDS physical layer" on page 43
- "Delimiters" on page 45
- "Assigning a message set to a broker" on page 45

Creating the message set

Follow the procedure “Creating a message set” on page 15 to create the message set (MQSI_TEST) required for this exercise.

Adding physical format layers

To enable the broker to communicate with external applications, a runtime dictionary (created when the message set from “Creating the message set” is deployed to the broker) holds a physical description of the incoming and outgoing messages that need to be mapped to the logical format. You must add a physical layer for each different physical format that the broker will be handling. You are converting from CWF to TDS and the broker will need to interpret the CWF format message and write the TDS format message. You need to add both of these formats as physical layers to the message set.

Note: If you are using the message set created in Chapter 4, “Importing C or Cobol data structures into the MRM” on page 15, you can skip steps 3 to 5 below as the CWF layer has already been added.

To add CWF and TDS physical format layers:

1. In the Control Center, select the **Message Sets** tab.
2. Right-click the message set (MRMP_MS1) you are adding the physical formats to and select **Check Out**.
3. Right-click the message set (MRMP_MS1) you are adding the CWF physical format to and select **Add -> Physical Format... -> Custom Wire Format...**. This brings up the **Add a Custom Wire Format** dialog.
4. Enter the name for the physical format, for example CWF or PaxData1CWF.
5. Click **Finish**.
6. Right-click the message set (MRMP_MS1) you are adding the TDS physical format to and select **Add -> Physical Format... -> Tagged/Delimited Format...**. This brings up a **Add a Tagged/Delimited Format** dialog.
7. Enter the name for the physical format, for example TDS or PaxData1TDS.
8. Click **Finish**.
9. Right-click the message set and select **Check In**.

The physical layers have now been added and each of the properties panes for the message set has additional tabs of the names you gave the CWF and TDS layers.

You might not immediately see the additional tabs until you refresh or restart the Control Center.

Importing the data structure and creating the associated message

Note: If you are using the message set created in Chapter 4, “Importing C or Cobol data structures into the MRM” on page 15, you can skip this section because the data structure has been imported and the logical message has been added.

Importing the data structure and creating the associated message

You have created the message set and added the physical layers required for importing the data structure. To import and create the message:

1. Use the process "Importing the data structure" on page 17 to import the data structure. During this process use the PaxData1.h (or PaxData1.cpy for Cobol) file supplied with this SupportPac as the CWF file to import.
2. Use the process "Creating a logical message" on page 19 to create the required message.

On completion of these steps the Control Center will look as Figure 21.

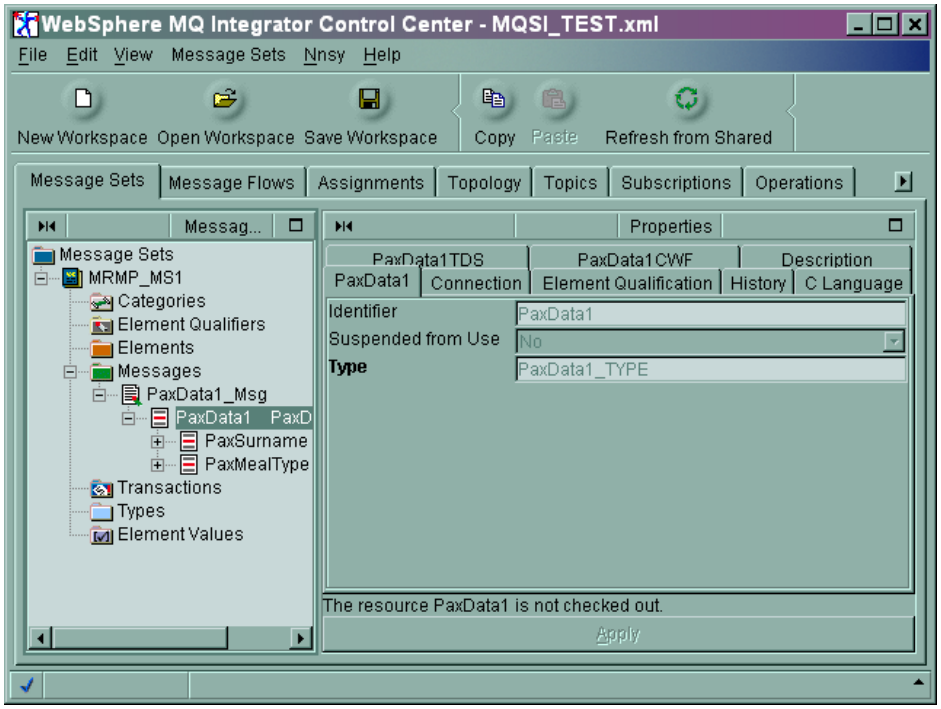

Figure 21. The Control Center after importing the message

Mapping the TDS physical layer

TDS is a very flexible way of defining a message. Each of the fields in the message could be of variable length with a delimiter, or could be of fixed length as for a CWF message. Unless you know that the default values that are set for the TDS layer are correct for the solution you are developing, these default values are likely to need updating to map correctly. The remainder of this section shows some typical changes that you might need to complete to ensure the physical and logical models map correctly.

If the physical layer is added after creating or importing the elements in the logical model, it is possible that not all the values are set. For further details see "Adding physical format layers" on page 16.

This exercise will generate a TDS format output message, where each of the elements is separated by a delimiter. This is achieved by setting the **Data Element Separation** property to **All Elements Delimited**. However, this causes a conflict with **Type Content** of **Open** and would prevent the message set from deploying.

Mapping the TDS physical layer

The **Type Content** property of the compound type (for example, PaxData1_TYPE) defaults to **Open**. This indicates a message or element of this compound type might contain those elements specified in the model, or any other additional element. For this exercise you will be working with messages that contain only those elements specified from the model (for example; a fixed number of elements) and so the **Type Content** property needs to be set to **Closed** to achieve this and to allow the message set to deploy.

The following exercise will update the message set properties so that:

- The message's compound type expects a fixed set of fields
- The elements retain a delimiter between each of the message fields

Use the following steps to make the above changes:

1. In the Control Center, select the **Message Sets** tab.
2. Expand the message set (MRMP_MS1) in the left pane by clicking on the **+** to the left of the message set name.
3. Right-click **Types** and select **Add to Workspace —> Compound Type...**. This will bring up the **Add an existing Compound Type** dialog.

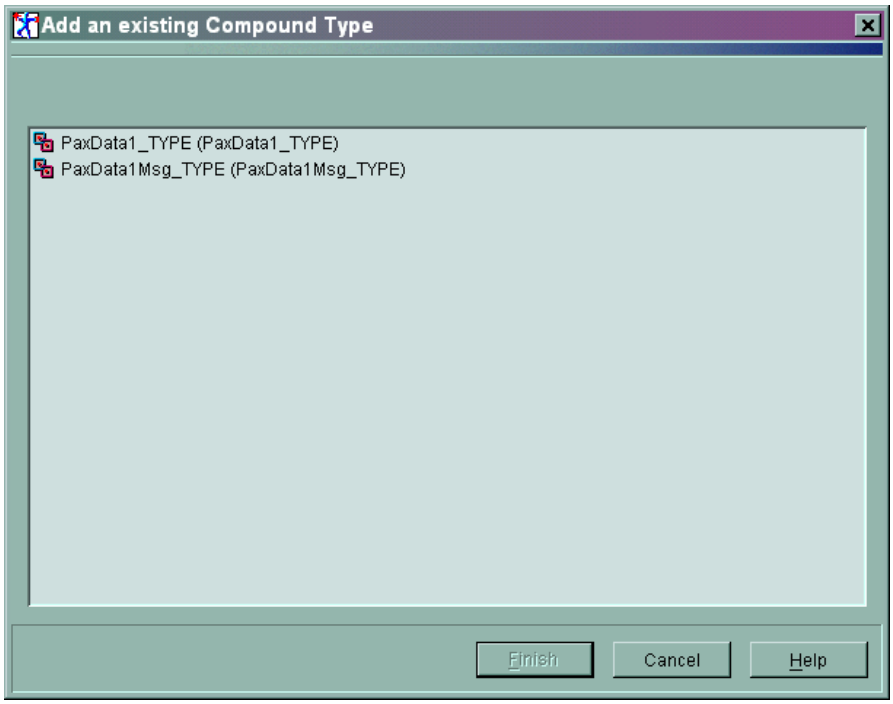

Figure 22. Add an existing Compound Type dialog

4. Select the **PaxData1_TYPE** and the **PaxData1Msg_TYPE**. These were created when the PaxData1.h file was imported, but were not added to the workspace at that time. This point was discussed in "Importing the data structure" on page 17 in Chapter 4, "Importing C or Cobol data structures into the MRM" on page 15.
5. Click **Finish**. These types have now been added and can be viewed by clicking the **+** to the left of **Types** in the left pane.
6. Right-click each of **PaxData1_TYPE** and **PaxData1Msg_TYPE** in the left pane and select **Check Out**.
7. Select **PaxData1_TYPE** type in the left pane.

8. In the right pane select the **PaxData1_TYPE** tab.
9. Ensure that **Type Composition** is set to **Ordered Set** and that **Type Content** is set to **Closed**.
10. In the right pane select the **PaxData1TDS** Tab.
11. In the **Data Element Separation** field select **All Elements Delimited**.
12. In the **Delimiter** field enter the character or characters that you will use as the field separator. This example uses the '|' character. You must choose characters (including hexadecimal) that do not appear in the data within the message. An example of a poor delimiter would be to use a space character, but the data in the message contains spaces within the text itself.
13. Repeat steps 7 to 12 for **PaxData1Msg_TYPE**.
14. From the menu select **File —> Check In —> All (Save to Shared)**.

Delimiters

Delimiters should not be confused with tags. Tags are text bracketted by '<' and '>' and are often referred to as delimiters. Tags are used in languages such as HTML and SGML to give structure and meaning to the text they contain. For example, text defines the word 'text' as a paragraph.

Delimiters are used to separate the fields or elements in a data stream and provide no form or structure to the data that they mark out.

Delimiters can be either a string of characters or hexadecimal data. Regardless of what you use to provide the break between each of the fields or elements in your data stream, it must be unique to that data stream. If you use a space (' ') to provide the break, this would cause serious problems if the data also contained spaces. The code that parses the data stream would not be able to differentiate between the space that was being used as a delimiter, and the space that was just part of the text.

This primer uses '|', '||', '|||', and '||||' as delimiters. This in itself can cause problems depending on how the parser is coded. For example if the parser is looking for a '|', it could interpret '||||' as four single '|' characters. If it was coded to look for the longer string first this problem would not occur. A better suggestion of delimiters could be '|01|', '|02|', '|03|', or '|04|'.

When designing the message structure you should spend time to consider what you will use as a delimiter. This is to ensure that the delimiter that is unique for the current data stream, as far as possible, remains unique as the contents of the data being transmitted are changed in the future.

Appendix K, "Delimiter examples for PaxData4 files" on page 119 uses the PaxData4.h and PaxData4.cpy files to show in more detail how delimiters have been used in this primer.

Assigning a message set to a broker

For the broker to be able to interpret and process the messages that it receives, it generally requires a runtime dictionary. This is created when you assign and deploy the message set to the broker. Use the following steps to add the message set created or modified in this chapter to the broker:

1. In the Control Center, select the **Assignments** tab.
2. In the left pane, right-click the broker (MQSI_SAMPLE_BROKER) that you are to add the message set to and select **Check Out**.

3. Expand the **Message Set** field in the center pane by clicking the **+** to the left of Message Sets.
4. Drag and drop the message set (MRMP_MS1) that you are to add from the center pane into the box representing the broker (MQSI_SAMPLE_BROKER) in the right pane.
5. From the menu select **File —> Check In —> All (Save to Shared)**.

On completion of these steps, the Control Center will look similar to the example in Figure 23 where the names shown are those that have been used in earlier procedures.

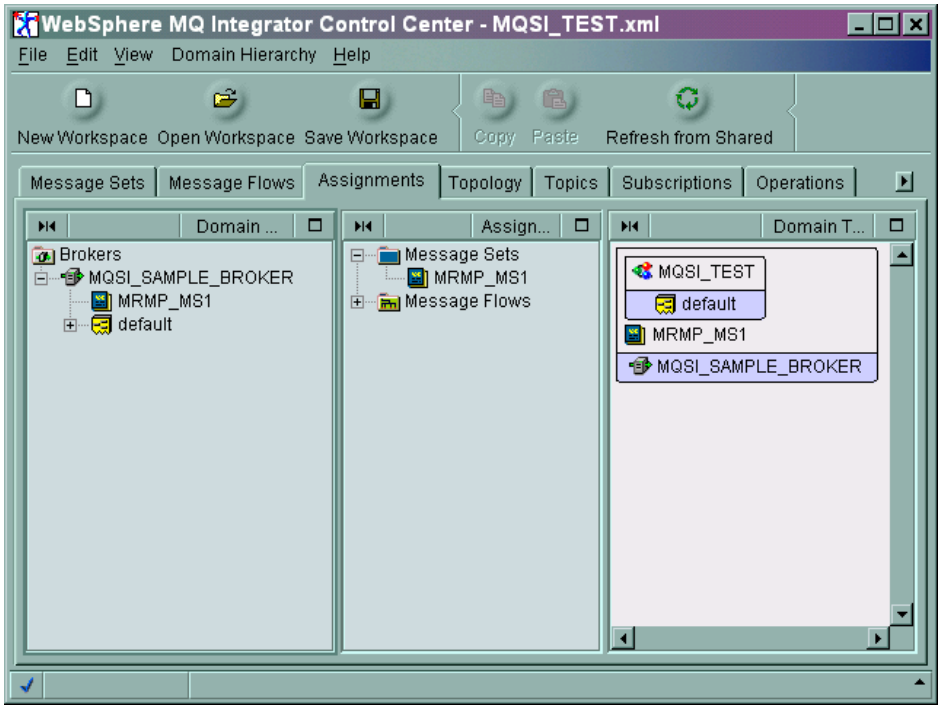

Figure 23. The Control Center after assigning a message set to the broker

Create a CWF to TDS message flow

The message set required to convert the message from CWF to TDS has been created and assigned to the broker in the section "Creating and assigning the message set" on page 41. In this section you will create the message flow to convert the incoming message from CWF format to TDS format. It covers:

- "Adjusting the Input node properties for the message set"
- "Adding a Compute node" on page 47
- "Setting the Compute node to convert CWF to TDS" on page 48
- "Defining the broker in the domain" on page 51
- "Assigning the message flow to the broker" on page 51
- "Saving all the changes" on page 51
- "Deploying the configuration to the broker" on page 52

Adjusting the Input node properties for the message set

When the broker receives a message it needs to know how to parse it. The broker first looks for an MQRFH or MQRFH2 header (An architected message header that

is used to provide metadata for the processing of a message). If this is not found, the broker checks to see if the input node specifies the parser. If the input node does not specify the parser, the broker uses the BLOB (binary large object) parser supplied with WebSphere MQ Integrator. A blob represents a block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one solid entity that cannot be interpreted.

You have defined how to interpret the incoming message in your message set (MRMP_MS1). You now need to specify this message set in the input node to allow the parser to interpret the message it will receive. This is because you are not setting an MQRFH or MQRFH2 header in the input message that defines how the message is to be parsed.

Use the following steps to alter the properties of the input node to specify the message set to use during parsing:

1. In the Control Center, select the **Message Flows** tab.
2. Select the message flow (MQSI_TEST) created in "Adding a Compute node".
3. Right-click the input node (MQInput1) and select **Properties**. This will bring up the Properties dialog.
4. Select the **Default** tab.

   For **Message Domain** select **MRM**.

   For **Message Set** select the identifier of the message set to be used. This identifier is allocated when the message set is created and is unlikely to the match the one used in the examples in this document.

   If there is more than one identifier in the drop-down section and you are not sure of the identifier in the message set you wish to use, select the **Message Sets** tab in the Control Center and select the name of the message set (MRMP_MS1) you are using. The identifier is displayed in the right pane under the tab that has the same name as the message set.

   For the **Message Type** enter the message identifier (PaxData1ID) for the message to be processed. (You can find this by selecting the **Message Set** tab and selecting the message you are processing in the left pane (PaxData1_Msg) and the **Identifier** is shown in the right pane.)

   For the **Message Format** select from the drop-down the format (PaxData1CWF) that has CWF in brackets next to it and that you defined in "Adding physical format layers" on page 42.
5. Click **OK**.

Adding a Compute node

This procedure gives an instructions on modifying an existing message flow. In many cases copying an existing message flow and updating it is a quicker way to complete a new flow than creating a new one. It is also possible to alter an existing message flow and deploy it to the broker to change the process that is currently running.

In this exercise, you will use a message flow that you created in an earlier exercise. If you have not done this earlier exercise, follow the procedure indicated to create it.

1. In the Control Center, select the **Message Flows** tab.
2. Create a message flow as shown in Chapter 5, "Creating and testing a basic message flow" on page 21 (MQSI_TEST).

3. If the message flow is not checked out, right-click the message flow (MQSI_TEST) in the left pane and select **Check Out**.
4. Right-click the connector between the input node (MQInput1) and the output node (MQOutput1) and select **Delete**.

   Click **Yes** to confirm the deletion.
5. Expand the IBMPrimitives in the left pane by clicking the **+** next to it.
6. Scroll down and select the Compute node. Drag and drop it between the input and output nodes in the right pane. This adds a Compute node called Compute1.
7. Right-click the input node (MQInput1) and select **Connect -> Out**.
8. Click the Compute node (Compute1). This adds a connector from the input node (MQInput1) to the Compute node (Compute1).
9. Right-click the Compute node (Compute1) and select **Connect -> Out**.
10. Click the output node (MQOutput1). This adds a connector from the Compute node (Compute1) to the output node (MQOutput1) node.

This creates the basic flow that is required, but the properties of the input node need to be updated to interpret the message and the Compute node properties need to be updated to convert the message.

Setting the Compute node to convert CWF to TDS

At this stage you have added a Compute node to the message flow, but it has not been configured to do any processing. If you deployed this message flow, it would not be able to process any messages. The Compute node needs to be 'programmed' so that it knows how to process the incoming message, and how you want the output message to be written.

The Compute node is a way of building a new message using a set of assignment statements. It works by always constructing a new message even if there is no change to the input message being processed. The simplest example of this is where the Compute node simply constructs an exact copy of the input message.

Introduction to the Compute node Properties dialog

The Compute node processes the input and output message by executing ESQL statements that allow you to determine the content and structure of the output message. You can enter these ESQL statements through the properties dialog.

Right-click the Compute node and select **Properties**: the properties dialog as shown in Figure 24 on page 49 is displayed.

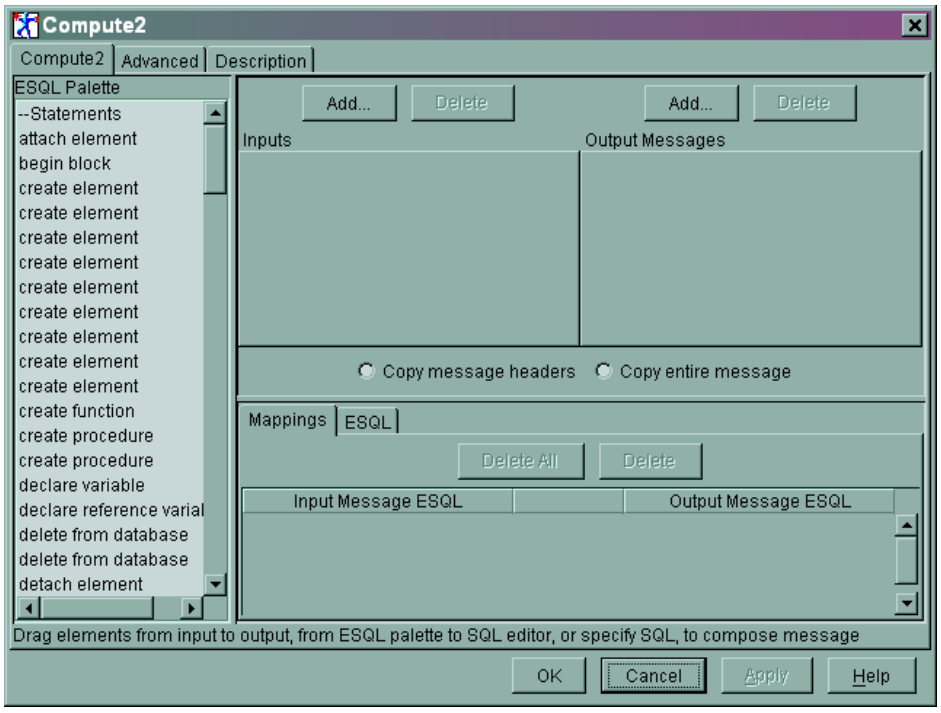

Figure 24. A Compute node properties dialog

When the properties dialog is opened, at the bottom of the dialog it displays the message:

```
Drag elements from input to output, from ESQL palette to SQL editor, >>
>>or specify SQL, to compose message*
```

* The line has been split at >> for the purposes of displaying in this document.

This statement informs you that you achieve a number of the operations that you want to complete by 'dragging and dropping' objects between the panes of the dialog.

The dialog has four panes.

- The left pane has predefined ESQL commands or structures that you drag and drop into the ESQL pane at the bottom. When you select an ESQL command in the left pane, the code it would generate is displayed at the bottom of the dialog. For example, the display might read:

  ```
  if <boolean-value> then <controlled-statements> end if;
  ```

  When the code has been dropped in to the ESQL pane (using the above example), it has place holders such as:

  ```
  <boolean-value>
  ```

  You must update these to reflect the fields or code you are developing.
- The top two panes are used for selecting the input and output logical messages that are being processed. These panes give you the ability to select a message or element in a message and drag and drop it into the ESQL pane creating the text necessary to address that object in the ESQL code.

  When you select **Add...** the **Add...** dialog will appear and you can specify the message set and message that you are to process.

(You can also specify a data source and table to process. See Control Center Help for further information.)

Selecting either of the radio buttons automatically generates the code that copies the message headers or the entire message to the output message (it can be neither).

Changes in the top panes generate ESQL code that can be viewed in the bottom pane.

- The bottom pane is generally used for viewing ESQL code that has been generated from the top and left panes, and for modifying the code.

For further information on this dialog, click **Help**.

Note: The panes at the top and to the left of the dialog, and the information in the mapping tab in the bottom pane, have no bearing on how the code in the Compute node will execute. These panes are only an aid in generating the ESQL code that the Compute node will process. **It is only the code that can be seen in the ESQL tab of the bottom pane that determines the behavior of the compute node.**

Setting the Compute node properties for CWF to TDS conversion

For any CWF to TDS conversion where only the format of the data is changing, there is no change to the logical representation of the message being processed, only to the physical format of the message that is being written out. There are two parts to this exercise:

- You will create an output message that is an exact copy of the input message.
- You will set the property for the output format to TDS.

Use the following steps for this exercise:

1. In the Control Center, select the **Message Flows** tab.
2. Select the message flow (MQSI_TEST) created in "Adding a Compute node" on page 47.
3. Right-click the Compute node (Compute1) and select **Properties**. This will bring up the **Properties** dialog.
4. Select the **Copy entire message** radio button

Click the **ESQL** tab in the bottom pane. You will see that the following ESQL statement has been added:

```
SET OutputRoot = InputRoot;
```

This code was automatically generated by following step 4 above and copies the input message to the output message.

The following line also appears:

```
-- Enter SQL below this line.  SQL above this line might be regenerated, >>
>>causing any modifications to be lost.
```

This line and any code above this line will be automatically generated. If you change the code above the line and now change one of the parameters that created the code in the first place, you lose the changes that you have made. To demonstrate this, select the **Copy message headers** radio button and see how the ESQL code changes. Ensure that you select the **Copy entire message** radio button again to revert to the correct code.

Setting the Compute node properties for CWF to TDS conversion

Use the following to complete the ESQL code:

1. Select the ESQL tab in the bottom pane of the Compute node properties dialog.
2. Move the cursor until it is after the last line of text (the one starting `-- Enter SQL below this line).`
3. Enter the following text at the beginning of the next line:

   ```
   SET OutputRoot.Properties.MessageFormat = 'TDS';
   ```

   This sets a property that tells the parser to write the output bitstream in TDS format (not CWF).

   On completion of your changes the code will look as follows:

   ```
   SET OutputRoot = InputRoot;
   -- Enter SQL below this line.  SQL above this line might be regenerated,>>
   >> causing any modifications to be lost.
   SET OutputRoot.Properties.MessageFormat = 'TDS';
   ```

4. Click **OK**.

You have now created ESQL code that will copy the message that the compute node receives to the message that it will propagate out. The code also changes a property in the compute node's output message that declares the physical format that the message flow's output message is to be written in.

Defining the broker in the domain

If you have already completed the work in Chapter 5, "Creating and testing a basic message flow" on page 21 you can skip this section because your broker is already defined in the domain.

Use the process "Defining a broker in the domain" on page 24 from Chapter 5, "Creating and testing a basic message flow" on page 21 to ensure that the broker that was created during the installation process has been added to the domain you are working in.

Assigning the message flow to the broker

If you have already completed the work in Chapter 5, "Creating and testing a basic message flow" on page 21 you can skip this section because you have already assigned the message flow to the broker.

Use the process "Assigning the message flow to the execution group" on page 26 from Chapter 5, "Creating and testing a basic message flow" on page 21 to ensure that the message flow has been added to the broker.

Saving all the changes

Before you deploy to the broker, you must save all the changes that have been made. If you do not save any of the changes, you will see an error message that lists the resources you have not checked in.

From the Control Center menu select **File —> Check In —> All (Saved to Shared)**.

Deploying the configuration to the broker

Use the process “Deploying the message flow to the broker” on page 27 to ensure that all the changes made to the message sets and the message flows are deployed to the broker for its use in processing messages.

Testing the CWF to TDS message flow

In Chapter 5, “Creating and testing a basic message flow” on page 21, almost any message would have been acceptable to test the message flow. This was due to the parser treating the incoming data as a BLOB (Binary Large Object) and not requiring it to be interpreted.

For the message flow that has been created in this chapter, the input message must be formatted to match the data structure that was imported and the CWF layer that this created. If the message fails to meet this format, the broker will not be able to interpret it and process it correctly.

If you have imported the PaxData1.h or PaxData1.cpy file in “Creating and assigning the message set” on page 41, you can use the PaxData1.ipt file supplied with this SupportPac that contains a message that is formatted to test the flow. This contains a mix of ASCII text and binary data and can be seen in Appendix C, “Example input message files” on page 95. If you have used another data structure and created a different CWF layer, you will need to create an appropriate message file for you to test the message flow with.

This process uses the SupportPac IH03 to test the message flow. Two copies of the utility are opened, one to put the messages on to queue and the second to read them.

It can be useful to have two copies of rfhutil running. When you are testing a range of messages, one copy of the utility can be used to select and place the messages on queue and the second copy can read the message from the output queue, without having to be altered.

If you prefer you can use a single copy of the utility, and just alter the queue name to read or write the messages.

The following steps can be used to test the message flow:

1. Start a copy of rfhutil (IH03). Set the **Queue Manager Name (to connect to)** to the name of the queue manager (MQSI_SAMPLE_QM) being used.
2. Set the **Queue Name** to the name of the input queue (MQSI_INQ).
3. Click **Read File** and select the message input file (PaxData1.ipt from this SupportPac) to use.
4. Click **Write Q** to place the message from the file on the input queue. A confirmation message `Message sent to MQSI_INQ` is displayed at the bottom of the utility.
5. Start a second copy of rfhutil (IH03). Set the **Queue Manager Name (to connect to)** to the name of the queue manager (MQSI_SAMPLE_QM) being used.
6. Set the **Queue Name** to the name of the output queue (MQSI_OUTQ).
7. Click **Read Q** and a confirmation message `Message read from MQSI_OUTQ` is displayed at the bottom of the utility.
8. Select the **Data** tab in the output utility. The data that has been read from the message is displayed there.

You can see in Figure 25 that the data that had been received (see PaxData1.ipt in Appendix C, “Example input message files” on page 95) has been converted to TDS output. The data now reads:

```
00000000 Passenge rSurname 50617373 656E6765 72537572 6E616D65
00000016     |1            20202020 7C31
```

The data is now all in ASCII code and the two fields are now separated by the '|' (X'7C') character that you defined as the delimiter. The meal type integer was originally X'01000000' (Windows uses little endian integer format), and now reads a character '1' (X'31').

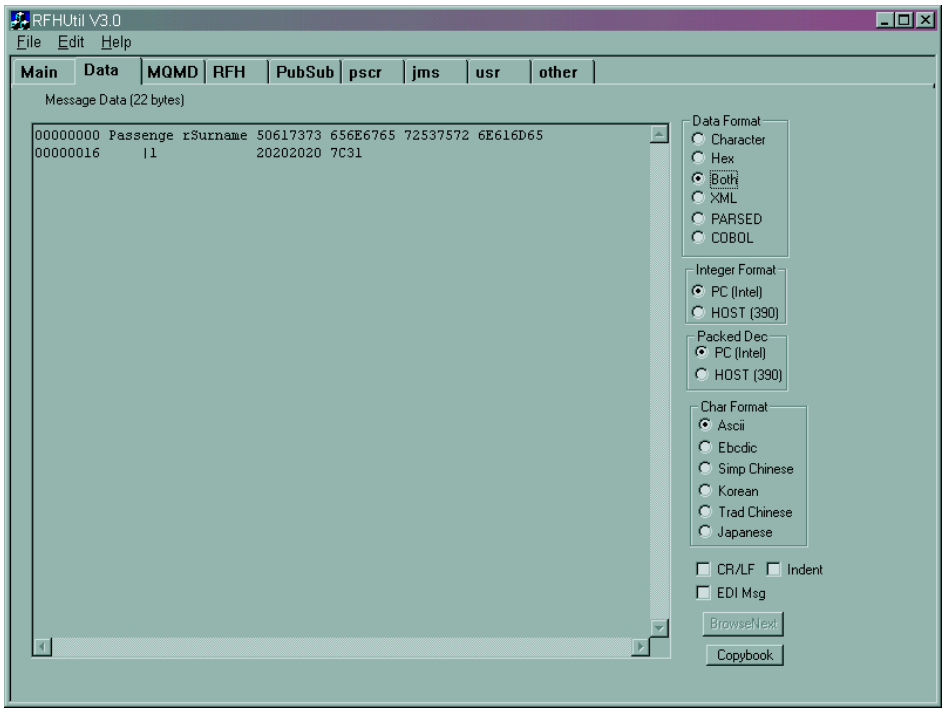

Figure 25. The IH03 dialog after reading the test message.

Error processing in a message flow

This chapter does not handle any errors that might occur in processing the message through the message flow. If you do encounter any errors and want to know how to handle them, further information on error handling with WebSphere MQ Integrator is covered in:

- *Dealing with errors in message flows* and *Debugging message flows* in Chapter 5 of *WebSphere MQ Integrator Using the Control Center*.
- *TryCatch Node* in Appendix A of *WebSphere MQ Integrator Using the Control Center*.
- *Error Handling* in Chapter 4 of *WebSphere MQ Integrator Introduction and Planning*.
- *Debugging message flows* in the Control Center Help.

In addition, each queue manager should have a dead letter queue. See “General assumptions” on page xii.

Error processing in a message flow

Chapter 8. Further CWF input processing

In Chapter 4, “Importing C or Cobol data structures into the MRM” on page 15, you looked at importing a data structure and in Chapter 7, “Converting CWF to TDS” on page 41 you saw how to create a message flow, deploy it, and test it. The file used (PaxData1.h or PaxData1.cpy), had a very simple structure containing two elements. In this chapter you will import the remaining data structures that come with this SupportPac and create the associated message flows.

This is repeating the exercises shown in Chapter 4, “Importing C or Cobol data structures into the MRM” on page 15, Chapter 5, “Creating and testing a basic message flow” on page 21, or Chapter 7, “Converting CWF to TDS” on page 41 and show that with increasing levels of complexity the basic process of importing and creating messages, message sets, and message flows is the same.

The remaining C files can be seen in Appendix A, “Example C header files” on page 87 and are PaxData2.h, PaxData3.h, and PaxData4.h.

The remaining Cobol files can be seen in Appendix B, “Example Cobol copybook files” on page 91 and are PaxData2.cpy, PaxData3.cpy, and PaxData4.cpy.

This chapter covers:

- “Reviewing the assumptions for this exercise”
- “Creating additional message sets” on page 56
- “Adding physical format layers” on page 56
- “Importing additional data structures” on page 56
- “Creating the logical messages” on page 57
- “Mapping additional TDS layers to the logical layers” on page 57
- “Assigning the message sets to the broker” on page 58
- “Creating the additional WebSphere MQ resources” on page 59
- “Creating the additional message flows” on page 59
- “Setting the additional compute node properties for CWF to TDS” on page 61
- “Setting the additional input and output node properties for CWF to TDS” on page 60
- “Confirm error processing” on page 62
- “Testing the additional message flows” on page 62

Reviewing the assumptions for this exercise

In addition to the “General assumptions” on page xii, the following apply:

- You have the default queue manager running
- You have the Configuration Manager running
- You have the default broker running
- You have started the Control Center
- You have access to the files in Appendix A, “Example C header files” on page 87 or Appendix B, “Example Cobol copybook files” on page 91
- If you are importing Cobol copybooks, you have reviewed “C header files and Cobol copybooks” on page xiii

Reviewing the assumptions for this exercise

If you not have used the defaults, you will need to substitute the names that you have used.

Creating additional message sets

Use the process "Creating a message set" on page 15 to create the message sets. Table 1 provides the names of the additional message sets you need.

Table 1. Message set names to create

File Name	Message Set
PaxData1.h or PaxData1.cpy	MRMP_MS1*
PaxData2.h or PaxData2.cpy	MRMP_MS2
PaxData3.h or PaxData3.cpy	MRMP_MS3
PaxData4.h or PaxData4.cpy	MRMP_MS4

* You created MRMP_MS1 in Chapter 4, "Importing C or Cobol data structures into the MRM" on page 15.

Adding physical format layers

Use the process "Adding physical format layers" on page 16 to add the physical layers needed for each of the message sets, prior to importing the data structure. Table 2 provides the names of the layers to use.

Table 2. CWF header files: physical format layer parameters

File Name	Message Set	CWF Layer Name	TDS Layer Name
PaxData1.h or PaxData1.cpy*	MRMP_MS1	PaxData1CWF or CWF	PaxData1TDS or TDS
PaxData2.h or PaxData2.cpy	MRMP_MS2	PaxData2CWF or CWF	PaxData2TDS or TDS
PaxData3.h or PaxData3.cpy	MRMP_MS3	PaxData3CWF or CWF	PaxData3TDS or TDS
PaxData4.h or PaxData4.cpy	MRMP_MS4	PaxData4CWF or CWF	PaxData4TDS or TDS

* You added PaxData1.h or PaxData1.cpy physical formats in Chapter 7, "Converting CWF to TDS" on page 41.

Importing additional data structures

Use the process "Importing the data structure" on page 17 to import the data structures needed for each of the message sets. Table 3 on page 57 provides the names of the messages and message IDs to use.

Note: At this point, the more complex the file structure you are importing, the more advisable it is to run a report against the file before actually importing it. In this way any errors that are likely to be generated will be picked up before the process of importing and making any changes to the message set. Appendix G, "Example CWF import report file" on page 105 contains the output from the report that was generated from importing PaxData4.h before importing it into the message set.

Table 3. CWF header files to import

Message Set	File Name
MRMP_MS1*	PaxData1.h or PaxData1.cpy
MRMP_MS2	PaxData2.h or PaxData2.cpy
MRMP_MS3	PaxData3.h or PaxData3.cpy
MRMP_MS4	PaxData4.h or PaxData4.cpy

* MRMP_MS1 was imported in Chapter 4, "Importing C or Cobol data structures into the MRM" on page 15.

Creating the logical messages

Use the process "Creating a logical message" on page 19 to create the messages needed for each of the message sets. Table 4 provides the names of the messages and message IDs to use.

Table 4. Message, Message IDs, Message Type names to use in creating the messages

Message Set	Message Name	Message ID	Message Type
MRMP_MS1*	PaxData1_Msg	PaxData1ID	PaxData1Msg_TYPE
MRMP_MS2	PaxData2_Msg	PaxData2ID	PaxData2Msg_TYPE
MRMP_MS3	PaxData3_Msg	PaxData3ID	PaxData3Msg_TYPE
MRMP_MS4	PaxData4_Msg	PaxData4ID	PaxData4Msg_TYPE

* The MRMP_MS1 message was created in Chapter 4, "Importing C or Cobol data structures into the MRM" on page 15.

Mapping additional TDS layers to the logical layers

Use the process "Mapping the TDS physical layer" on page 43 to make the additional changes to the TDS layer for each of the message sets. Table 5 on page 58 provides the names of the types for each message set that need updating

Note: The setting of properties in this section is more complex due to the nature of the structures, substructures, and repeating elements that have been imported. You will find that when you try to check in MRMP_MS3 and MRMP_MS4, you could get a TDS delimiter error. Within each of these structures there are repeating elements within substructures. These lower level elements will need to have delimiters set in addition to the upper level elements.

For the examples used in this section, '||', '|||', and '||||' have been used to delimit increasing levels of structure. Appendix D, "Example TDS output" on page 97 gives examples of the output that has been created using these. For further information on delimiters see "Delimiters" on page 45.

Mapping additional TDS layers to the logical layers

Table 5. Type names requiring modification

Message Set	Compound Types
MRMP_MS1*	PaxData1_TYPE, PaxData1Msg_TYPE
MRMP_MS2	PaxData2_TYPE, PaxData2Msg_TYPE, PaxFirstName_Type
MRMP_MS3	PaxData3_TYPE, PaxData3Msg_TYPE, PaxDestinationMsg_TYPE, PaxFirstName_TYPE, PaxRoute_TYPE
MRMP_MS4	PaxData4_TYPE, PaxData4Msg_TYPE, PaxDestinationMsg_TYPE, PaxFirstName_TYPE, PaxRoute_TYPE

* MRMP_MS1 Types were updated in Chapter 7, "Converting CWF to TDS" on page 41.

Assigning the message sets to the broker

Use the process "Assigning a message set to a broker" on page 45 to assign each of the message sets to the broker (MQSI_SAMPLE_BROKER). Table 6 provides the names of the message sets that need assigning to the broker.

Table 6. Message sets to assign to the broker (MQSI_SAMPLE_BROKER)

Message Set
MRMP_MS1*
MRMP_MS2
MRMP_MS3
MRMP_MS4

* You assigned MRMP_MS1 to the broker in Chapter 7, "Converting CWF to TDS" on page 41.

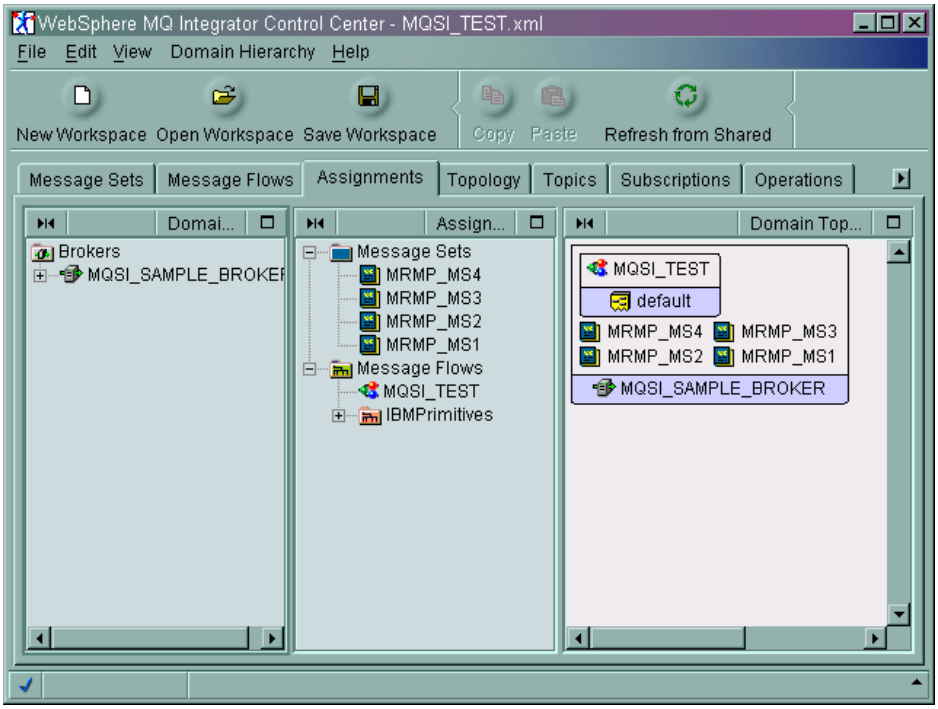

Figure 26. The WebSphere MQ Integrator Control Center after assigning the message sets

Creating the additional WebSphere MQ resources

Use the process "Creating the WebSphere MQ resources" on page 22 to create each of the input and output queues needed for the message flows. Table 7 provides the names for the queues that need creating.

Table 7. Input and output queues needed for the message flows

Message Flow	Queue Manager	Input Queue	Output Queue
MQSI_TEST*	MQSI_SAMPLE_QM	MQSI_INQ	MQSI_OUTQ
MQSI_TEST2	MQSI_SAMPLE_QM	MQSI_INQ2	MQSI_OUTQ2
MQSI_TEST3	MQSI_SAMPLE_QM	MQSI_INQ3	MQSI_OUTQ3
MQSI_TEST4	MQSI_SAMPLE_QM	MQSI_INQ4	MQSI_OUTQ4

* You created the queues for MQSI_TEST in Chapter 5, "Creating and testing a basic message flow" on page 21.

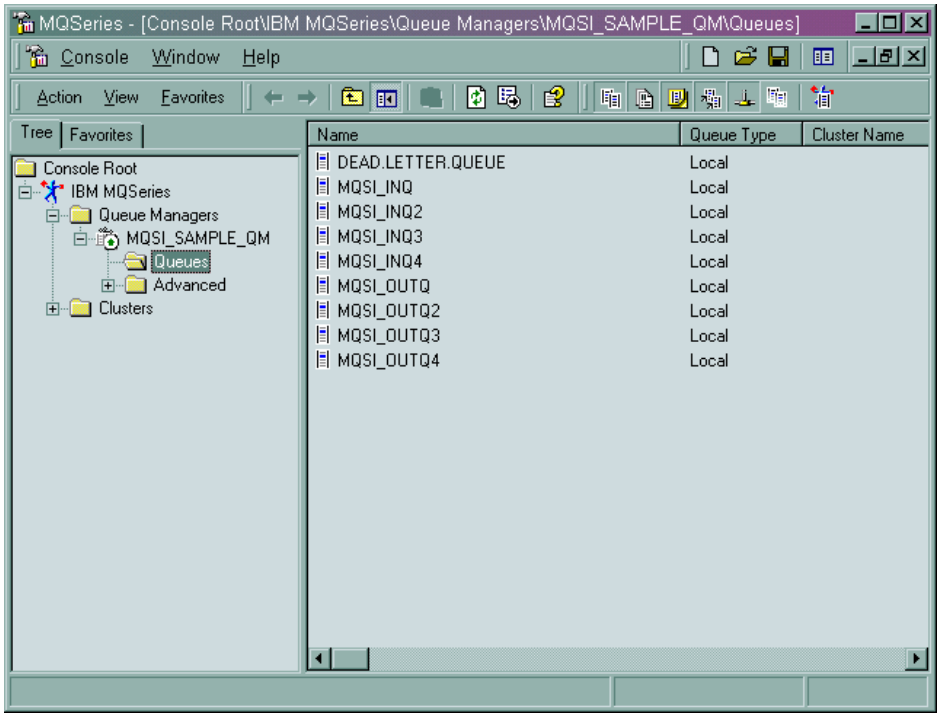

Figure 27. MQSeries Explorer after adding the additional local queues

Creating the additional message flows

Use the process "Adding a Compute node" on page 47 to create a message flow with a compute node for each of the data structures. Table 8 on page 60 provides the names for the message flows that need creating. As the input, compute, and output nodes are all in separate message flows they are able to have the same name.

Note: It is also possible to copy an existing message flow (this is covered further in Chapter 9, "Further message transformation" on page 65) and update the relevant properties or nodes to reflect the changes that you require.

Creating the additional message flows

Table 8. Message flow and node names

File Name	Message Flow	Input Node	Compute Node	Output Queue
PaxData1.h or PaxData1.cpy	MQSI_TEST*	MQInput1	Compute1	MQOutput1
PaxData2.h or PaxData2.cpy	MQSI_TEST2	MQInput1	Compute1	MQOutput1
PaxData3.h or PaxData3.cpy	MQSI_TEST3	MQInput1	Compute1	MQOutput1
PaxData4.h or PaxData4.cpy	MQSI_TEST4	MQInput1	Compute1	MQOutput1

* The message flow, MQSI_TEST, was created or modified in Chapter 7, "Converting CWF to TDS" on page 41.

Setting the additional input and output node properties for CWF to TDS

Use the process "Adjusting the Input node properties for the message set" on page 46 to update the properties of the input node for each of the data structures to convert the message from CWF to TDS. Table 9 and Table 10 provide the names for the relevant properties for each of the input and output nodes that need updating.

Table 9. Input node properties needed for the message flows

Message Flow	Input Node	Queue Name	Message Domain	Message Set**	Message Type	Message Format
MQSI_TEST*	MQInput1	MQSI_INQ	MRM	MRMP_MS1	PaxData1ID	PaxData1CWF (CWF)
MQSI_TEST2	MQInput1	MQSI_INQ2	MRM	MRMP_MS2	PaxData2ID	PaxData2CWF (CWF)
MQSI_TEST3	MQInput1	MQSI_INQ3	MRM	MRMP_MS3	PaxData3ID	PaxData3CWF (CWF)
MQSI_TEST4	MQInput1	MQSI_INQ4	MRM	MRMP_MS4	PaxData4ID	PaxData4CWF (CWF)

Table 10. Output node properties needed for the message flows

Message Flow	Output Node	Queue Manager	Queue Name
MQSI_TEST	MQOutput1	MQSI_SAMPLE_QM	MQSI_OUTQ
MQSI_TEST2	MQOutput1	MQSI_SAMPLE_QM	MQSI_OUTQ2
MQSI_TEST3	MQOutput1	MQSI_SAMPLE_QM	MQSI_OUTQ3
MQSI_TEST4	MQOutput1	MQSI_SAMPLE_QM	MQSI_OUTQ4

* You updated the input node in MQSI_TEST in Chapter 7, "Converting CWF to TDS" on page 41.

** You must specify the message set ID and not its name. This ID is unique to the message set on your system. Further details are in "Adjusting the Input node properties for the message set" on page 46.

Setting the additional compute node properties for CWF to TDS

Use the process "Setting the Compute node properties for CWF to TDS conversion" on page 50 to update the properties of the compute node for each of the data structures to convert the message from CWF to TDS. Table 11 provides the names for the compute nodes for each message flow that needs updating.

Table 11. Input and output queues needed for the message flows

File Name	Message Flow	Compute Node	Message Set	Message
PaxData1.h or PaxData1.cpy*	MQSI_TEST	Compute1	MRMP_MS1	PaxData1_Msg
PaxData2.h or PaxData2.cpy	MQSI_TEST2	Compute1	MRMP_MS2	PaxData2_Msg
PaxData3.h or PaxData3.cpy	MQSI_TEST3	Compute1	MRMP_MS3	PaxData3_Msg
PaxData4.h or PaxData4.cpy	MQSI_TEST4	Compute1	MRMP_MS4	PaxData4_Msg

* The compute node in MQSI_TEST was updated in Chapter 7, "Converting CWF to TDS" on page 41.

Defining the broker in the domain

If you have already completed the work in Chapter 5, "Creating and testing a basic message flow" on page 21 or Chapter 7, "Converting CWF to TDS" on page 41 you can skip this section because your broker is already defined.

Use the process "Defining a broker in the domain" on page 24 from Chapter 5, "Creating and testing a basic message flow" on page 21 to ensure that the broker that was created during the installation process has been added to the domain you are working in.

Assigning the message flows to the execution group

Use the process "Assigning the message flow to the execution group" on page 26 to ensure that the message flow has been assigned to the broker. Table 12 provide the names of the message flows that need assigning to the broker.

Table 12. Message flows to assign to the broker

Message Flow
MQSI_TEST*
MQSI_TEST2
MQSI_TEST3
MQSI_TEST4

* The message flow MQSI_TEST was assigned in Chapter 7, "Converting CWF to TDS" on page 41.

At the end of this processing your Control Center will appear similar to Figure 28 on page 62.

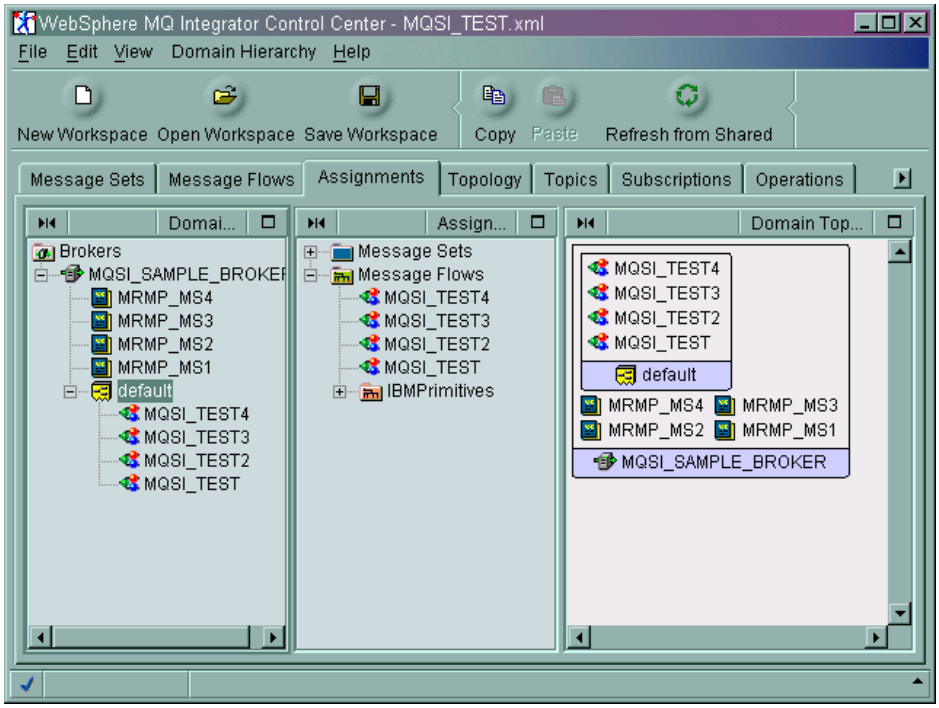

Figure 28. The Control Center after assigning all the resources to the broker

Deploying the configuration to the broker

Use the process "Deploying the message flow to the broker" on page 27 to ensure that all the changes made to the message sets and the message flows are deployed to the broker for its use in processing messages.

Note: You were told in an earlier chapter that error messages can be generated either when you check in or deploy any changes that you have made. Where BINARY types have been generated for elements that you have imported, the TDS physical layer will not have set the length for them and you will get an error when you deploy the message sets that contain these BINARY elements. You will need to update the TDS length for each of these elements to have the same length as that shown in the CWF physical layer.

Confirm error processing

Check that a dead letter queue has been set up. "Error processing in a message flow" on page 53 lists where you can find further information on error processing.

Testing the additional message flows

Use the process "Testing the CWF to TDS message flow" on page 52 to test the message flows you have created to convert the messages from CWF to TDS. Table 13 on page 63 provides the names of the input files, input queues, and output queues that you use for checking each of the flows you have created.

Put the message from the specified file on the appropriate input queue.

Table 13. Input and output queues needed for the message flows

File Name	Queue Manager	Input Queue	Output Queue
PaxData1.ipt*	MQSI_SAMPLE_QM	MQSI_INQ	MQSI_OUTQ
PaxData2.ipt	MQSI_SAMPLE_QM	MQSI_INQ2	MQSI_OUTQ2
PaxData3.ipt	MQSI_SAMPLE_QM	MQSI_INQ3	MQSI_OUTQ3
PaxData4.ipt	MQSI_SAMPLE_QM	MQSI_INQ4	MQSI_OUTQ4

* You tested the first message flow with PaxData1.ipt in Chapter 7, "Converting CWF to TDS" on page 41.

Testing the additional message flows

Chapter 9. Further message transformation

In previous chapters you have imported data structures, created message sets from them and, using message flows, transformed the incoming messages into a different data format. This chapter will look at how to examine an element (field) in an incoming message and process the message as defined by the data in that element.

This is illustrated in its simplest form to show you the principle of this technique.

This chapter will also introduce how to create message sets based on another message set, add an element to a message set, and copy a message flow.

The message that you will use to test the message flow is based on PaxData4.ipt and has an additional field at the beginning of it. This field is used to specify the output format of the message. It is a three character field and could contain any data.

The process you will create will expect the characters CWF, TDS, or XML in this field and this specifies the output format of the message. Any other data that exists in this field, will cause the output format of the message to default to CWF.

In this chapter you will:

- "Review the assumptions for this exercise"
- "Create the message set" on page 66
- "Create the message flow" on page 69
- "Assign, deploy, and test the message flow" on page 72

Review the assumptions for this exercise

In addition to the "General assumptions" on page xii, the following apply:

- You have the default queue manager running
- You have the Configuration Manager running
- You have the default broker running
- You have started the Control Center
- You have access to the files in Appendix C, "Example input message files" on page 95
- If you have imported Cobol copybooks, you have reviewed "C header files and Cobol copybooks" on page xiii

If you not have used the defaults, you will need to substitute the names that you have used.

Create the message set

In order for the broker to interpret the incoming message, the message needs to be defined in a message set. You have done this previously, by importing data structures that have already existed. Message sets can also be created totally within the MRM without using any external structures or components. In addition to these methods, you can base a new message set on an existing one, making use of all components previously created.

During this exercise you will create the message set needed to parse this new message format, based on an existing one created in an earlier chapter.

In this section you will:

- "Create a message set based on another message set"
- "Add the message and message types to the message set workspace" on page 67
- "Add an XML physical layer" on page 67
- "Add an element to a message set" on page 67

Create a message set based on another message set

To create a message set based on another, the message set to be copied must be in Finalized state. Details on message set states can be found in the *Working with Messages* book. A message set that has been finalized cannot be changed or checked out.

The message sets that you have created so far are in Normal state (that is, they are checked in) and could be considered to be 'work in progress'. You can Finalize a message set when you have completed its development and do not need to make any changes to it. This does not stop development of the message set in the future because it can be copied and have the same name. This gives you the ability to create versions of the message set showing the evolution of how it developed.

You will use the last message set that you created in Chapter 8, "Further CWF input processing" on page 55 (MRMP_MS4) to base your new message set on.

To create a message set based on another message set:

1. In the Control Center, select the **Message Sets** tab.
2. Right-click the message set (MRMP_MS4) you are copying and select **Finalize**.

   Select **Yes** to confirm you want to finalize the message set.

   You cannot create a message set based on another without doing this first. You will see that the finalized field in the right pane is now set to true.
3. Right-click the **Message Sets** root in the left pane and select **Create —> Message Set...**. This opens the **Create a new Message Set** dialog.
4. Give the message set a name (MRMP_MS_TR1) and from the **Base Message Set** drop down list at the bottom of the dialog, select the name of the message set (MRMP_MS4) you are basing this message set on.
5. Click **Finish**.

This creates the new message set based on the original you developed in earlier chapters. At this point it has only created the base message set. Such things as the message, or types have not yet been added to the workspace.

Add the message and message types to the message set workspace

You saw in an earlier chapter that importing a data structure creates the components such as messages, elements, and types, but they are not present in the workspace until you add them yourself. You will need to work with the message and message types, therefore you must add these to the workspace:

1. In the Control Center, select the **Message Sets** tab.
2. Expand the message set (MRMP_MS_TR1) root in the left pane. (Click the **+** to the left of the message set name.)
3. Right-click **Messages** and select **Add to Workspace —> Message...**.
4. This opens the **Add existing Message** dialog. Select the message (PaxData4_Msg) from this dialog.
5. Click **Finish**.
6. Right-click **Types** and select **Add to Workspace —> Compound Types...**.
7. This opens the **Add an existing Compound Type** dialog. Select all the Types from this dialog.
8. Click **Finish**.

All the messages and types that were in the original message sets workspace have been added to this one. By expanding the message set, messages, and types root for both message sets (MRMP_MS4 and MRMP_MS_TR1) you can see that they are the same.

What you will see is that the icons to the left of the components in the new message set (MRMP_MS_TR1) are different. This icon (a small world) indicates that these objects have an external reference. The color of this icon indicates the state of the component. For further information on the definition of these icons see the Control Center help.

Add an XML physical layer

The message set (MRMP_MS4) that was used as the base to create this message set already had a CWF and TDS physical layer added to it. You can see this by selecting a message set component in the Control Center and see the CWF or TDS tabs in the right pane.

This exercise will allow the output format to be a choice of either CWF, TDS, or XML. To format the output to XML, you must add an XML physical layer to the message set.

To add an XML physical layer to the message set:

1. In the Control Center, select the **Message Sets** tab.
2. Right-click the message set (MRMP_MS_TR1) root in the left pane and select **Add —> Physical Format... —> XML Format...**. This opens the **Add a XML Format** dialog.
3. Enter the name for this XML layer (XML or PaxData4XML).
4. Click **Finish**.

Add an element to a message set

The message flow you are creating will read an element (field) in the message that is received. Based on the data in this element, it will adjust the output format to the one defined in the element.

Add an element to a message set

The basis of the message to be used is the most complex message created in Chapter 8, "Further CWF input processing" on page 55 (PaxData4). All that you will add to the structure of this message is an element at the beginning that will define the message output format. The process for adding an element is the same, regardless of how the message is constructed (for example, either a new message or an imported one).

To add an element into a message:

1. In the Control Center, select the **Message Sets** tab.
2. Expand the message set (MRMP_MS_TR1) root in the left pane. (Click the **+** to the left of the message set name.)
3. Right-click the **Elements** root and select **Create —> Element...**. This opens the **Create a new Element** dialog.

   In the **Name** field, enter the name (OptMsgFormat) for your element.

   In the **Identifier** field, enter the identifier name (OptMsgFormatID) for your element.

   Note: You will use this identifier to address this element when you create ESQL code to process a message in the compute node.

   In the **Type** field, use the drop-down box to select **STRING**.

   Click **Finish**.
4. Right-click the element (OptMsgFormat) and select **Check In**.

   This has created a new element, and its properties include the TDS and XML layers, but not the CWF layer. TDS and XML elements have both element properties and Type Member properties (created when an element is instantiated as a child of a compound type). CWF elements just have Type Member properties and this is why the CWF tab does not appear at the element level. Type Member properties can be viewed by selecting the element within the parent type.

   You will now add this element to a compound type.
5. Expand the **Types** root for the message set (MRMP_MS_TR1) in the left pane. (Click the **+** to the left of Types.)

   Expand the main message compound type (PaxData4Msg_TYPE). (Click the **+** to the left of the compound type name (PaxData4Msg_TYPE).)
6. Right-click the compound type name to add the element to (PaxData4_TYPE) and select **Check Out List...**.

   The **Check Out List** dialog appears. Select **Check Out**.
7. Right-click the compound type name to add the element to (PaxData4_TYPE) and select **Add —> Element...**. This will open the **Add an existing Element** dialog.

   Select the element that you have just created (OptMsgFormat) and click **Finish**.
8. From the File menu, select **Check In —> All (Saved to Shared)**.

   If you select the element that you have just added from within the compound type, you will see that the CWF layer is visible.
9. Right-click the compound type (PaxData4_TYPE) and select **Check Out List...**. This opens the **Check Out List** dialog (see Figure 29 on page 69). This dialog has listed and selected all the components (for example Elements, Element Values, and Types) that are associated with this compound type. Click **Check Out**.

It can be useful to check out components in this manner if you are not sure whether the property you want to update is an element property or a type member property.

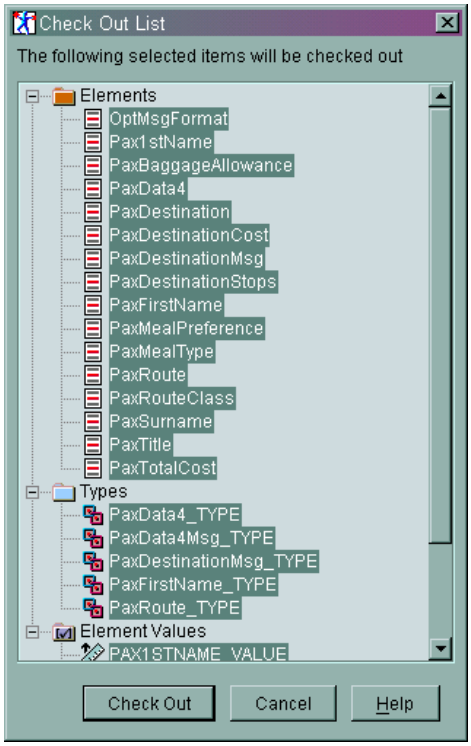

Figure 29. The Check Out List dialog

10. Under the **Type** root in the left pane, expand the message compound type (PaxData4Msg_TYPE) until you can see the output message format element (OptMsgFormat) element and select it.

    In the right pane select the CWF physical layer tab (CWF or PaxData4CWF). Update the **Length Count** (currently empty) to read **3**.

    Click **Apply** in the bottom of the right pane.
11. From the menu select **File —> Check In —> All (Saved to Shared)**.

Create the message flow

As in previous exercises you will need a message flow to process the message and make any changes to it. In the previous exercises you have created the message flows from the beginning, but in this one you will copy an existing message flow and update it to handle the new processing.

In this section you will:

- "Create the additional WebSphere MQ resources" on page 70
- "Copy an existing message flow" on page 70
- "Update the message flow nodes" on page 70

Create the additional WebSphere MQ resources

Use the process “Creating the WebSphere MQ resources” on page 22 from Chapter 5, “Creating and testing a basic message flow” on page 21 to create an additional input (MQSI_INQ_TR1) and output (MQSI_OUTQ_TR1) queue needed for the new message flow.

Copy an existing message flow

To copy an existing message flow:

1. In the Control Center, select the **Message Flows** tab.
2. In the left pane, right-click the message flow that you are to copy (MQSI_TEST4) and select **Copy**.
3. In the left pane, right-click the **Message Flows** root and select **Paste**.

   This adds a copy of the message flow with a similar name (MQSI_TEST4_1) to the message flow being copied. All the nodes in the new message flow will have the same attributes as those in the original message flow.
4. Right-click the new message flow name (MQSI_TEST4_1) and select **Rename...**.

   This opens the **Rename Message Flow** dialog.

   Change the name to the name (MQSI_TEST_TR1) you will use during this exercise.

   Click **Finish**.

You have now added a copy of a message flow to your workspace. Apart from the new name that you have assigned, it is identical to the original.

Update the message flow nodes

This new message flow has input and output nodes that have properties associating them with the original message flow queues. In addition, the input node has the original message flow’s message set defined in it. All of these need changing to reflect the new processing this message flow is to assume.

Update the input node

To update the input node:

1. In the Control Center, select the **Message Flows** tab.
2. In the right pane, right-click the input node (MQInput1) and select **Properties**. This opens the input node (MQInput1) properties dialog.
3. Select the **Basic** tab.

   Change the **Queue Name** field to the name of the new input queue (MQSI_INQ_TR1).
4. Select the **Default** tab.
5. Ensure the **Message Domain** field specifies **MRM**

   Change the **Message Set** field to the ID of the message set you created in “Create a message set based on another message set” on page 66.

   Ensure the **Message Type** field is the **Identifier** (PaxData4ID) of your incoming message. (From the message in your new message set, but based on the original.)

   Ensure the **Message Format** field reads the name of your CWF layer (PaxData4CWF (CWF)). (From the new message set, but based on the original.)
6. Click **OK**.

Update the output node

To update the output node:

1. In the Control Center, select the **Message Flows** tab.
2. In the right pane, right-click the output node (MQOutput1) and select **Properties**. This opens the output node (MQOutput1) properties dialog.
3. Select the **Basic** tab.
4. Ensure the **Queue Manager** field reads name of the queue manager you are using (MQSI_SAMPLE_QM).

   Change the **Queue Name** field to the name of the new output queue (MQSI_OUTQ_TR1).
5. Click **OK**.

Update the compute node

In previous exercises, the compute node has only handled the transformation from CWF format to TDS format. In this exercise the compute node needs to look at data in the output message to decide what format the message needs to be written in.

This is done using ESQL statements to check the data and set the output format for the message. Other than this transformation, no other processing of the data occurs. What you need to do is update the ESQL code.

To update the compute node:

1. In the Control Center, select the **Message Flows** tab.
2. In the left pane select the message flow to update (MQSI_TEST_TR1).
3. In the right pane, right-click the compute node (Compute1) and select **Properties**. This opens the compute node (Compute1) properties dialog.
4. Select the **ESQL** tab in the bottom pane. The code showing in this field is:

   ```
   SET OutputRoot = InputRoot;
   -- Enter SQL below this line.  SQL above this line might be regenerated,>>
   >> causing any modifications to be lost.*
   SET OutputRoot.Properties.MessageFormat = 'TDS';
   ```

   * The line has been split at >> for the purposes of displaying in this document.
5. Update the field to reflect the following changes:

   ```
   SET OutputRoot = InputRoot;
   -- Enter SQL below this line.  SQL above this line might be regenerated,>>
   >> causing any modifications to be lost.*
   -- Default output format is to be CWF
   IF (InputBody.PaxData4.OptMsgFormatID = 'TDS') THEN
       SET OutputRoot.Properties.MessageFormat = 'TDS';
   ELSE
       IF (InputBody.PaxData4.OptMsgFormatID = 'XML') THEN
           SET OutputRoot.Properties.MessageFormat = 'XML';
       END IF;
   END IF;
   ```

   * The line has been split at >> for the purposes of displaying in this document.

   Note: PaxData4 and OptMsgFormatID are the names used in the development of this and earlier exercises. If you have used other names, you must substitute these in the code.

The comment clarifies the purpose of the code indicating that if the message received specifies something other than expected, it will default to a CWF output format.

The code checks the output message format field and sets the format accordingly.

There are a number of different ways you could code this and still achieve the same result. This is just one of them and it works!

6. Click **OK**.
7. From the menu select **File —> Check In —> All (Save to Shared)**.

Assign, deploy, and test the message flow

In this chapter, you have created the message set, message, and message flow needed for this exercise. In this section you will:

- "Assign the message set to the broker"
- "Assign the message flow to an execution group"
- "Deploy the configuration to the broker"
- "Test the message flow"

Assign the message set to the broker

Using the instructions in "Assigning a message set to a broker" on page 45 from Chapter 7, "Converting CWF to TDS" on page 41, assign the message set (MRMP_MS_TR1) to the broker (MQSI_SAMPLE_BROKER).

Assign the message flow to an execution group

Use the process "Assigning the message flow to the execution group" on page 26 from Chapter 5, "Creating and testing a basic message flow" on page 21 to add the message flow (MQSI_TEST_TR1) to the broker.

Deploy the configuration to the broker

Use the process "Deploying the message flow to the broker" on page 27 from Chapter 5, "Creating and testing a basic message flow" on page 21 to ensure that all the changes made to the message sets and the message flows are deployed to the broker for its use in processing messages.

Test the message flow

Use the process "Testing the CWF to TDS message flow" on page 52 from Chapter 7, "Converting CWF to TDS" on page 41 to test the message flow you have created.

Use the input and output message queues that you created in "Create the additional WebSphere MQ resources" on page 70.

The input files MQSI_TR1_CWF.ipt, MQSI_TR1_TDS.ipt, MQSI_TR1_XML.ipt, and MQSI_TR1_Other.ipt have been created to help you test the message flow. The first three characters in the message contained in the file are set to the three letters defined in the file name. You can see these files in Appendix H, "Variable conversion input files" on page 107.

You can see the output that is generated from each of these files in Appendix I, "Variable conversion output files" on page 111. The default for the output message that is generated is CWF. You will see that the output from the CWF input file and the Other input file, is almost identical. The only difference is the OTH string in the Other input file: because this is not explicitly handled in the ESQL, the corresponding element in the output message defaults to CWF.

Test the message flow

Chapter 10. Basic message manipulation

In previous exercises you have transformed the message from one format to another and looked at data within the message to determine the output format for the message.

In this chapter you will look at how to perform some basic message manipulation. You will:

- Remove an element from the input message
- Add a date time stamp to the output message

As in previous chapters, you will be referred to exercises that are performed in earlier chapters if they need repeating in this chapter. You will be provided with alternate names if they are needed.

This chapter covers:

- "Reviewing assumptions for this exercise"
- "Creating the message set"
- "Creating the message flow" on page 81
- "Assigning, deploying, and testing the message flow" on page 84

Reviewing assumptions for this exercise

In addition to the "General assumptions" on page xii, the following apply:

- You have the default queue manager running
- You have the Configuration Manager running
- You have the default broker running
- You have started the Control Center
- You have access to the files in Appendix C, "Example input message files" on page 95
- If you have imported Cobol copybooks, you have reviewed "C header files and Cobol copybooks" on page xiii

If you not have used the defaults, you will need to substitute the names that you have used.

Creating the message set

For this exercise, you will base the new message set on one created in an earlier exercise. When you are manipulating data within a message, if it is only the data within an element that is changing, you can use the same message definition for both the input and output. Because you are adding elements and deleting elements, you will use a separate message for both the input and the output to ensure that the message is handled and parsed correctly.

In this section, you need to create a new message set based on an earlier one and then create the necessary elements, types, and messages to support the processing in this chapter. A majority of the information in the new objects is identical to the original incoming message. For this reason most of the new objects are based on the original message set.

Creating the message set

One of the features of the structures you have been working with, is that they consists of a structure within a structure. For example PaxData4.h reads:

```
struct _PaxData4Msg          /*Start of first structure*/
{
    struct _PaxData4         /*Start of second structure*/
    {
        char            PaxSurname [20];
        struct          _PaxFirstName
        {
            char            Pax1stName [20];
        }PaxFirstName;
        unsigned char   PaxTitle[10];
        struct          _PaxRoute
        {
            char            PaxRouteClass [20];
            struct          _PaxDestinationMsg
            {
                char            PaxDestination [20];
                float           PaxDestinationCost;
                char            PaxDestinationStops [3] [10];
            } PaxDestinationMsg [4];
        }PaxRoute [2];
        unsigned char   PaxBaggageAllowance[2];
        short           PaxMealType;
        char            PaxMealPreference [20];
        float           PaxTotalCost;
    } PaxData4;           /*End of second structure*/
}PaxData4Msg;            /*End of first structure*/
```

Each of the example C header files (and equivalent Cobol copybooks) that you have been using have had this type of structure. This 'doubling' style of definition has little bearing on the form of the message, but does add an extra layer of complexity in defining and creating the objects within the MRM.

After importing the data structure, it would have been possible to create a message without this 'double' structure and still process the message correctly within the MRM. This could have been achieved by importing the data structure and creating a new message that is based on the Type of the 'inner' structure. This would also have had a follow on effect on the addressing of objects as the exercises progressed.

In this exercise you are still using this type of structure and need to create an output message (PaxData5_Msg) and an element (PaxData5) based on a type to reflect the structure that has been imported and provide consistency in the processing.

Create the message set and add the message and types

You will need to create a message set based on the message set that you created in Chapter 9, "Further message transformation" on page 65.

Use the following steps for this section:

1. "Create a message set based on another message set" on page 66. The message set to base it on is the one you created in that specific exercise (MRMP_MS_TR1 and not MRMP_MS4) and you will create a new one (MRMP_MS_TR2).
2. "Add the message and message types to the message set workspace" on page 67. Use the new message set name (MRMP_MS_TR2).

You will not need to add the XML layer to this message set because it was added to the message set that this new one is based (MRMP_MS_TR1) on.

Copy a message set type

The fastest way to create a new message type with the necessary elements is to copy an existing one and update it as necessary. This is not always appropriate if the new message type is completely unrelated to an existing one. In this exercise there is only one difference, and copying is therefore an appropriate method for creating this new message type.

Use the following steps for this exercise:

1. In the Control Center, select the **Message Sets** tab.
2. In the left pane, expand (click the **+** to the left of an object) the message set (MRMP_MS_TR2) root and the Types within that message set to show all the data types.
3. Right-click the PaxData4_Type (or the name you assigned in the earlier exercise) and select **Copy**.
4. Right-click **Types** and select **Paste**. This creates a new message type called **PaxData4_TYPE_1**.
5. Right-click this new Type and select **Rename**. This opens the **Rename Compound Type** dialog.
6. Give the Type the name **PaxData5_TYPE**.
7. Click **Finish**.
8. From the menu select **Check In —> All (Saved to Shared)**.

Create an element based on the new Type

The next stage in the creating your new message set is to create an element that is based on the type you created in “Copy a message set type”. This is required to ensure that the new message you are creating has the same format and structure as the message being received.

Use the following steps for this exercise:

1. In the Control Center, select the **Message Sets** tab.
2. In the left pane, expand (click the **+** to the left of an object) the message set (MRMP_MS_TR2) root.
3. Right-click **Elements** and select **Create**. This opens a **Create a new Element** dialog.
4. Give the **Element** the name **PaxData5** and the identifier **PaxData5**.
5. In the **Type** property, use the drop down selection box and select **PaxData5_TYPE**.
6. Click **Finish**.
7. From the menu select **Check In —> All (Saved to Shared)**.

Create a type and add an element

In "Copy a message set type" on page 77 and "Create an element based on the new Type" on page 77 you have created the 'inner' structure of the message as discussed in "Creating the message set" on page 75. In this section you will now create the type and add the element to it that completes the structure (the inner structure created above) within a structure (the outer structure created in this section).

Use the following steps for this exercise:

1. In the Control Center, select the **Message Sets** tab.
2. In the left pane, expand (click the **+** to the left of an object) the message set (MRMP_MS_TR2) root.
3. Right-click **Types** and select **Create —> Compound Type...**. This opens a **Create a new Compound Type** dialog.

   Set the **Name:** to **PaxData5Msg_TYPE**.

   Set the **Type Content** from the drop-down selection to **Closed**.

   Set the **Identifier** to **PaxData5Msg_TYPE**.

   Leave all other settings to their default value.

   Click **Finish**.

   Right-click the new compound type (PaxData5Msg_TYPE) and select **Check In**. The TDS layer tab will now display.

   Right-click the new compound type (PaxData5Msg_TYPE) and select **Check Out**.

   Select the TDS tab (PaxData4TDS as defined in the original message set) in the right pane.

   Set the **Data Element Separation** to **All Elements Delimited**.

   Set the **Delimiter** to **'|'**.

   Select **Apply** in the bottom of the right pane.
4. Right-click the new Type name (PaxData5Msg_TYPE) and select **Add —> Element...**. This opens the **Add an existing Element** dialog.
5. Select the element created in "Create an element based on the new Type" on page 77 (PaxData5).
6. Click **Finish**.
7. From the menu select **Check In —> All (Saved to Shared)**.

At this stage you now have the compound Types PaxData4, PaxData4Msg, PaxData5 and PaxData5Msg. If you now compare the contents of each of the types (PaxData4 to PaxData5, and PaxData4Msg to PaxData5Msg), apart from the icons, the elements will be identical. See Figure 30 on page 79.

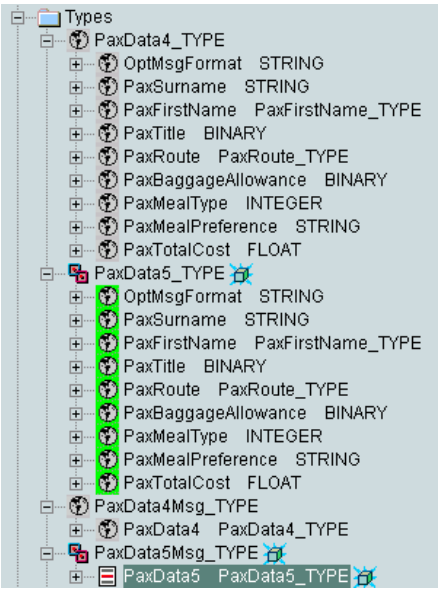

Figure 30. Screen shot of the Type tree that has just been created

Add a new message to the message set

So far you have added the necessary Elements and Types to your message set that you require for the output message. You will now need to create this new message in the message set.

Use the following steps for this exercise:

1. In the Control Center, select the **Message Sets** tab.
2. In the left pane, expand (click the + to the left of an object) the message set (MRMP_MS_TR2) root.
3. Right-click **Messages** and select **Create —> Message...**. This opens a **Create a new Message** dialog.
4. Give the **Message** the name **PaxData5_Msg** and the identifier **PaxData5ID**.
5. For the **Type** use the drop down box to select the Type **PaxData5Msg_TYPE**.
6. Click **Finish**.
7. From the menu select **Check In —> All (Saved to Shared)**.

At this point you will now have two messages in the workspace that have the same elements within their data types. This can be seen by expanding (click the + to the left of an object) the data types within the message to show the elements.

Removing an element from a compound type

At this point you have created a copy of the incoming message and these are the templates used for parsing the incoming and outgoing message. You know that the incoming message definition is correct because it is identical to the one used to test in Chapter 9, “Further message transformation” on page 65. At this point the outgoing message definition has not been updated and needs to reflect the content and structure of the message you are to send.

Now you need to change the message to remove an element that you do not want to send.

Removing an element from a compound type

Use the following steps for this exercise:

1. In the Control Center, select the **Message Sets** tab.
2. In the left pane, expand (click the **+** to the left of an object) the root the message set (MRMP_MS_TR2) root.
3. In the left pane, expand the **Types** root.
4. Right-click **PaxData5_TYPE** and select **Check Out**.
5. Expand the **PaxData5_TYPE** and right-click **OptMsgFormat**. Select **Remove**.
6. From the menu select **Check In —> All (Saved to Shared)**.

This removes the Element OptMsgFormat from the Type PaxData5_TYPE. If you expand the PaxData5Msg_TYPE under Types or the PaxData5_Msg under Messages, you will see that this removal applies to the element wherever it appears in the PaxData5 message structure.

Create and add an element to a compound type

You now need to create and add the new element that is to contain the additional data you need to send. This element is going to hold a date time stamp as a string.

Use the following steps for this exercise:

1. In the Control Center, select the **Message Sets** tab.
2. In the left pane, expand (click the **+** to the left of an object) the root the message set (MRMP_MS_TR2) root.
3. Right-click **Elements** and select **Create —> Element...**. This opens the **Create a new Element** dialog.
4. In the **Name** field enter **DateTimeStamp**.
5. In the **Identifier** field enter **DateTimeStampID**.
6. In the **Type** field, use the drop down box to select **STRING**.
7. Click **Finish**.
8. Right-click **PaxData5_TYPE** under the **Types** root. and select **Check Out**.
9. Right-click **PaxData5_TYPE** under the **Types** root. and select **Add —> Element...**. This opens the **Add an existing Element** dialog.
10. Select **DateTimeStamp** and click **Finish**. This adds the element DateTimeStamp to the Type.

    This has added the DateTimeStamp element to the top of the list of elements within the compound type. The order of the elements in a compound type can be important for some physical formats (for example CWF) as it will be the order that the broker expects to receive the elements in a message and the order it will write them in.

    It is possible to reorder the elements by right-clicking the compound type and selecting **Reorder**.

    For the message you are creating, the DateTimeStamp has been added in the correct position.
11. From the menu select **Check In —> All (Saved to Shared)**. This is required to ensure the physical layers are displayed correctly with the new element.
12. Right-click **PaxData5_TYPE** under the **Types** root. and select **Check Out**.
13. Right-click **DateTimeStamp** under **PaxData5_TYPE** and select **Check Out**.
14. Select the CWF tab (PaxDataCWF) in the right pane for the **DateTimeStamp** element.

Set the **Length Count** property to **23**. This will be the length of the date time stamp in the output message.

Click **Apply** at the bottom of the right pane.

15. From the menu select **Check In —> All (Saved to Shared)**.

You will now have an input message (PaxData4_Msg) and an output message (PaxData5_Msg) that will look as those shown in Figure 31.

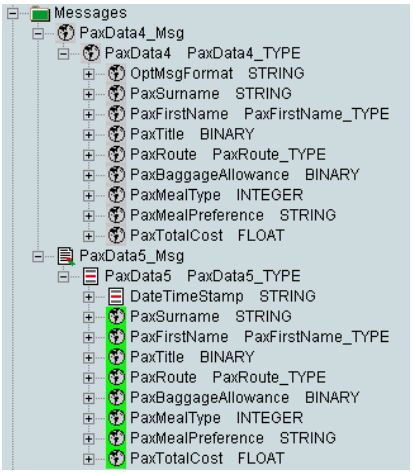

Figure 31. The input and output message

Creating the message flow

In the previous section you have created the necessary message set that will allow the broker to interpret the message it receives and to correctly format the data for the output message. In this section you will be creating the message flow that will propagate the incoming data to the outgoing message.

Create the additional WebSphere MQ resources

Use the process “Creating the WebSphere MQ resources” on page 22 from Chapter 5, “Creating and testing a basic message flow” on page 21 to create an additional input (MQSI_INQ_TR2) and output (MQSI_OUTQ_TR2) queue needed for the new message flow.

Copy an existing message flow

Use the section “Copy an existing message flow” on page 70 to copy the message flow. The changes to the process are:

- MQSI_TEST_TR1 is the message flow to copy.
- MQSI_TEST_TR1_1 is the message flow name created.
- MQSI_TEST_TR2 is the message flow name to rename it to.

Update the message flow nodes

This new message flow has input and output nodes that have properties assigning them to the original message flow queues. In addition, the input node has the original message flows message set assigned to it. All of these need changing to reflect the new processing this message flow is to assume.

Update the input node

Use the section "Update the input node" on page 70 to update the input node to use the new input queue and also set the new message set information to ensure that it is correctly parsed. The changes to the process are:

- The message flow to update is MQSI_TEST_TR2.
- The message set to use is that created in MRMP_MS_TR2.

Update the output node

Use the section "Update the output node" on page 71 to update the output node to use the new output queue. The only change to the process is the output message queue is MQSI_OUTQ_TR2.

Update the compute node

The processing in previous compute nodes has been fairly basic. In the earlier ones the input message was copied to the output message and later on had the format setting changed to send the message in a different format.

In this section you will be manipulating the data and ensuring that it is sent in the correct order. The section *Message Tree Structures* in the *Working with Messages* book and *Referring to information in messages* in the *ESQL Reference* book are useful sections to read to understand some of the concepts that will be dealt with during the changes to this node.

To update the compute node:

1. In the Control Center, select the **Message Flows** tab.
2. In the left pane select the message flow to update (MQSI_TEST_TR2).
3. In the right pane, right-click the compute node (Compute1) and select **Properties**. This opens the compute node (Compute1) properties dialog.
4. Select the **ESQL** tab in the bottom pane. The code showing in this field is currently:

```
SET OutputRoot = InputRoot;
-- Enter SQL below this line.  SQL above this line might be regenerated,>>
>> causing any modifications to be lost.*
-- Default output format is to be CWF
IF (InputBody.PaxData4.OptMsgFormatID = 'TDS') THEN
    SET OutputRoot.Properties.MessageFormat = 'TDS';
ELSE
    IF (InputBody.PaxData4.OptMsgFormatID = 'XML') THEN
        SET OutputRoot.Properties.MessageFormat = 'XML';
    END IF;
END IF;
```

   * The lines have been split at >> for the purposes of displaying in this document.

5. In the processing that you will apply here, you do not need all the data from the input message. You will delete the OptMsgFormat element by not copying it to the output message. This means that you will not need to copy the whole message, but only the header. The first step is to select the radio button **Copy Message Headers**. This will change the code from:

```
SET OutputRoot = InputRoot;
-- Enter SQL below this line.  SQL above this line might be regenerated,>>
>> causing any modifications to be lost.
```

to:

```
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
 SET OutputRoot.*[I] = InputRoot.*[I];
 SET I=I+1;
END WHILE;
-- Enter SQL below this line.  SQL above this line might be regenerated,>>
>> causing any modifications to be lost.
```

6. The broker needs to be told to use the new message (PaxData5) to write (parse) the data to the output message. To do this add the following statement:

   ```
   SET OutputRoot.Properties.MessageType = 'PaxData5ID';
   ```

7. You will now add the code to set the data in the output message. The order that you copy the data to the output message in could be important . If you have a message type that has a **Type Composition** of **Ordered Set** (this is the default), and you copy the data across in the wrong order, the broker will produce an error if the output format is CWF. If the output format is XML, the broker would not produce an error and the order of the elements would be the order they had been in the message tree.

   You will need to consider this carefully when you decide how to develop your messages. If you want the broker to handle the data in an unordered manner, the Type Composition should be set to Unordered Set.

   Because the Type Composition is Ordered Set the message data needs copying across in the correct order. Add the following to the ESQL code:

   ```
   -- Set the date/time stamp
   SET OutputRoot.MRM.PaxData5.DateTimeStampID = SUBSTRING(CAST(current_>>
   >>timestamp AS CHAR) FROM 12 FOR 23);

   -- Copy across the remaining elements leaving out the OptMsgFormat
   SET I = 2;
   WHILE I <= CARDINALITY(InputBody.PaxData4.*[]) DO
    SET OutputRoot.MRM.PaxData5.*[I] = InputBody.PaxData4.*[I];
    SET I=I+1;
   END WHILE;
   ```

   The first part sets the date time stamp. Note that the DateTimeStamp element is of type STRING, but current_timestamp is of type DATETIME. The CAST command overcomes this issue. In addition the SUBSTRING command removes the data that is not needed from the current_timestamp. (It includes the word TIMESTAMP and quotes.)

8. The last section of code is to set the correct output format. This code is identical to that completed in Chapter 9, “Further message transformation” on page 65. It will read:

   ```
   -- Set the output format for the message - Default is to be CWF
   IF (InputBody.PaxData4.OptMsgFormatID = 'TDS') THEN
       SET OutputRoot.Properties.MessageFormat = 'TDS';
   ELSE
       IF (InputBody.PaxData4.OptMsgFormatID = 'XML') THEN
           SET OutputRoot.Properties.MessageFormat = 'XML';
       END IF;
   END IF;
   ```

9. Click **OK**.

Update the compute node

On completion of these changes the code will read:

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
 SET OutputRoot.*[I] = InputRoot.*[I];
 SET I=I+1;
END WHILE;
-- Enter SQL below this line.  SQL above this line might be regenerated,>>
>> causing any modifications to be lost.**

-- Define the output message that the broker uses to format the output
SET OutputRoot.Properties.MessageType = 'PaxData5ID';

-- Set the date/time stamp
SET OutputRoot.MRM.PaxData5.DateTimeStampID = SUBSTRING(CAST(current_>>
>>timestamp AS CHAR) FROM 12 FOR 23);**

-- Copy across the remaining elements leaving out the OptMsgFormat
SET I = 2;
WHILE I <= CARDINALITY(InputBody.PaxData4.*[]) DO
 SET OutputRoot.MRM.PaxData5.*[I] = InputBody.PaxData4.*[I];
 SET I=I+1;
END WHILE;

-- Set the output format for the message - Default is to be CWF
IF (InputBody.PaxData4.OptMsgFormatID = 'TDS') THEN
    SET OutputRoot.Properties.MessageFormat = 'TDS';
ELSE
    IF (InputBody.PaxData4.OptMsgFormatID = 'XML') THEN
        SET OutputRoot.Properties.MessageFormat = 'XML';
    END IF;
END IF;
```

** The lines have been split at >> for the purposes of displaying in this document. (Do not leave the '**' in the code!)

Assigning, deploying, and testing the message flow

In this chapter, you have created the message set, message, message flow needed for this exercise. In this section you will:

- "Assign the message set to the broker"
- "Assign the message flow to an execution group"
- "Deploy the configuration to the broker"
- "Test the message flow" on page 85

Assign the message set to the broker

Using the instructions in "Assigning a message set to a broker" on page 45 from Chapter 7, "Converting CWF to TDS" on page 41, assign the message set (MRMP_MS_TR2) to the broker (MQSI_SAMPLE_BROKER).

Assign the message flow to an execution group

Use the process "Assigning the message flow to the execution group" on page 26 from Chapter 5, "Creating and testing a basic message flow" on page 21 to add the message flow (MQSI_TEST_TR2) to the broker.

Deploy the configuration to the broker

Use the process "Deploying the message flow to the broker" on page 27 from Chapter 5, "Creating and testing a basic message flow" on page 21 to ensure that

all the changes made to the message sets and the message flows are deployed to the broker for its use in processing messages.

Test the message flow

Use the process “Testing the CWF to TDS message flow” on page 52 from Chapter 7, “Converting CWF to TDS” on page 41 to test the message flow you have created.

Use the input and output message queues that you created in “Create the additional WebSphere MQ resources” on page 81.

The input files MQSI_TR1_CWF.ipt, MQSI_TR1_TDS.ipt, MQSI_TR1_XML.ipt, and MQSI_TR1_Other.ipt that were used in the previous chapter can also be used to help you test the message flow.

You can see the output that is generated from each of these files in Appendix J, “Manipulated output files” on page 115. You will see that the first field that specified the output message format has been replaced with a date and time stamp. Using the OTH input file creates a CWF output format with the date time stamp the only difference.

Test the message flow

Appendix A. Example C header files

The following files are the C header files used in the examples within this document.

Example 1: PaxData1.h

```
/*
*********************************************************
  Passenger Data Test Trees for the C Importer
  Filename: PaxData1.h
  Description:
  Four headers describe four sizes of PaxData test.
  This is the smallest header.

*********************************************************
*/
struct _PaxData1Msg
{
    struct _PaxData1
    {
        char    PaxSurname [20];
        long    PaxMealType;
    } PaxData1;
}PaxData1Msg;
```

Example 2: PaxData2.h

```
/*
*********************************************************
  Passenger Data Test Trees for the C Importer
  Filename: PaxData2.h
  Description:
  Four headers describe four sizes of PaxData test.
  This is the second smallest header.

*********************************************************
*/
struct _PaxData2Msg
{
    struct _PaxData2
    {
        char            PaxSurname [20];
        struct          _PaxFirstName
        {
            char            Pax1stName [20];
        }PaxFirstName;
        unsigned char   PaxTitle[10];
        long            PaxMealType;
        char            PaxMealPreference [20];
    } PaxData2;
}PaxData2Msg;
```

Example 3: PaxData3.h

```
/*
*********************************************************
  Passenger Data Test Trees for the C Importer
  Filename: PaxData3.h
  Description:
  Four headers describe four sizes of PaxData test.
  This is the second largest header.
*********************************************************
*/
struct _PaxData3Msg
{
    struct _PaxData3
    {
        char            PaxSurname [20];
        struct          _PaxFirstName
        {
            char            Pax1stName [20];
        }PaxFirstName;
        unsigned char   PaxTitle[10];
        struct          _PaxRoute
        {
            char            PaxRouteClass [20];
            struct          _PaxDestinationMsg
            {
                char            PaxDestination [20];
                double          PaxDestinationCost;
                char            PaxDestinationStops [3] [10];
            } PaxDestinationMsg;
        }PaxRoute;
        unsigned char   PaxBaggageAllowance[2];
        long            PaxMealType;
        char            PaxMealPreference [20];
    } PaxData3;
}PaxData3Msg;
```

Example 4: PaxData4.h

```
/*
*********************************************************
  Passenger Data Test Trees for the C Importer
  Filename: PaxData4.h
  Description:
  Four headers describe four sizes of PaxData test.
  This is the largest header.
*********************************************************
*/
struct _PaxData4Msg
{
    struct _PaxData4
    {
        char            PaxSurname [20];
        struct          _PaxFirstName
        {
            char            Pax1stName [20];
        }PaxFirstName;
        unsigned char   PaxTitle[10];
        struct          _PaxRoute
        {
            char            PaxRouteClass [20];
            struct          _PaxDestinationMsg
            {
                char            PaxDestination [20];
                double          PaxDestinationCost;
                char            PaxDestinationStops [3] [10];
            } PaxDestinationMsg [4];
        }PaxRoute [2];
        unsigned char   PaxBaggageAllowance[2];
        long            PaxMealType;
        char            PaxMealPreference [20];
        double          PaxTotalCost;
    } PaxData4;
}PaxData4Msg;
```

Example C header files

Appendix B. Example Cobol copybook files

The following files are the Cobol copybook files that you can use in the exercises within this document.

Example 1: PaxData1.cpy

```
*********************************************************
* Passenger Data Test Trees for the Cobol Importer
* Filename: PaxData1.cpy
* Description:
* Four structures describe four sizes of PaxData test.
* This is the smallest structure.
*********************************************************
*
 01  PaxData1Msg.
     03  PaxData1.
         05  PaxSurname                  PIC X(20).
         05  PaxMealtype                 PIC S9(9) COMP-5.
*********************************************************
```

Example 2: PaxData2.cpy

```
*****************************************************
*  Passenger Data Test Trees for the Cobol Importer
*  Filename: PaxData2.cpy
*  Description:
*  Four headers describe four sizes of PaxData test.
*  This is the second smallest header.
*****************************************************
*
 01  PaxData2Msg.
     03  PaxData2.
         05  PaxSurname        PIC X(20).
         05  PaxFirstName.
             07  Pax1stName        PIC X(20).
         05  PaxTitle          PIC X(10).
         05  PaxMealType       PIC S9(9) COMP-5.
         05  PaxMealPreference PIC X(20).
*****************************************************
*
```

Example 3: PaxData3.cpy

```
************************************************
*  Passenger Data Test Trees for the Cobol Importer
*  Filename: PaxData3.cpy
*  Description:
*  Four headers describe four sizes of PaxData test.
*  This is the second largest header.
************************************************
*
 01  PaxData3Msg.
     03  PaxData3.
         05  PaxSurname           PIC X(20).
         05  PaxFirstName.
             10  Pax1stName           PIC X(20).
         05  PaxTitle             PIC X(10).
         05  PaxRoute.
             10  PaxRouteClass        PIC X(20).
             10  PaxDestinationMsg.
                 15  PaxDestination       PIC X(20).
                 15  PaxDestinationCost   COMP-2.
                 15  PaxDestinationStops OCCURS 3 TIMES PIC X(10).
         05  PaxBaggageAllowance  PIC XX.
         05  PaxMealType          PIC S9(9) COMP-5.
         05  PaxMealPreference    PIC X(20).
         05  PaxTotalCost         COMP-2.
************************************************
*
```

Example 4: PaxData4.cpy

```
************************************************
*  Passenger Data Test Trees for the Cobol Importer
*  Filename: PaxData4.cpy
*  Description:
*  Four headers describe four sizes of PaxData test.
*  This is the second largest header.
************************************************
*
 01  PaxData4Msg.
     03  PaxData4.
         05  PaxSurname           PIC X(20).
         05  PaxFirstName.
             10  Pax1stName           PIC X(20).
         05  PaxTitle             PIC X(10).
         05  PaxRoute OCCURS 2 TIMES.
             10  PaxRouteClass        PIC X(20).
             10  PaxDestinationMsg OCCURS 4 TIMES.
                 15  PaxDestination       PIC X(20).
                 15  PaxDestinationCost   COMP-2.
                 15  PaxDestinationStops OCCURS 3 TIMES PIC X(10).
         05  PaxBaggageAllowance  PIC XX.
         05  PaxMealType          PIC S9(9) COMP-5.
         05  PaxMealPreference    PIC X(20).
         05  PaxTotalCost         COMP-2.
************************************************
*
```

Example Cobol copybook files

Appendix C. Example input message files

The following files are the example input message files used in the exercises within this document. The files are formatted as follows:

```
Column 1 Column 2 Column 3 Column 4 Column 5 Column 6 Column 7
00000000 Passenge rSurname 50617373 656E6765 72537572 6E616D65
```

- The first column gives the displacement of the beginning of the data on that display line from the beginning of the file.
- Columns 2 and 3 show the ASCII representation of the data for each line of the display.
- Columns 4 to 7 show the hexadecimal representation of the data for each line of the display

There are no spaces between columns 2 and 3 or columns 4 to 7. The data is displayed this way to assist in reading!

Example 1: PaxData1.ipt

```
00000000 Passenge rSurname 50617373 656E6765 72537572 6E616D65
00000016     ....          20202020 01000000
```

Example 2: PaxData2.ipt

```
00000000 Passenge rSurname 50617373 656E6765 72537572 6E616D65
00000016     Pass engerFir 20202020 50617373 656E6765 72466972
00000032 stname   PaxTitle 73746E61 6D652020 50617854 69746C65
00000048   ....Pa xMealPre 20200100 00005061 784D6561 6C507265
00000064 ference           66657265 6E636520 2020
```

Example 3: PaxData3.ipt

```
00000000 Passenge rSurname 50617373 656E6765 72537572 6E616D65
00000016     Pass engerFir 20202020 50617373 656E6765 72466972
00000032 stname   PaxTitle 73746E61 6D652020 50617854 69746C65
00000048   Passen gerRoute 20205061 7373656E 67657252 6F757465
00000064 Class Pa ssengerD 436C6173 73205061 7373656E 67657244
00000080 estinati onU0*..I 65737469 6E617469 6F6E5530 2AA9D349
00000096 .@PaxSto pOnePaxS 93405061 7853746F 704F6E65 50617853
00000112 topTwoPa xDestThr 746F7054 776F5061 78446573 74546872
00000128 BA....Pa xMealPre 42410100 00005061 784D6561 6C507265
00000144 ference           66657265 6E636520 2020
```

Example 4: PaxData4.ipt

```
00000000 Passenge rSurname 50617373 656E6765 72537572 6E616D65
00000016     Pass engerFir 20202020 50617373 656E6765 72466972
00000032 stname   PaxTitle 73746E61 6D652020 50617854 69746C65
00000048   Passen gerRoute 20205061 7373656E 67657252 6F757465
00000064 Class Pa ssengerD 436C6173 73205061 7373656E 67657244
00000080 estinat1 -1U0*..I 65737469 6E617431 2D315530 2AA9D349
00000096 .@PaxSto p1-1PaxS 93405061 7853746F 70312D31 50617853
00000112 top1-2Pa xStop1-3 746F7031 2D325061 7853746F 70312D33
00000128 Passenge rDestina 50617373 656E6765 72446573 74696E61
00000144 t1-2U0*. .I.@PaxS 74312D32 55302AA9 D3499340 50617853
00000160 top2-1Pa xStop2-2 746F7032 2D315061 7853746F 70322D32
00000176 PaxStop2 -3Passen 50617853 746F7032 2D335061 7373656E
00000192 gerDesti nat1-3U0 67657244 65737469 6E617431 2D335530
```

Example input message files

```
00000208 *..I.@Pa xStop3-1 2AA9D349 93405061 7853746F 70332D31
00000224 PaxStop3 -2PaxSto 50617853 746F7033 2D325061 7853746F
00000240 p3-3Pass engerDes 70332D33 50617373 656E6765 72446573
00000256 tinat1-4 U0*..I.@ 74696E61 74312D34 55302AA9 D3499340
00000272 PaxStop4 -1PaxSto 50617853 746F7034 2D315061 7853746F
00000288 p4-2PaxS top4-3Pa 70342D32 50617853 746F7034 2D335061
00000304 ssengerR outeClas 7373656E 67657252 6F757465 436C6173
00000320 s Passen gerDesti 73205061 7373656E 67657244 65737469
00000336 nat2-1U0 *..I.@Pa 6E617432 2D315530 2AA9D349 93405061
00000352 xStop1-1 PaxStop1 7853746F 70312D31 50617853 746F7031
00000368 -2PaxSto p1-3Pass 2D325061 7853746F 70312D33 50617373
00000384 engerDes tinat2-2 656E6765 72446573 74696E61 74322D32
00000400 U0*..I.@ PaxStop2 55302AA9 D3499340 50617853 746F7032
00000416 -1PaxSto p2-2PaxS 2D315061 7853746F 70322D32 50617853
00000432 top2-3Pa ssengerD 746F7032 2D335061 7373656E 67657244
00000448 estinat2 -3U0*..I 65737469 6E617432 2D335530 2AA9D349
00000464 .@PaxSto p3-1PaxS 93405061 7853746F 70332D31 50617853
00000480 top3-2Pa xStop3-3 746F7033 2D325061 7853746F 70332D33
00000496 Passenge rDestina 50617373 656E6765 72446573 74696E61
00000512 t2-4U0*. .I.@PaxS 74322D34 55302AA9 D3499340 50617853
00000528 top4-1Pa xStop4-2 746F7034 2D315061 7853746F 70342D32
00000544 PaxStop4 -3BA.... 50617853 746F7034 2D334241 01000000
00000560 PaxMealP referenc 5061784D 65616C50 72656665 72656E63
00000576 e   U0*. .I.@     65202020 55302AA9 D3499340
```

Appendix D. Example TDS output

The following output examples show the TDS data that is generated from the input files in Appendix C, "Example input message files" on page 95.

Note: If you have imported Cobol copybooks, review "C header files and Cobol copybooks" on page xiii. You will find that if you have left the padding character set to SPACE for STRING elements, the blanks at the end of those input message fields will not be in the output message.

The data is formatted as follows:

```
Column 1 Column 2
00000000 PassengerSurname    |PassengerFi
```

- The first column gives the displacement (in hexadecimal) of the beginning of the data on that display line from the beginning of the file.
- Columns 2 displays the data for that line

Example 1: From PaxData1.ipt

```
00000000 PassengerSurname    |1
```

Example 2: From PaxData2.ipt

```
00000000 PassengerSurname    |PassengerFi
00000032 rstname  |PaxTitle  |1|PaxMealPr
00000064 eference
```

Example 3: From PaxData3.ipt

```
00000000 PassengerSurname    |PassengerFi
00000032 rstname  |PaxTitle  |PassengerRo
00000064 uteClass |PassengerDestination|1
00000096 234.4567|PaxStopOne||PaxStopTwo|
00000128 |PaxDestThr|BA|1|PaxMealPreferen
00000160 ce
```

Example 4: From PaxData4.ipt

```
00000000 PassengerSurname    |PassengerFi
00000032 rstname  |PaxTitle  |PassengerRo
00000064 uteClass |PassengerDestinat1-1|1
00000096 234.4567|PaxStop1-1||||PaxStop1-
00000128 2||||PaxStop1-3|||PassengerDesti
00000160 nat1-2|1234.4567|PaxStop2-1||||P
00000192 axStop2-2||||PaxStop2-3|||Passen
00000224 gerDestinat1-3|1234.4567|PaxStop
00000256 3-1||||PaxStop3-2||||PaxStop3-3|
00000288 ||PassengerDestinat1-4|1234.4567
00000320 |PaxStop4-1||||PaxStop4-2||||Pax
00000352 Stop4-3||PassengerRouteClass |Pa
00000384 ssengerDestinat2-1|1234.4567|Pax
00000416 Stop1-1||||PaxStop1-2||||PaxStop
00000448 1-3|||PassengerDestinat2-2|1234.
00000480 4567|PaxStop2-1||||PaxStop2-2|||
00000512 |PaxStop2-3|||PassengerDestinat2
00000544 -3|1234.4567|PaxStop3-1||||PaxSt
```

Example TDS output

```
00000576 op3-2||||PaxStop3-3|||PassengerD
00000608 estinat2-4|1234.4567|PaxStop4-1|
00000640 |||PaxStop4-2||||PaxStop4-3|BA|1
00000672 |PaxMealPreference   |1234.4567
```

Appendix E. .XML trace file

This is the XML file that was created by the **mqsireadlog** command in Chapter 6, "The Trace node" on page 35. For the purposes of this document, it has been necessary to split some of the lines at '>>'.

```
<:?xml version="1.0" encoding="UTF-8" ?><UserTraceLog uuid="UserTraceLo>>
>>g" userTraceLevel="none" traceLevel="none" userTraceFilter="debugTrace" t>>
>>raceFilter="none" fileSize="4194304" bufferSize="0" fileMode="safe"><Er>>
>>ror timestamp='2002-07-02 08:26:07.295000' thread='2628' function='ImbLibra>>
>>ry::ImbLibrary' text='Failed to load library file' catalog='WMQIv210' numbe>>
>>r='2308' file='F:\build\S210_P\src\DataFlowEngine\DataFlowDLL\Win32\ImbLibr>>
>>ary.cpp' line='83'><Insert type='string'>C:\IBM\WMQI\bin\imbdfneo.lil>>
>></Insert><Insert type='integer'>126</Insert></Error><Erro>>
>>r timestamp='2002-07-02 08:26:13.654000' thread='2628' function='ImbLibrary>>
>>::ImbLibrary' text='Failed to load library file' catalog='WMQIv210' number=>>

... 665 lines deleted for the purposes of this document

>>IbmTraceNode' name='6bde51c6-ee00-0000-0080-df695815836b' label='MQSI_TEST.>>
>>Trace1' text='Evaluating expression at (&1, &2)' catalog='WMQIv210'>>
>> number='2538' file='F:\build\S210_P\src\DataFlowEngine\ImbRdl\ImbRdlFieldR>>
>>ef.cpp' line='1234'><Insert type='integer'>1</Insert><Insert ty>>
>>pe='integer'>3</Insert><Insert type='string'>Root</Insert><>>
>>Insert type='string'>MQSI_TEST.Trace1</Insert></UserTrace><User>>
>>Trace timestamp='2002-07-02 08:32:53.048000' thread='764' function='ImbTrac>>
>>eNode::writeToLog' type='ComIbmTraceNode' name='6bde51c6-ee00-0000-0080-df6>>
>>95815836b' label='MQSI_TEST.Trace1' text='Application trace output from Tra>>
>>ceNode' catalog='WMQIv210' number='4060' file='F:\build\S210_P\src\DataFlow>>
>>Engine\ImbTraceNode.cpp' line='426'><Insert type='string'>(
  (0x1000000)Properties = (
    (0x3000000)MessageSet      = 'DPQ898C072001'
    (0x3000000)MessageType     = 'PaxData1Msg'
    (0x3000000)MessageFormat   = 'CWF'
    (0x3000000)Encoding        = 546
    (0x3000000)CodedCharSetId  = 437
    (0x3000000)Transactional   = TRUE
    (0x3000000)Persistence     = FALSE
    (0x3000000)CreationTime    = GMTTIMESTAMP '2002-07-02 08:32:52.750'
    (0x3000000)ExpirationTime  = -1
    (0x3000000)Priority        = 0
    (0x3000000)ReplyIdentifier = X'00000000000000000000000000000000000000>>
    >>00000000000'
    (0x3000000)ReplyProtocol   = 'MQ'
    (0x3000000)Topic           = NULL
  )
  (0x1000000)MQMD       = (
    (0x3000000)SourceQueue      = 'MQSI_INQ'
    (0x3000000)Transactional    = TRUE
    (0x3000000)Encoding         = 546
    (0x3000000)CodedCharSetId   = 437
    (0x3000000)Format           = '        '
    (0x3000000)Version          = 2
    (0x3000000)Report           = 0
    (0x3000000)MsgType          = 8
    (0x3000000)Expiry           = -1
    (0x3000000)Feedback         = 0
    (0x3000000)Priority         = 0
    (0x3000000)Persistence      = 0
    (0x3000000)MsgId            = X'414d51204d5153495f53414d504c455f2163>>
    >>213d12900000'
    (0x3000000)CorrelId         = X'00000000000000000000000000000000000000>>
    >>000000000000'
```

XML trace file

```
    (0x3000000)BackoutCount     = 0
    (0x3000000)ReplyToQ         = '                                        >>
    >>          '
    (0x3000000)ReplyToQMgr      = 'MQSI_SAMPLE_QM                          >>
    >>          '
    (0x3000000)UserIdentifier   = 'ZZ112233    '
    (0x3000000)AccountingToken  = X'16010515000000c373de18cb4a5f7d4738b7>>
    >>5ff4010000000000000000000000b'
    (0x3000000)ApplIdentityData = '                                    '
    (0x3000000)PutApplType      = 11
    (0x3000000)PutApplName      = 'C:\IBM\IH03\rfhutil.exe     '
    (0x3000000)PutDate          = DATE '2002-07-02'
    (0x3000000)PutTime          = GMTTIME '08:32:52.750'
    (0x3000000)ApplOriginData   = '    '
    (0x3000000)GroupId          = X'000000000000000000000000000000000000>>
    >>000000000000'
    (0x3000000)MsgSeqNumber     = 1
    (0x3000000)Offset           = 0
    (0x3000000)MsgFlags         = 0
    (0x3000000)OriginalLength   = -1
  )
  (0x1000021)MRM        = (
    (0x1000013)PaxData1 = (
      (0x300000B)PaxSurname  = 'PassengerSurname    '
      (0x300000B)PaxMealType = 1
    )
  )
)
</Insert><Insert type='string'>MQSI_TEST.Trace1</Insert></UserT>>
>>race><UserTrace timestamp='2002-07-02 08:32:53.048000' thread='764' fun>>
>>ction='ImbTraceNode::evaluate' type='ComIbmTraceNode' name='6bde51c6-ee00-0>>
>>000-0080-df695815836b' label='MQSI_TEST.Trace1' text='Propagating to output>>
>> terminal' catalog='WMQIv210' number='4067' file='F:\build\S210_P\src\DataF>>
>>lowEngine\ImbTraceNode.cpp' line='337'><Insert type='string'>MQSI_TES>>
>>T.Trace1</Insert></UserTrace><UserTrace timestamp='2002-07-02 08:>>
>>32:53.048000' thread='764' function='ImbMqOutputNode::putMessage' type='Com>>
>>IbmMQOutputNode' name='07241a17-ee00-0000-0080-df695815836b' label='MQSI_TE>>
>>ST.MQOutput1' text='MQPUT issued to put message to the specified output que>>

... 111 lines deleted for the purposes of this document

>> type='integer'>364</Insert><Insert type='string'>MQHMD</Insert>>
>>><Insert type='string'>ConfigurationMessageFlow.InputNode</Insert>>>
>></UserTrace><UserTrace timestamp='2002-07-02 08:36:20.096000' thread>>
>>='2692' function='ImbRootParser::parseNextItem' type='ComIbmMQInputNode' na>>
>>me='InputNode' label='ConfigurationMessageFlow.InputNode' text='Created par>>
>>ser' catalog='WMQIv210' number='6061' file='F:\build\S210_P\src\DataFlowEng>>
>>ine\ImbRootParser.cpp' line='523'><Insert type='string'>XML</Insert>>
>>><Insert type='integer'>364</Insert><Insert type='integer'>24>>
>>1</Insert><Insert type='string'>XML</Insert><Insert type='str>>
>>ing'>ConfigurationMessageFlow.InputNode</Insert></UserTrace></U>>
>>serTraceLog>
```

Appendix F. .log trace file

This is the formatted log file that was created by the **mqsiformatlog** command in Chapter 6, "The Trace node" on page 35.

Some of the lines in this Trace file have been split at '>>' for the purposes of displaying in this document.

```
Timestamps are formatted in local time, 60 minutes past GMT.

2002-07-02 09:32:25.048000     2692   UserTrace   BIP2632I: Message received and propagated to 'out'>>
>> terminal of MQ input node node 'ConfigurationMessageFlow.InputNode'.
2002-07-02 09:32:25.048000     2692   UserTrace   BIP6060I: Parser type 'Properties' created on beh>>
>>alf of node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length >>
>>0 bytes beginning at offset '0'.
2002-07-02 09:32:25.048000     2692   UserTrace   BIP6061I: Parser type 'MQMD' created on behalf of>>
>> node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length '364' >>
>>bytes beginning at offset '0'. Parser type selected based on value 'MQHMD' from previous parser.
2002-07-02 09:32:25.048000     2692   UserTrace   BIP6061I: Parser type 'XML' created on behalf of >>
>>node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length '241' b>>
>>ytes beginning at offset '364'. Parser type selected based on value 'XML' from previous parser.
2002-07-02 09:32:29.795000     2692   UserTrace   BIP2265I: Attribute in message flow 'MQSI_TEST' (>>
>>uuid='0b870817-ee00-0000-0080-df695815836b') successfully changed.
                                      The message broker received a configuration message containi>>
>>ng an instruction to change an attribute in the message flow 'MQSI_TEST' (uuid='0b870817-ee00-000>>
>>0-0080-df695815836b') and successfully performed this action.
                                      No user action required.
2002-07-02 09:32:29.954999     2692   UserTrace   BIP4040I: The Execution Group 'default' has proce>>
>>ssed a configuration message successfully.
                                      A configuration message has been processed successfully. Any>>
>> configuration changes have been made and stored persistently.
                                      No user action required.
2002-07-02 09:32:29.995000     2692   UserTrace   BIP2638I: The MQ output node 'ConfigurationMessag>>
>>eFlow.outputNode' attempted to write a message to the specified queue 'SYSTEM.BROKER.EXECUTIONGRO>>
>>UP.REPLY' connected to queue manager 'MQSI_SAMPLE_QM'.  The MQCC was 0 and the MQRC was 0.
2002-07-02 09:32:29.995000     2692   UserTrace   BIP2622I: Message successfully output by MQ outpu>>
>>t node 'ConfigurationMessageFlow.outputNode' to queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on que>>
>>ue manager 'MQSI_SAMPLE_QM'.
2002-07-02 09:32:52.757999      764   UserTrace   BIP2632I: Message received and propagated to 'out'>>
>> terminal of MQ input node node 'MQSI_TEST.MQInput1'.
2002-07-02 09:32:52.757999      764   UserTrace   BIP6060I: Parser type 'Properties' created on beh>>
>>alf of node 'MQSI_TEST.MQInput1' to handle portion of incoming message of length 0 bytes beginnin>>
>>g at offset '0'.
2002-07-02 09:32:52.757999      764   UserTrace   BIP6061I: Parser type 'MQMD' created on behalf of>>
>> node 'MQSI_TEST.MQInput1' to handle portion of incoming message of length '364' bytes beginning a>>
>>t offset '0'. Parser type selected based on value 'MQHMD' from previous parser.
2002-07-02 09:32:52.958000      764   UserTrace   BIP6061I: Parser type 'MRM' created on behalf of >>
>>node 'MQSI_TEST.MQInput1' to handle portion of incoming message of length '24' bytes beginning at>>
>> offset '364'. Parser type selected based on value 'MRM' from previous parser.
2002-07-02 09:32:52.998001      764   UserTrace   BIP2538I: Node 'MQSI_TEST.Trace1': Evaluating exp>>
>>ression 'Root' at (1, 3).
2002-07-02 09:32:53.048000      764   UserTrace   BIP4060I: Data '(
                                        (0x1000000)Properties = (
                                          (0x3000000)MessageSet      = 'DPQ898C072001'
                                          (0x3000000)MessageType     = 'PaxData1Msg'
                                          (0x3000000)MessageFormat   = 'CWF'
                                          (0x3000000)Encoding        = 546
                                          (0x3000000)CodedCharSetId  = 437
                                          (0x3000000)Transactional   = TRUE
                                          (0x3000000)Persistence     = FALSE
                                          (0x3000000)CreationTime    = GMTTIMESTAMP '2002-07-02 08>>
                                          >>:32:52.750'
                                          (0x3000000)ExpirationTime  = -1
```

```
                                    (0x3000000)Priority        = 0
                                    (0x3000000)ReplyIdentifier = X'00000000000000000000000000>>
                                    >>000000000000000000000000'
                                    (0x3000000)ReplyProtocol   = 'MQ'
                                    (0x3000000)Topic           = NULL
                                  )
                                  (0x1000000)MQMD       = (
                                    (0x3000000)SourceQueue      = 'MQSI_INQ'
                                    (0x3000000)Transactional    = TRUE
                                    (0x3000000)Encoding         = 546
                                    (0x3000000)CodedCharSetId   = 437
                                    (0x3000000)Format           = '        '
                                    (0x3000000)Version          = 2
                                    (0x3000000)Report           = 0
                                    (0x3000000)MsgType          = 8
                                    (0x3000000)Expiry           = -1
                                    (0x3000000)Feedback         = 0
                                    (0x3000000)Priority         = 0
                                    (0x3000000)Persistence      = 0
                                    (0x3000000)MsgId            = X'414d51204d5153495f53414d>>
                                    >>504c455f2163213d12900000'
                                    (0x3000000)CorrelId         = X'00000000000000000000000000>>
                                    >>000000000000000000000000'
                                    (0x3000000)BackoutCount     = 0
                                    (0x3000000)ReplyToQ         = '                          '
                                    (0x3000000)ReplyToQMgr      = 'MQSI_SAMPLE_QM            '
                                    (0x3000000)UserIdentifier   = 'GB028334    '
                                    (0x3000000)AccountingToken  = X'16010515000000c373de18cb>>
                                    >>4a5f7d4738b75ff401000000000000000000000b'
                                    (0x3000000)ApplIdentityData = '                        '
                                    (0x3000000)PutApplType      = 11
                                    (0x3000000)PutApplName      = 'C:\IBM\IH03\rfhutil.exe'
                                    (0x3000000)PutDate          = DATE '2002-07-02'
                                    (0x3000000)PutTime          = GMTTIME '08:32:52.750'
                                    (0x3000000)ApplOriginData   = '    '
                                    (0x3000000)GroupId          = X'00000000000000000000000000>>
                                    >>000000000000000000000000'
                                    (0x3000000)MsgSeqNumber     = 1
                                    (0x3000000)Offset           = 0
                                    (0x3000000)MsgFlags         = 0
                                    (0x3000000)OriginalLength   = -1
                                  )
                                  (0x1000021)MRM        = (
                                    (0x1000013)PaxData1 = (
                                      (0x300000B)PaxSurname  = 'PassengerSurname    '
                                      (0x300000B)PaxMealType = 1
                                    )
                                  )
                                )
                                ' from trace node 'MQSI_TEST.Trace1'.
                                The trace node 'MQSI_TEST.Trace1' has output the specified tr>>
                                >>ace data.
                                This is an information message provided by the message flow >>
                                >>designer.  The user response will be determined by the loc>>
                                >>al environment.
2002-07-02 09:32:53.048000      764   UserTrace   BIP4067I: Message propagated to output terminal f>>
>>or trace node 'MQSI_TEST.Trace1'.
                                The trace node 'MQSI_TEST.Trace1' has received a message and >>
>>is propagating it to any nodes connected to its output terminal.
                                No user action required.
2002-07-02 09:32:53.048000      764   UserTrace   BIP2638I: The MQ output node 'MQSI_TEST.MQOutput1>>
>>' attempted to write a message to the specified queue 'MQSI_OUTQ' connected to queue manager 'MQS>>
>>I_SAMPLE_QM'.  The MQCC was 0 and the MQRC was 0.
2002-07-02 09:32:53.048000      764   UserTrace   BIP2622I: Message successfully output by MQ outpu>>
>>t node 'MQSI_TEST.MQOutput1' to queue 'MQSI_OUTQ' on queue manager 'MQSI_SAMPLE_QM'.
2002-07-02 09:34:02.338000     2692   UserTrace   BIP2632I: Message received and propagated to 'out'>>
>> terminal of MQ input node node 'ConfigurationMessageFlow.InputNode'.
```

```
2002-07-02 09:34:02.338000     2692   UserTrace   BIP6060I: Parser type 'Properties' created on beh>>
>>alf of node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length >>
>>0 bytes beginning at offset '0'.
2002-07-02 09:34:02.338000     2692   UserTrace   BIP6061I: Parser type 'MQMD' created on behalf of>>
>> node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length '364' >>
>>bytes beginning at offset '0'. Parser type selected based on value 'MQHMD' from previous parser.
2002-07-02 09:34:02.338000     2692   UserTrace   BIP6061I: Parser type 'XML' created on behalf of >>
>>node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length '235' b>>
>>ytes beginning at offset '364'. Parser type selected based on value 'XML' from previous parser.
2002-07-02 09:34:03.069000     2692   UserTrace   BIP2265I: Attribute in message flow 'MQSI_TEST' (>>
>>uuid='0b870817-ee00-0000-0080-df695815836b') successfully changed.
                                      The message broker received a configuration message containi>>
>>ng an instruction to change an attribute in the message flow 'MQSI_TEST' (uuid='0b870817-ee00-000>>
>>0-0080-df695815836b') and successfully performed this action.
                                      No user action required.
2002-07-02 09:34:03.108999     2692   UserTrace   BIP4040I: The Execution Group 'default' has proce>>
>>ssed a configuration message successfully.
                                      A configuration message has been processed successfully. Any>>
>> configuration changes have been made and stored persistently.
                                      No user action required.
2002-07-02 09:34:03.108999     2692   UserTrace   BIP2638I: The MQ output node 'ConfigurationMessag>>
>>eFlow.outputNode' attempted to write a message to the specified queue 'SYSTEM.BROKER.EXECUTIONGRO>>
>>UP.REPLY' connected to queue manager 'MQSI_SAMPLE_QM'.  The MQCC was 0 and the MQRC was 0.
2002-07-02 09:34:03.108999     2692   UserTrace   BIP2622I: Message successfully output by MQ outpu>>
>>t node 'ConfigurationMessageFlow.outputNode' to queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on que>>
>>ue manager 'MQSI_SAMPLE_QM'.
2002-07-02 09:36:20.096000     2692   UserTrace   BIP2632I: Message received and propagated to 'out>>
>>' terminal of MQ input node node 'ConfigurationMessageFlow.InputNode'.
2002-07-02 09:36:20.096000     2692   UserTrace   BIP6060I: Parser type 'Properties' created on beh>>
>>alf of node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length >>
>>0 bytes beginning at offset '0'.
2002-07-02 09:36:20.096000     2692   UserTrace   BIP6061I: Parser type 'MQMD' created on behalf of>>
>> node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length '364' >>
>>bytes beginning at offset '0'. Parser type selected based on value 'MQHMD' from previous parser.
2002-07-02 09:36:20.096000     2692   UserTrace   BIP6061I: Parser type 'XML' created on behalf of >>
>>node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming message of length '241' b>>
>>ytes beginning at offset '364'. Parser type selected based on value 'XML' from previous parser.

Threads encountered in this trace:
  2692
```

log trace file

Appendix G. Example CWF import report file

The following file was generated by importing PaxData4.h (See Appendix A, "Example C header files" on page 87) into a message set.

The warning that is at the top of the file is generated because PaxData4.h is imported before the CWF layer was added. This is discussed further in Chapter 4, "Importing C or Cobol data structures into the MRM" on page 15.

```
Importing C header file C:\PaxData4.h...


Warning: The Custom Wire Format Plugin is not installed and registered for the project.
  No Custom Wire Format values will be set as a result

Created Value DEFAULT_CHAR_VALUE.
Importing C structure PaxData4Msg.
Importing C structure PaxData4.
Created Value PAXSURNAME_VALUE.
Created Element PaxSurname.
Inserted Value PAXSURNAME_VALUE into Element PaxSurname.
Importing C structure PaxFirstName.
Created Value PAX1STNAME_VALUE.
Created Element Pax1stName.
Inserted Value PAX1STNAME_VALUE into Element Pax1stName.
Created Type PaxFirstName_TYPE.
Inserted Element Pax1stName into Type PaxFirstName_TYPE.
Created Element PaxFirstName.
Created Element PaxTitle.
Importing C structure PaxRoute.
Created Value PAXROUTECLASS_VALUE.
Created Element PaxRouteClass.
Inserted Value PAXROUTECLASS_VALUE into Element PaxRouteClass.
Importing C structure PaxDestinationMsg.
Created Value PAXDESTINATION_VALUE.
Created Element PaxDestination.
Inserted Value PAXDESTINATION_VALUE into Element PaxDestination.
Created Element PaxDestinationCost.
Created Value PAXDESTINATIONSTOPS_VALUE.
Created Element PaxDestinationStops.
Inserted Value PAXDESTINATIONSTOPS_VALUE into Element PaxDestinationStops.
Created Type PaxDestinationMsg_TYPE.
Inserted Element PaxDestination into Type PaxDestinationMsg_TYPE.
Inserted Element PaxDestinationCost into Type PaxDestinationMsg_TYPE.
Inserted Element PaxDestinationStops into Type PaxDestinationMsg_TYPE.
Created Element PaxDestinationMsg.
Created Type PaxRoute_TYPE.
Inserted Element PaxRouteClass into Type PaxRoute_TYPE.
Inserted Element PaxDestinationMsg into Type PaxRoute_TYPE.
Created Element PaxRoute.
Created Element PaxBaggageAllowance.
Created Element PaxMealType.
Created Value PAXMEALPREFERENCE_VALUE.
Created Element PaxMealPreference.
Inserted Value PAXMEALPREFERENCE_VALUE into Element PaxMealPreference.
Created Element PaxTotalCost.
Created Type PaxData4_TYPE.
Inserted Element PaxSurname into Type PaxData4_TYPE.
Inserted Element PaxFirstName into Type PaxData4_TYPE.
Inserted Element PaxTitle into Type PaxData4_TYPE.
Inserted Element PaxRoute into Type PaxData4_TYPE.
Inserted Element PaxBaggageAllowance into Type PaxData4_TYPE.
```

Example CWF import report file

```
Inserted Element PaxMealType into Type PaxData4_TYPE.
Inserted Element PaxMealPreference into Type PaxData4_TYPE.
Inserted Element PaxTotalCost into Type PaxData4_TYPE.
Created Element PaxData4.
Created Type PaxData4Msg_TYPE.
Inserted Element PaxData4 into Type PaxData4Msg_TYPE.
C Structure PaxData4Msg successfully imported.

Summary:
No. of Errors:   0
No. of Types created:  5
No. of Elements created: 15
No. of Values created:  7
```

Appendix H. Variable conversion input files

The following files are those that have been created to test the message flow created in Chapter 9, "Further message transformation" on page 65. These files are all based on the PaxData4.ipt file used earlier in this document with an additional field of 3 characters being added at the beginning to define the output format.

The files are formatted as follows:

```
Column 1 Column 2 Column 3 Column 4 Column 5 Column 6 Column 7
00000000 Passenge rSurname 50617373 656E6765 72537572 6E616D65
```

- The first column gives the displacement of the beginning of the data on that display line from the beginning of the file.
- Columns 2 and 3 show the ASCII representation of the data for each line of the display.
- Columns 4 to 7 show the hexadecimal representation of the data for each line of the display

Example 1: CWF input file (PaxData4_TR1_CWF.ipt)

```
00000000 CWFPasse ngerSurn 43574650 61737365 6E676572 5375726E
00000016 ame    P assenger 616D6520 20202050 61737365 6E676572
00000032 Firstnam e  PaxTi 46697273 746E616D 65202050 61785469
00000048 tle  Pas sengerRo 746C6520 20506173 73656E67 6572526F
00000064 uteClass  Passeng 75746543 6C617373 20506173 73656E67
00000080 erDestin at1-1U0* 65724465 7374696E 6174312D 3155302A
00000096 ..I.@Pax Stop1-1P A9D34993 40506178 53746F70 312D3150
00000112 axStop1- 2PaxStop 61785374 6F70312D 32506178 53746F70
00000128 1-3Passe ngerDest 312D3350 61737365 6E676572 44657374
00000144 inat1-2U 0*..I.@P 696E6174 312D3255 302AA9D3 49934050
00000160 axStop2- 1PaxStop 61785374 6F70322D 31506178 53746F70
00000176 2-2PaxSt op2-3Pas 322D3250 61785374 6F70322D 33506173
00000192 sengerDe stinat1- 73656E67 65724465 7374696E 6174312D
00000208 3U0*..I. @PaxStop 3355302A A9D34993 40506178 53746F70
00000224 3-1PaxSt op3-2Pax 332D3150 61785374 6F70332D 32506178
00000240 Stop3-3P assenger 53746F70 332D3350 61737365 6E676572
00000256 Destinat 1-4U0*.. 44657374 696E6174 312D3455 302AA9D3
00000272 I.@PaxSt op4-1Pax 49934050 61785374 6F70342D 31506178
00000288 Stop4-2P axStop4- 53746F70 342D3250 61785374 6F70342D
00000304 3Passeng erRouteC 33506173 73656E67 6572526F 75746543
00000320 lass Pas sengerDe 6C617373 20506173 73656E67 65724465
00000336 stinat2- 1U0*..I. 7374696E 6174322D 3155302A A9D34993
00000352 @PaxStop 1-1PaxSt 40506178 53746F70 312D3150 61785374
00000368 op1-2Pax Stop1-3P 6F70312D 32506178 53746F70 312D3350
00000384 assenger Destinat 61737365 6E676572 44657374 696E6174
00000400 2-2U0*.. I.@PaxSt 322D3255 302AA9D3 49934050 61785374
00000416 op2-1Pax Stop2-2P 6F70322D 31506178 53746F70 322D3250
00000432 axStop2- 3Passeng 61785374 6F70322D 33506173 73656E67
00000448 erDestin at2-3U0* 65724465 7374696E 6174322D 3355302A
00000464 ..I.@Pax Stop3-1P A9D34993 40506178 53746F70 332D3150
00000480 axStop3- 2PaxStop 61785374 6F70332D 32506178 53746F70
00000496 3-3Passe ngerDest 332D3350 61737365 6E676572 44657374
00000512 inat2-4U 0*..I.@P 696E6174 322D3455 302AA9D3 49934050
00000528 axStop4- 1PaxStop 61785374 6F70342D 31506178 53746F70
00000544 4-2PaxSt op4-3BA. 342D3250 61785374 6F70342D 33424101
00000560 ...PaxMe alPrefer 00000050 61784D65 616C5072 65666572
00000576 ence   U 0*..I.@  656E6365 20202055 302AA9D3 499340
```

Example 2: TDS input file (PaxData4_TR1_TDS.ipt)

```
00000000 TDSPasse ngerSurn 54445350 61737365 6E676572 5375726E
00000016 ame    P assenger 616D6520 20202050 61737365 6E676572
00000032 Firstnam e  PaxTi 46697273 746E616D 65202050 61785469
00000048 tle  Pas sengerRo 746C6520 20506173 73656E67 6572526F
00000064 uteClass  Passeng 75746543 6C617373 20506173 73656E67
00000080 erDestin at1-1U0* 65724465 7374696E 6174312D 3155302A
00000096 ..I.@Pax Stop1-1P A9D34993 40506178 53746F70 312D3150
00000112 axStop1- 2PaxStop 61785374 6F70312D 32506178 53746F70
00000128 1-3Passe ngerDest 312D3350 61737365 6E676572 44657374
00000144 inat1-2U 0*..I.@P 696E6174 312D3255 302AA9D3 49934050
00000160 axStop2- 1PaxStop 61785374 6F70322D 31506178 53746F70
00000176 2-2PaxSt op2-3Pas 322D3250 61785374 6F70322D 33506173
00000192 sengerDe stinat1- 73656E67 65724465 7374696E 6174312D
00000208 3U0*..I. @PaxStop 3355302A A9D34993 40506178 53746F70
00000224 3-1PaxSt op3-2Pax 332D3150 61785374 6F70332D 32506178
00000240 Stop3-3P assenger 53746F70 332D3350 61737365 6E676572
00000256 Destinat 1-4U0*.. 44657374 696E6174 312D3455 302AA9D3
00000272 I.@PaxSt op4-1Pax 49934050 61785374 6F70342D 31506178
00000288 Stop4-2P axStop4- 53746F70 342D3250 61785374 6F70342D
00000304 3Passeng erRouteC 33506173 73656E67 6572526F 75746543
00000320 lass Pas sengerDe 6C617373 20506173 73656E67 65724465
00000336 stinat2- 1U0*..I. 7374696E 6174322D 3155302A A9D34993
00000352 @PaxStop 1-1PaxSt 40506178 53746F70 312D3150 61785374
00000368 op1-2Pax Stop1-3P 6F70312D 32506178 53746F70 312D3350
00000384 assenger Destinat 61737365 6E676572 44657374 696E6174
00000400 2-2U0*.. I.@PaxSt 322D3255 302AA9D3 49934050 61785374
00000416 op2-1Pax Stop2-2P 6F70322D 31506178 53746F70 322D3250
00000432 axStop2- 3Passeng 61785374 6F70322D 33506173 73656E67
00000448 erDestin at2-3U0* 65724465 7374696E 6174322D 3355302A
00000464 ..I.@Pax Stop3-1P A9D34993 40506178 53746F70 332D3150
00000480 axStop3- 2PaxStop 61785374 6F70332D 32506178 53746F70
00000496 3-3Passe ngerDest 332D3350 61737365 6E676572 44657374
00000512 inat2-4U 0*..I.@P 696E6174 322D3455 302AA9D3 49934050
00000528 axStop4- 1PaxStop 61785374 6F70342D 31506178 53746F70
00000544 4-2PaxSt op4-3BA. 342D3250 61785374 6F70342D 33424101
00000560 ...PaxMe alPrefer 00000050 61784D65 616C5072 65666572
00000576 ence   U 0*..I.@  656E6365 20202055 302AA9D3 499340
```

Example 3: XML input file (PaxData4_TR1_XML.ipt)

```
00000000 XMLPasse ngerSurn 584D4C50 61737365 6E676572 5375726E
00000016 ame    P assenger 616D6520 20202050 61737365 6E676572
00000032 Firstnam e  PaxTi 46697273 746E616D 65202050 61785469
00000048 tle  Pas sengerRo 746C6520 20506173 73656E67 6572526F
00000064 uteClass  Passeng 75746543 6C617373 20506173 73656E67
00000080 erDestin at1-1U0* 65724465 7374696E 6174312D 3155302A
00000096 ..I.@Pax Stop1-1P A9D34993 40506178 53746F70 312D3150
00000112 axStop1- 2PaxStop 61785374 6F70312D 32506178 53746F70
00000128 1-3Passe ngerDest 312D3350 61737365 6E676572 44657374
00000144 inat1-2U 0*..I.@P 696E6174 312D3255 302AA9D3 49934050
00000160 axStop2- 1PaxStop 61785374 6F70322D 31506178 53746F70
00000176 2-2PaxSt op2-3Pas 322D3250 61785374 6F70322D 33506173
00000192 sengerDe stinat1- 73656E67 65724465 7374696E 6174312D
00000208 3U0*..I. @PaxStop 3355302A A9D34993 40506178 53746F70
00000224 3-1PaxSt op3-2Pax 332D3150 61785374 6F70332D 32506178
00000240 Stop3-3P assenger 53746F70 332D3350 61737365 6E676572
00000256 Destinat 1-4U0*.. 44657374 696E6174 312D3455 302AA9D3
00000272 I.@PaxSt op4-1Pax 49934050 61785374 6F70342D 31506178
00000288 Stop4-2P axStop4- 53746F70 342D3250 61785374 6F70342D
00000304 3Passeng erRouteC 33506173 73656E67 6572526F 75746543
00000320 lass Pas sengerDe 6C617373 20506173 73656E67 65724465
00000336 stinat2- 1U0*..I. 7374696E 6174322D 3155302A A9D34993
00000352 @PaxStop 1-1PaxSt 40506178 53746F70 312D3150 61785374
00000368 op1-2Pax Stop1-3P 6F70312D 32506178 53746F70 312D3350
```

```
00000384 assenger Destinat 61737365 6E676572 44657374 696E6174
00000400 2-2U0*.. I.@PaxSt 322D3255 302AA9D3 49934050 61785374
00000416 op2-1Pax Stop2-2P 6F70322D 31506178 53746F70 322D3250
00000432 axStop2- 3Passeng 61785374 6F70322D 33506173 73656E67
00000448 erDestin at2-3U0* 65724465 7374696E 6174322D 3355302A
00000464 ..I.@Pax Stop3-1P A9D34993 40506178 53746F70 332D3150
00000480 axStop3- 2PaxStop 61785374 6F70332D 32506178 53746F70
00000496 3-3Passe ngerDest 332D3350 61737365 6E676572 44657374
00000512 inat2-4U 0*..I.@P 696E6174 322D3455 302AA9D3 49934050
00000528 axStop4- 1PaxStop 61785374 6F70342D 31506178 53746F70
00000544 4-2PaxSt op4-3BA. 342D3250 61785374 6F70342D 33424101
00000560 ...PaxMe alPrefer 00000050 61784D65 616C5072 65666572
00000576 ence   U 0*..I.@  656E6365 20202055 302AA9D3 499340
```

Example 4: OTH (Other) input file (PaxData4_TR1_Other.ipt)

```
00000000 OTHPasse ngerSurn 4F544850 61737365 6E676572 5375726E
00000016 ame    P assenger 616D6520 20202050 61737365 6E676572
00000032 Firstnam e  PaxTi 46697273 746E616D 65202050 61785469
00000048 tle  Pas sengerRo 746C6520 20506173 73656E67 6572526F
00000064 uteClass  Passeng 75746543 6C617373 20506173 73656E67
00000080 erDestin at1-1U0* 65724465 7374696E 6174312D 3155302A
00000096 ..I.@Pax Stop1-1P A9D34993 40506178 53746F70 312D3150
00000112 axStop1- 2PaxStop 61785374 6F70312D 32506178 53746F70
00000128 1-3Passe ngerDest 312D3350 61737365 6E676572 44657374
00000144 inat1-2U 0*..I.@P 696E6174 312D3255 302AA9D3 49934050
00000160 axStop2- 1PaxStop 61785374 6F70322D 31506178 53746F70
00000176 2-2PaxSt op2-3Pas 322D3250 61785374 6F70322D 33506173
00000192 sengerDe stinat1- 73656E67 65724465 7374696E 6174312D
00000208 3U0*..I. @PaxStop 3355302A A9D34993 40506178 53746F70
00000224 3-1PaxSt op3-2Pax 332D3150 61785374 6F70332D 32506178
00000240 Stop3-3P assenger 53746F70 332D3350 61737365 6E676572
00000256 Destinat 1-4U0*.. 44657374 696E6174 312D3455 302AA9D3
00000272 I.@PaxSt op4-1Pax 49934050 61785374 6F70342D 31506178
00000288 Stop4-2P axStop4- 53746F70 342D3250 61785374 6F70342D
00000304 3Passeng erRouteC 33506173 73656E67 6572526F 75746543
00000320 lass Pas sengerDe 6C617373 20506173 73656E67 65724465
00000336 stinat2- 1U0*..I. 7374696E 6174322D 3155302A A9D34993
00000352 @PaxStop 1-1PaxSt 40506178 53746F70 312D3150 61785374
00000368 op1-2Pax Stop1-3P 6F70312D 32506178 53746F70 312D3350
00000384 assenger Destinat 61737365 6E676572 44657374 696E6174
00000400 2-2U0*.. I.@PaxSt 322D3255 302AA9D3 49934050 61785374
00000416 op2-1Pax Stop2-2P 6F70322D 31506178 53746F70 322D3250
00000432 axStop2- 3Passeng 61785374 6F70322D 33506173 73656E67
00000448 erDestin at2-3U0* 65724465 7374696E 6174322D 3355302A
00000464 ..I.@Pax Stop3-1P A9D34993 40506178 53746F70 332D3150
00000480 axStop3- 2PaxStop 61785374 6F70332D 32506178 53746F70
00000496 3-3Passe ngerDest 332D3350 61737365 6E676572 44657374
00000512 inat2-4U 0*..I.@P 696E6174 322D3455 302AA9D3 49934050
00000528 axStop4- 1PaxStop 61785374 6F70342D 31506178 53746F70
00000544 4-2PaxSt op4-3BA. 342D3250 61785374 6F70342D 33424101
00000560 ...PaxMe alPrefer 00000050 61784D65 616C5072 65666572
00000576 ence   U 0*..I.@  656E6365 20202055 302AA9D3 499340
```

Variable conversion input files

Appendix I. Variable conversion output files

The following files are the output messages that were generated from the input test files in Appendix H, "Variable conversion input files" on page 107. These were used to test the message flow created in Chapter 9, "Further message transformation" on page 65. These files contain three characters at the start of each file that define the format the output was to be created in.

The default for the output was CWF. You can see that the last file that had a format type of OTH (Other), was not coded for in the compute node and was created in the default format of CWF.

- The first column is the displacement from the beginning of the data of the first character in the second column.
- The second column is the data

Note: If you have imported Cobol copybooks, review "C header files and Cobol copybooks" on page xiii. You will find that if you have left the padding character set to SPACE for STRING elements, the blanks at the end of those input message fields will not be in the output message.

Example 1: From CWF input file (PaxData4_TR1_CWF.ipt)

```
00000000 CWFPassengerSurname    Passenger
00000032 Firstname  PaxTitle  PassengerRo
00000064 uteClass PassengerDestinat1-1U0*
00000096 ..I.@PaxStop1-1PaxStop1-2PaxStop
00000128 1-3PassengerDestinat1-2U0*..I.@P
00000160 axStop2-1PaxStop2-2PaxStop2-3Pas
00000192 sengerDestinat1-3U0*..I.@PaxStop
00000224 3-1PaxStop3-2PaxStop3-3Passenger
00000256 Destinat1-4U0*..I.@PaxStop4-1Pax
00000288 Stop4-2PaxStop4-3PassengerRouteC
00000320 lass PassengerDestinat2-1U0*..I.
00000352 @PaxStop1-1PaxStop1-2PaxStop1-3P
00000384 assengerDestinat2-2U0*..I.@PaxSt
00000416 op2-1PaxStop2-2PaxStop2-3Passeng
00000448 erDestinat2-3U0*..I.@PaxStop3-1P
00000480 axStop3-2PaxStop3-3PassengerDest
00000512 inat2-4U0*..I.@PaxStop4-1PaxStop
00000544 4-2PaxStop4-3BA....PaxMealPrefer
00000576 ence   U0*..I.@
```

Example 2: From TDS input file (PaxData4_TR1_TDS.ipt)

```
00000000 TDS|PassengerSurname    |Passeng
00000032 erFirstname  |PaxTitle  |Passeng
00000064 erRouteClass |PassengerDestinat1
00000096 -1|1234.4567|PaxStop1-1||||PaxSt
00000128 op1-2||||PaxStop1-3|||PassengerD
00000160 estinat1-2|1234.4567|PaxStop2-1|
00000192 |||PaxStop2-2||||PaxStop2-3|||Pa
00000224 ssengerDestinat1-3|1234.4567|Pax
00000256 Stop3-1||||PaxStop3-2||||PaxStop
00000288 3-3|||PassengerDestinat1-4|1234.
00000320 4567|PaxStop4-1||||PaxStop4-2|||
00000352 |PaxStop4-3||PassengerRouteClass
00000384  |PassengerDestinat2-1|1234.4567
00000416 |PaxStop1-1||||PaxStop1-2||||Pax
```

Variable conversion output files

```
00000448 Stop1-3|||PassengerDestinat2-2|1
00000480 234.4567|PaxStop2-1||||PaxStop2-
00000512 2||||PaxStop2-3|||PassengerDesti
00000544 nat2-3|1234.4567|PaxStop3-1||||P
00000576 axStop3-2||||PaxStop3-3|||Passen
00000608 gerDestinat2-4|1234.4567|PaxStop
00000640 4-1||||PaxStop4-2||||PaxStop4-3|
00000672 BA|1|PaxMealPreference   |1234.4
00000704 567
```

Example 3: From XML input file (PaxData4_TR1_XML.ipt)

```
00000000 <?xml version="1.0"?><!DOCTYPE M
00000032 RM PUBLIC "DPQ898C07E001" "www.m
00000064 rmnames.net/DPQ898C07E001"><!--M
00000096 RM Generated XML Output on :Mon
00000128 Jul 29 16:00:10 2002--><MRM xmln
00000160 s="www.mrmnames.net/DPQ898C07C00
00000192 1" xmlns:N1="www.mrmnames.net/DP
00000224 Q898C07C001-DPQ898C07E001"><PaxD
00000256 ata4ID><PaxData4><OptMsgFormatID
00000288 >XML</OptMsgFormatID><PaxSurname
00000320 >PassengerSurname    </PaxSurnam
00000352 e><PaxFirstName><Pax1stName>Pass
00000384 engerFirstname  </Pax1stName></P
00000416 axFirstName><PaxTitle><![CDATA[5
00000448 061785469746c652020]]></PaxTitle
00000480 ><PaxRoute><PaxRouteClass>Passen
00000512 gerRouteClass </PaxRouteClass><P
00000544 axDestinationMsg><PaxDestination
00000576 >PassengerDestinat1-1</PaxDestin
00000608 ation><PaxDestinationCost>1.2344
00000640 567E+3</PaxDestinationCost><PaxD
00000672 estinationStops>PaxStop1-1</PaxD
00000704 estinationStops><PaxDestinationS
00000736 tops>PaxStop1-2</PaxDestinationS
00000768 tops><PaxDestinationStops>PaxSto
00000800 p1-3</PaxDestinationStops></PaxD
00000832 estinationMsg><PaxDestinationMsg
00000864 ><PaxDestination>PassengerDestin
00000896 at1-2</PaxDestination><PaxDestin
00000928 ationCost>1.2344567E+3</PaxDesti
00000960 nationCost><PaxDestinationStops>
00000992 PaxStop2-1</PaxDestinationStops>
00001024 <PaxDestinationStops>PaxStop2-2<
00001056 /PaxDestinationStops><PaxDestina
00001088 tionStops>PaxStop2-3</PaxDestina
00001120 tionStops></PaxDestinationMsg><P
00001152 axDestinationMsg><PaxDestination
00001184 >PassengerDestinat1-3</PaxDestin
00001216 ation><PaxDestinationCost>1.2344
00001248 567E+3</PaxDestinationCost><PaxD
00001280 estinationStops>PaxStop3-1</PaxD
00001312 estinationStops><PaxDestinationS
00001344 tops>PaxStop3-2</PaxDestinationS
00001376 tops><PaxDestinationStops>PaxSto
00001408 p3-3</PaxDestinationStops></PaxD
00001440 estinationMsg><PaxDestinationMsg
00001472 ><PaxDestination>PassengerDestin
00001504 at1-4</PaxDestination><PaxDestin
00001536 ationCost>1.2344567E+3</PaxDesti
00001568 nationCost><PaxDestinationStops>
00001600 PaxStop4-1</PaxDestinationStops>
00001632 <PaxDestinationStops>PaxStop4-2<
00001664 /PaxDestinationStops><PaxDestina
00001696 tionStops>PaxStop4-3</PaxDestina
00001728 tionStops></PaxDestinationMsg></
```

```
00001760 PaxRoute><PaxRoute><PaxRouteClas
00001792 s>PassengerRouteClass </PaxRoute
00001824 Class><PaxDestinationMsg><PaxDes
00001856 tination>PassengerDestinat2-1</P
00001888 axDestination><PaxDestinationCos
00001920 t>1.2344567E+3</PaxDestinationCo
00001952 st><PaxDestinationStops>PaxStop1
00001984 -1</PaxDestinationStops><PaxDest
00002016 inationStops>PaxStop1-2</PaxDest
00002048 inationStops><PaxDestinationStop
00002080 s>PaxStop1-3</PaxDestinationStop
00002112 s></PaxDestinationMsg><PaxDestin
00002144 ationMsg><PaxDestination>Passeng
00002176 erDestinat2-2</PaxDestination><P
00002208 axDestinationCost>1.2344567E+3</
00002240 PaxDestinationCost><PaxDestinati
00002272 onStops>PaxStop2-1</PaxDestinati
00002304 onStops><PaxDestinationStops>Pax
00002336 Stop2-2</PaxDestinationStops><Pa
00002368 xDestinationStops>PaxStop2-3</Pa
00002400 xDestinationStops></PaxDestinati
00002432 onMsg><PaxDestinationMsg><PaxDes
00002464 tination>PassengerDestinat2-3</P
00002496 axDestination><PaxDestinationCos
00002528 t>1.2344567E+3</PaxDestinationCo
00002560 st><PaxDestinationStops>PaxStop3
00002592 -1</PaxDestinationStops><PaxDest
00002624 inationStops>PaxStop3-2</PaxDest
00002656 inationStops><PaxDestinationStop
00002688 s>PaxStop3-3</PaxDestinationStop
00002720 s></PaxDestinationMsg><PaxDestin
00002752 ationMsg><PaxDestination>Passeng
00002784 erDestinat2-4</PaxDestination><P
00002816 axDestinationCost>1.2344567E+3</
00002848 PaxDestinationCost><PaxDestinati
00002880 onStops>PaxStop4-1</PaxDestinati
00002912 onStops><PaxDestinationStops>Pax
00002944 Stop4-2</PaxDestinationStops><Pa
00002976 xDestinationStops>PaxStop4-3</Pa
00003008 xDestinationStops></PaxDestinati
00003040 onMsg></PaxRoute><PaxBaggageAllo
00003072 wance><![CDATA[4241]]></PaxBagga
00003104 geAllowance><PaxMealType>1</PaxM
00003136 ealType><PaxMealPreference>PaxMe
00003168 alPreference   </PaxMealPreferen
00003200 ce><PaxTotalCost>1.2344567E+3</P
00003232 axTotalCost></PaxData4></PaxData
00003264 4ID></MRM>
```

Example 4: From OTH (Other) input file (PaxData4_TR1_Other.ipt)

```
00000000 OTHPassengerSurname    Passenger
00000032 Firstname  PaxTitle  PassengerRo
00000064 uteClass PassengerDestinat1-1U0*
00000096 ..I.@PaxStop1-1PaxStop1-2PaxStop
00000128 1-3PassengerDestinat1-2U0*..I.@P
00000160 axStop2-1PaxStop2-2PaxStop2-3Pas
00000192 sengerDestinat1-3U0*..I.@PaxStop
00000224 3-1PaxStop3-2PaxStop3-3Passenger
00000256 Destinat1-4U0*..I.@PaxStop4-1Pax
00000288 Stop4-2PaxStop4-3PassengerRouteC
00000320 lass PassengerDestinat2-1U0*..I.
00000352 @PaxStop1-1PaxStop1-2PaxStop1-3P
00000384 assengerDestinat2-2U0*..I.@PaxSt
00000416 op2-1PaxStop2-2PaxStop2-3Passeng
00000448 erDestinat2-3U0*..I.@PaxStop3-1P
```

Variable conversion output files

```
00000480 axStop3-2PaxStop3-3PassengerDest
00000512 inat2-4U0*..I.@PaxStop4-1PaxStop
00000544 4-2PaxStop4-3BA....PaxMealPrefer
00000576 ence   U0*..I.@
```

Appendix J. Manipulated output files

The following files are the output messages that were generated from the input test files in Appendix H, "Variable conversion input files" on page 107. These were used to test the message flow created in Chapter 10, "Basic message manipulation" on page 75. These files contained three characters at the start of each file that defined the format the output was to be created in.

The first three characters were removed and a time stamp added to transform the message.

The default for the output was CWF. You can see that the last file that had a format type of OTH (Other), was not coded for in the compute node and was created in the default format of CWF.

- The first column is the displacement from the beginning of the data of the first character in the second column.
- The second column is the data

Note: If you have imported Cobol copybooks, review "C header files and Cobol copybooks" on page xiii. You will find that if you have left the padding character set to SPACE for STRING elements, the blanks at the end of those input message fields will not be in the output message.

Example 1: From CWF input file (PaxData4_TR1_CWF.ipt)

```
00000000 2002-07-31 10:00:02.371Passenger
00000032 Surname    PassengerFirstname  P
00000064 axTitle  PassengerRouteClass Pas
00000096 sengerDestinat1-1U0*..I.@PaxStop
00000128 1-1PaxStop1-2PaxStop1-3Passenger
00000160 Destinat1-2U0*..I.@PaxStop2-1Pax
00000192 Stop2-2PaxStop2-3PassengerDestin
00000224 at1-3U0*..I.@PaxStop3-1PaxStop3-
00000256 2PaxStop3-3PassengerDestinat1-4U
00000288 0*..I.@PaxStop4-1PaxStop4-2PaxSt
00000320 op4-3PassengerRouteClass Passeng
00000352 erDestinat2-1U0*..I.@PaxStop1-1P
00000384 axStop1-2PaxStop1-3PassengerDest
00000416 inat2-2U0*..I.@PaxStop2-1PaxStop
00000448 2-2PaxStop2-3PassengerDestinat2-
00000480 3U0*..I.@PaxStop3-1PaxStop3-2Pax
00000512 Stop3-3PassengerDestinat2-4U0*..
00000544 I.@PaxStop4-1PaxStop4-2PaxStop4-
00000576 3BA....PaxMealPreference   U0*..
00000608 I.@
```

Example 2: From TDS input file (PaxData4_TR1_TDS.ipt)

```
00000000 2002-07-31 10:02:07.641|Passenge
00000032 rSurname    |PassengerFirstname
00000064  |PaxTitle  |PassengerRouteClass
00000096  |PassengerDestinat1-1|1234.4567
00000128 |PaxStop1-1||||PaxStop1-2||||Pax
00000160 Stop1-3|||PassengerDestinat1-2|1
00000192 234.4567|PaxStop2-1||||PaxStop2-
00000224 2||||PaxStop2-3|||PassengerDesti
00000256 nat1-3|1234.4567|PaxStop3-1||||P
```

Manipulated output files

```
00000288 axStop3-2||||PaxStop3-3|||Passen
00000320 gerDestinat1-4|1234.4567|PaxStop
00000352 4-1||||PaxStop4-2||||PaxStop4-3|
00000384 |PassengerRouteClass |PassengerD
00000416 estinat2-1|1234.4567|PaxStop1-1|
00000448 |||PaxStop1-2||||PaxStop1-3|||Pa
00000480 ssengerDestinat2-2|1234.4567|Pax
00000512 Stop2-1||||PaxStop2-2||||PaxStop
00000544 2-3|||PassengerDestinat2-3|1234.
00000576 4567|PaxStop3-1||||PaxStop3-2|||
00000608 |PaxStop3-3|||PassengerDestinat2
00000640 -4|1234.4567|PaxStop4-1||||PaxSt
00000672 op4-2||||PaxStop4-3|BA|1|PaxMeal
00000704 Preference   |1234.4567
```

Example 3: From XML input file (PaxData4_TR1_XML.ipt)

```
00000000 <?xml version="1.0"?><!DOCTYPE M
00000032 RM PUBLIC "DPQ898C07G001" "www.m
00000064 rmnames.net/DPQ898C07G001"><!--M
00000096 RM Generated XML Output on :Wed
00000128 Jul 31 10:02:55 2002--><MRM xmln
00000160 s="www.mrmnames.net/DPQ898C07C00
00000192 1" xmlns:N1="www.mrmnames.net/DP
00000224 Q898C07C001-DPQ898C07E001" xmlns
00000256 :N2="www.mrmnames.net/DPQ898C07E
00000288 001-DPQ898C07G001"><PaxData5ID><
00000320 PaxData5><DateTimeStampID>2002-0
00000352 7-31 10:02:55.510</DateTimeStamp
00000384 ID><PaxSurname>PassengerSurname
00000416    </PaxSurname><PaxFirstName><P
00000448 ax1stName>PassengerFirstname  </
00000480 Pax1stName></PaxFirstName><PaxTi
00000512 tle><![CDATA[5061785469746c65202
00000544 0]]></PaxTitle><PaxRoute><PaxRou
00000576 teClass>PassengerRouteClass </Pa
00000608 xRouteClass><PaxDestinationMsg><
00000640 PaxDestination>PassengerDestinat
00000672 1-1</PaxDestination><PaxDestinat
00000704 ionCost>1.2344567E+3</PaxDestina
00000736 tionCost><PaxDestinationStops>Pa
00000768 xStop1-1</PaxDestinationStops><P
00000800 axDestinationStops>PaxStop1-2</P
00000832 axDestinationStops><PaxDestinati
00000864 onStops>PaxStop1-3</PaxDestinati
00000896 onStops></PaxDestinationMsg><Pax
00000928 DestinationMsg><PaxDestination>P
00000960 assengerDestinat1-2</PaxDestinat
00000992 ion><PaxDestinationCost>1.234456
00001024 7E+3</PaxDestinationCost><PaxDes
00001056 tinationStops>PaxStop2-1</PaxDes
00001088 tinationStops><PaxDestinationSto
00001120 ps>PaxStop2-2</PaxDestinationSto
00001152 ps><PaxDestinationStops>PaxStop2
00001184 -3</PaxDestinationStops></PaxDes
00001216 tinationMsg><PaxDestinationMsg><
00001248 PaxDestination>PassengerDestinat
00001280 1-3</PaxDestination><PaxDestinat
00001312 ionCost>1.2344567E+3</PaxDestina
00001344 tionCost><PaxDestinationStops>Pa
00001376 xStop3-1</PaxDestinationStops><P
00001408 axDestinationStops>PaxStop3-2</P
00001440 axDestinationStops><PaxDestinati
00001472 onStops>PaxStop3-3</PaxDestinati
00001504 onStops></PaxDestinationMsg><Pax
00001536 DestinationMsg><PaxDestination>P
00001568 assengerDestinat1-4</PaxDestinat
```

```
00001600 ion><PaxDestinationCost>1.234456
00001632 7E+3</PaxDestinationCost><PaxDes
00001664 tinationStops>PaxStop4-1</PaxDes
00001696 tinationStops><PaxDestinationSto
00001728 ps>PaxStop4-2</PaxDestinationSto
00001760 ps><PaxDestinationStops>PaxStop4
00001792 -3</PaxDestinationStops></PaxDes
00001824 tinationMsg></PaxRoute><PaxRoute
00001856 ><PaxRouteClass>PassengerRouteCl
00001888 ass </PaxRouteClass><PaxDestinat
00001920 ionMsg><PaxDestination>Passenger
00001952 Destinat2-1</PaxDestination><Pax
00001984 DestinationCost>1.2344567E+3</Pa
00002016 xDestinationCost><PaxDestination
00002048 Stops>PaxStop1-1</PaxDestination
00002080 Stops><PaxDestinationStops>PaxSt
00002112 op1-2</PaxDestinationStops><PaxD
00002144 estinationStops>PaxStop1-3</PaxD
00002176 estinationStops></PaxDestination
00002208 Msg><PaxDestinationMsg><PaxDesti
00002240 nation>PassengerDestinat2-2</Pax
00002272 Destination><PaxDestinationCost>
00002304 1.2344567E+3</PaxDestinationCost
00002336 ><PaxDestinationStops>PaxStop2-1
00002368 </PaxDestinationStops><PaxDestin
00002400 ationStops>PaxStop2-2</PaxDestin
00002432 ationStops><PaxDestinationStops>
00002464 PaxStop2-3</PaxDestinationStops>
00002496 </PaxDestinationMsg><PaxDestinat
00002528 ionMsg><PaxDestination>Passenger
00002560 Destinat2-3</PaxDestination><Pax
00002592 DestinationCost>1.2344567E+3</Pa
00002624 xDestinationCost><PaxDestination
00002656 Stops>PaxStop3-1</PaxDestination
00002688 Stops><PaxDestinationStops>PaxSt
00002720 op3-2</PaxDestinationStops><PaxD
00002752 estinationStops>PaxStop3-3</PaxD
00002784 estinationStops></PaxDestination
00002816 Msg><PaxDestinationMsg><PaxDesti
00002848 nation>PassengerDestinat2-4</Pax
00002880 Destination><PaxDestinationCost>
00002912 1.2344567E+3</PaxDestinationCost
00002944 ><PaxDestinationStops>PaxStop4-1
00002976 </PaxDestinationStops><PaxDestin
00003008 ationStops>PaxStop4-2</PaxDestin
00003040 ationStops><PaxDestinationStops>
00003072 PaxStop4-3</PaxDestinationStops>
00003104 </PaxDestinationMsg></PaxRoute><
00003136 PaxBaggageAllowance><![CDATA[424
00003168 1]]></PaxBaggageAllowance><PaxMe
00003200 alType>1</PaxMealType><PaxMealPr
00003232 eference>PaxMealPreference   </P
00003264 axMealPreference><PaxTotalCost>1
00003296 .2344567E+3</PaxTotalCost></PaxD
00003328 ata5></PaxData5ID></MRM>
```

Example 4: From OTH (Other) input file (PaxData4_TR1_Other.ipt)

```
00000000 2002-07-31 10:01:12.524Passenger
00000032 Surname    PassengerFirstname  P
00000064 axTitle  PassengerRouteClass Pas
00000096 sengerDestinat1-1U0*..I.@PaxStop
00000128 1-1PaxStop1-2PaxStop1-3Passenger
00000160 Destinat1-2U0*..I.@PaxStop2-1Pax
00000192 Stop2-2PaxStop2-3PassengerDestin
00000224 at1-3U0*..I.@PaxStop3-1PaxStop3-
00000256 2PaxStop3-3PassengerDestinat1-4U
```

Manipulated output files

```
00000288 0*..I.@PaxStop4-1PaxStop4-2PaxSt
00000320 op4-3PassengerRouteClass Passeng
00000352 erDestinat2-1U0*..I.@PaxStop1-1P
00000384 axStop1-2PaxStop1-3PassengerDest
00000416 inat2-2U0*..I.@PaxStop2-1PaxStop
00000448 2-2PaxStop2-3PassengerDestinat2-
00000480 3U0*..I.@PaxStop3-1PaxStop3-2Pax
00000512 Stop3-3PassengerDestinat2-4U0*..
00000544 I.@PaxStop4-1PaxStop4-2PaxStop4-
00000576 3BA....PaxMealPreference   U0*..
00000608 I.@
```

Appendix K. Delimiter examples for PaxData4 files

This appendix uses the PaxData4.h and PaxData4.cpy files to demonstrate how you could use delimiters to separate the fields in the structures and substructures of these files.

The points in the files below that are marked *1, *2, *3, and *4 are where the files either start or increase the depth of the structure through repeating elements.

- *1. Is the start of the main structure within the message. You could call this the first or top level and where you might set a particular delimiter to reflect this.

  In this primer the '|' was used to delimit this level, but it was discussed in "Delimiters" on page 45 that you could also use other delimiters such as '|1|' .
- *2. PaxRoute is a repeating element within the main message. When developing your message structures you need to consider whether to use the same delimiter between repeating elements as you used between the top level elements. Attributes of the message that you might need to consider are if your message always has the same number of repeating elements, if there is the possibility that the message has variable numbers of repeating elements, or if the elements are fixed or variable lengths. You will find that having a different delimiter at this level will aid you in looking at the data in the message.

  This primer uses '||' as the delimiter between these repeating elements.
- *3. PaxDestinationMsg is a repeating element within PaxRoute. This has again increased the depth of the message and you need to consider the points made for *2.

  This primer uses '|||' as the delimiter between these repeating elements.
- *4. PaxDestinationStops is a repeating element within PaxDestinationMsg. At this level within the structure you have a repeating element, within a repeating element, within a repeating element, within the main message. This message structure has been deliberately created to demonstrate this approach of repeating elements within elements and you will find that this type of structure is common within existing messages as well as those messages you will create.

  This primer uses '||||' as the delimiter between each of these repeating elements.

The TDS output file for these structures and the delimiters used in this primer can be seen in "Example 4: From PaxData4.ipt" on page 97. The delimiters that have been used in this primer might not be appropriate for the messages that you are working with or developing, and you need to make sure that you fully review your design.

PaxData4.h

```
/*
*******************************************************
  Passenger Data Test Trees for the C Importer
  Filename: PaxData4.h
  Description:
  Four headers describe four sizes of PaxData test.
  This is the largest header.
*******************************************************
*/
struct _PaxData4Msg
{
    struct _PaxData4
    {
        char            PaxSurname [20];                            *1
        struct          _PaxFirstName
        {
            char            Pax1stName [20];
        }PaxFirstName;
        unsigned char   PaxTitle[10];
        struct          _PaxRoute                                   *2
        {
            char            PaxRouteClass [20];
            struct          _PaxDestinationMsg                      *3
            {
                char            PaxDestination [20];
                double          PaxDestinationCost;
                char            PaxDestinationStops [3] [10];       *4
            } PaxDestinationMsg [4];
        }PaxRoute [2];
        unsigned char   PaxBaggageAllowance[2];
        long            PaxMealType;
        char            PaxMealPreference [20];
        double          PaxTotalCost;
    } PaxData4;
}PaxData4Msg;
```

PaxData4.cpy

```
      ************************************************
      *  Passenger Data Test Trees for the Cobol Importer
      *  Filename: PaxData4.cpy
      *  Description:
      *  Four headers describe four sizes of PaxData test.
      *  This is the second largest header.
      ************************************************
      *
       01  PaxData4Msg.
           03  PaxData4.
               05  PaxSurname           PIC X(20).                  *1
               05  PaxFirstName.
                   10  Pax1stName           PIC X(20).
               05  PaxTitle             PIC X(10).
               05  PaxRoute OCCURS 2 TIMES.                         *2
                   10  PaxRouteClass        PIC X(20).
                   10  PaxDestinationMsg OCCURS 4 TIMES.            *3
                       15  PaxDestination       PIC X(20).
                       15  PaxDestinationCost   COMP-2.
                       15  PaxDestinationStops OCCURS 3 TIMES PIC X(10).  *4
               05  PaxBaggageAllowance  PIC XX.
               05  PaxMealType          PIC S9(9) COMP-5.
               05  PaxMealPreference    PIC X(20).
               05  PaxTotalCost         COMP-2.
      ************************************************
      *
```

A break down of the TDS output showing delimiter separation

The following break down of the TDS output file in "Example 4: From PaxData4.ipt" on page 97 uses the *1, *2, *3, and *4 to show the increasing depth of the repeating elements in PaxData4.h and PaxData4.cpy. What you will see is that the repeating element delimiters only occur between each of the repeating elements and not before the first of the repeating elements, or after the last of the repeating elements.

The elements that are not marked are a non-repeating element within a repeating element. You could decide that these need to have a different delimiter and you would need to update the message set to reflect this.

```
PassengerSurname                *1
|PassengerFirstname             *1
|PaxTitle                       *1

|PassengerRouteClass            *2

|PassengerDestinat1-1           *3
|1234.4567
|PaxStop1-1                     *4
||||PaxStop1-2                  *4
||||PaxStop1-3                  *4

|||PassengerDestinat1-2         *3
|1234.4567
|PaxStop2-1                     *4
||||PaxStop2-2                  *4
||||PaxStop2-3                  *4

|||PassengerDestinat1-3         *3
|1234.4567
|PaxStop3-1                     *4
||||PaxStop3-2                  *4
||||PaxStop3-3                  *4

|||PassengerDestinat1-4         *3
|1234.4567
|PaxStop4-1                     *4
||||PaxStop4-2                  *4
||||PaxStop4-3                  *4

||PassengerRouteClass           *2

|PassengerDestinat2-1           *3
|1234.4567
|PaxStop1-1                     *4
||||PaxStop1-2                  *4
||||PaxStop1-3                  *4

|||PassengerDestinat2-2         *3
|1234.4567
|PaxStop2-1                     *4
||||PaxStop2-2                  *4
||||PaxStop2-3                  *4

|||PassengerDestinat2-3         *3
|1234.4567
|PaxStop3-1                     *4
||||PaxStop3-2                  *4
||||PaxStop3-3                  *4

|||PassengerDestinat2-4         *3
|1234.4567
```

Delimiter examples for PaxData4 files

```
|PaxStop4-1                    *4
||||PaxStop4-2                 *4
||||PaxStop4-3                 *4

|BA                            *1
|1                             *1
|PaxMealPreference             *1
|1234.4567                     *1
```

Appendix L. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

DB2	IBM	MQSeries
SupportPac	WebSphere	

Microsoft, Windows, Windows NT, Windows 2000 and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Bibliography

- IBM *WebSphere MQ Integrator Version 2.1 Introduction and Planning*, GC34-5599

  This provides an overview of the product, and introduces the concepts and the facilities that are available. It is available in hard and soft copy.
- The operating system specific installation guides:

  IBM *WebSphere MQ Integrator for Windows NT Version 2.1 Installation Guide*, GC34-5600

  These books describe the tasks you need to complete to install MQSeries Integrator Version 2 on the appropriate operating system, and to verify your installation. They also provide details about servicing and uninstalling the product.
- IBM *WebSphere MQ Integrator Version 2.1 Using the Control Center*, SC34-5602

  This book introduces the Control Center, and provides detailed instruction on how to work with message sets, message sets, topics, and the broker domain topology. It explains the MRM message model and how to manipulate messages. It also contains examples of how you can use all the message processing nodes.
- IBM *WebSphere MQ Integrator Version 2.1 Working with Messages*, SC34–5923

  This book provides a comprehensive reference and examples of the use of ESQL with the WebSphere MQ Integrator message processing nodes.
- IBM *WebSphere MQ Integrator Version 2.1 ESQL Reference*, SC34–6039

  This book provides a comprehensive reference and examples of the use of ESQL with the WebSphere MQ Integrator message processing nodes.
- IBM *WebSphere MQ Integrator Version 2.1 Programming Guide*, SC34-5603

  This book is for application programmers who are writing or updating applications to use the facilities provided by MQSeries Integrator Version 2.
- IBM *WebSphere MQ Integrator Version 2.1 Administration Guide*, SC34-5792

  This book is a reference book for MQSeries Integrator Version 2 system administrators. It also provides guidance information for configuring and maintaining a broker domain.
- IBM *WebSphere MQ Integrator Version 2.1 Messages*, GC34-5601

  This book documents the error and information messages generated by the product.
- *IH03 MQSeries Integrator V2 — Message display, test and performance utilities*

  This is message display and test utility is a GUI based program to assist in the development and testing of WebSphere MQ Integrator applications. It can display messages in a variety of formats, including XML and COBOL copybook representations. It can read data from files and write data to files as well as WebSphere MQ queues. (See:http://www-3.ibm.com

  /software/ts/mqseries/txppacs/txpm1.html)

Index

A

about this SupportPac xi
add
  element to compound type 80
Add a new message to the message set 79
Add an element to a message set 67
Add an XML physical layer 67
Add the message and message types to the message set workspace 67
adding
  Compute node 47
  element to message set 67
  XML physical layer 67
Adding a Compute node 47
adding a Trace node to a message flow 36
adding physical format layers 16, 42
adjusting the Input node properties 46
Assigning a message set to a broker 45
assigning the message flow to the execution group 26
assumptions
  general xii

B

Basic message manipulation
  Assign, deploy and test the message flow 84
broker
  assigning a message set 45
  defining 24
  deploying message flow 27
Buildtime and runtime environments 11
buildtime environment 11

C

C data structure
  importing 17
C data structures
  importing 15
C header files xiii, 87
  PaxData1.h 87
  PaxData2.h 87
  PaxData3.h 88
  PaxData4.h 89
check in
  error messages xiii
Cobol copybook files 91
  PaxData1.cpy 91
  PaxData2.cpy 91
  PaxData3.cpy 92
  PaxData4.cpy 93
Cobol copybooks xiii
Cobol data structure
  importing 17
Cobol data structures
  importing 15
compound type
  add element 80
  removing an element 79
compute node
  update 71, 82
Compute node
  adding 47
  introduction to properties 48
  setting properties 50
Configuration Manager 3
Control Center 3
copy
  existing message flow 70
  message set type 77
copy a message set type 77
copy an existing message flow 70
create
  element 80
  element based on new Type 77
  message set based on another message set 66
Create a CWF to TDS message flow 46
create a message set based on another message set 66
create a type and add an element 78
create an element based on the new Type 77
create and add an element to a compound type 80
creating
  logical message 19
  message flow 22
  message set 15, 42
  MQ resources 22
  WebSphere MQ resources 22
creating a basic message flow 22
creating a logical message 19
creating a message set 15
Creating and assigning the message set 41
creating and testing a basic message flow 21
creating message models 12
Creating the WebSphere MQ resources 22
CWF import report file
  example 105

D

data interpretation 5
defining a broker in the domain 24
delimiter examples for PaxData4 files 119
Delimiters 45
deploying the message flow to the broker 27
deployment
  error messages xiii

E

element
  add to message set 67
  create 80
  removing from compound type 79
environment
  buildtime 11
  runtime 11
error messages
  check in xiii
  deployment xiii
error messages during check in and deployment xiii
error processing in a message flow 53
Example C header files 87
example Cobol copybook files 91
Example CWF import report file 105
example input files
  PaxData1.ipt 95
  PaxData2.ipt 95
  PaxData3.ipt 95
  PaxData4.ipt 95
example input message files 95
example TDS output 97
execution group
  assigning a message flow 26
exercise testing xiv

F

flexibility xiv
Further message transformation 65

H

how the broker processes a message 12

I

importing a C data structure 17
importing a Cobol data structure 17
importing C data structures 15
importing Cobol data structures 15
input message files 95
input node
  update 70
Input node
  adjusting properties 46
introduction to messaging 1
Introduction to the Compute node Properties dialog 48
Introduction to the MRM 9

L

log trace file 101
logical format 6
logical message
  creating 19

logical message model 10
element 10
message 10
message set 10
type 10

M

Manipulated output files 115
from CWF input file 115
from OTH input file 117
from TDS input file 115
from XML input file 116
mapping the TDS physical layer 43
message
add to message set 79
add to workspace 67
logical format 6
physical format 6
message flow
adding a Trace node 36
assigning 26
copying 70
creating 22
deploying 27
Message flow
error processing 53
message model
logical 10
message models
creating 12
message processing concepts 5
message repository 12
message set
add message 79
assigning 45
creating 15, 42
message set type
copy 77
message types
add to workspace 67
MQ resources
creating 22
MRM
introduction 9

O

output node
update 71

P

physical format 6
physical format layers
adding 16, 42
physical layer
TDS mapping 43

R

removing an element from a compound type 79
required knowledge xi
runtime environment 11

S

Setting the Compute node properties for CWF to TDS conversion 50
Setting the Compute node to convert CWF to TDS 48
SupportPac contents xiv

T

TDS output
PaxData1.ipt 97
PaxData2.ipt 97
PaxData3.ipt 97
PaxData4.ipt 97
TDS Output 97
TDS physical layer
mapping 43
testing
Using the IH03 SupportPac 31
Using WebSphere MQ Explorer 29
Testing the CWF to TDS message flow 52
testing the exercises xiv
trace
demonstrating 37
Trace node 35
adding to a message flow 36

U

update
compute node 71, 82
input node 70
output node 71
update the compute node 71, 82
update the input node 70
update the output node 71
Use of the Trace node 35

V

variable conversion input files 107
CWF input file 107
OTH input file 109
TDS input file 108
XML input file 108
variable conversion output files 111
from CWF input file 111
from OTH input file 113
from TDS input file 111
from XML input file 112

W

WebSphere MQ 1
WebSphere MQ Integrator 2
Configuration Manager and Control Center 3
high level overview 3
message broker 4
summary 4
WebSphere MQ resources
creating 22
who this document is for xi

X

XML physical layer
adding 67
XML trace file 99

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM®.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–842327
  - From within the U.K., use 01962–842327
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

IC00-IA7A-00