WebSphere MQ Integrator

SupportPac IC03 – Examples of message flows

Version 2.1

WebSphere MQ Integrator

SupportPac IC03 –
Examples of message flows

Version 2.1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 91.

Second edition (August 2002)

This edition applies to Version 2.0 of SupportPac™ IC03 Examples of WebSphere® MQ Integrator message flows and to all subsequent releases and modifications until otherwise indicated in new editions.

Version 1.0 of SupportPac IC03 was entitled Business Scenarios for IBM® MQSeries® Integrator.

Contents

Figures

Tables

About this SupportPac

This SupportPac, IC03, provides a set of files that define the resources that are required by six business scenarios that you can use in your WebSphere MQ Integrator broker domain.

This document provides detailed instructions about how to use these files to:
- Create message sets
- Create and configure message flows
- Deploy the message flows to your brokers

The six examples are:
- A retail operation
- A loan request application
- A dynamic routing example
- A travel agent (single message flow) example
- A travel agent (double message flow) example
- A price quotations, or estimations, example

The description of each example highlights the facilities that are used in that example. This allows you to assess whether the contents of that example are of interest to you, and whether it can be used in your broker domain.

Each example is self-contained. You need to install only the files that are provided for the examples that you want to use.

The documentation of each example provides instructions for using the definition files that are provided in the SupportPac. It also contains information to help you use the Control Center to create your own resources, should you prefer to do this rather than use the supplied definition files.

Who this document is for

This document is for users of WebSphere MQ Integrator Version 2.1. The previous release of this SupportPac should be used for the predecessor product, MQSeries Integrator Version 2.

The documentation for this SupportPac (IC03-01) and its predecessor (IC03-00) can be obtained from the Web site at:

`http://www.ibm.com/software/mqseries/`

What you need to know to understand this document

You must be familiar with the concepts of WebSphere MQ Integrator, and know how to use the Control Center to design message sets and message flows.

"Bibliography" on page 95 describes the contents of the WebSphere MQ Integrator library. Refer to the books in this library for more information about WebSphere MQ Integrator.

Throughout this document, the term *UNIX* is used to refer to AIX, HP-UX, and Sun Solaris operating systems, where their behavior is the same.

The term *local error log* is used within this document to mean the Event Log on Windows NT, or the `syslog` on UNIX systems.

The term *home directory* is used to refer to the directory into which WebSphere MQ Integrator is installed. The names of the default home directories are shown in the following table:

Operating System	**Default Home Directory**
AIX	/usr/opt/mqsi
HP-UX	/opt/mqsi
Solaris	/opt/mqsi
Windows NT	C:\Program Files\IBM\MQSeries Integrator 2.1\

Assumptions

If you download and use the files in this SupportPac, it is assumed that:

- You have read, and agree to, the conditions documented in the file `licence2.txt` that is included in the SupportPac.
- You have installed WebSphere MQ Integrator, Version 2.1 or later. The examples might not work with earlier versions or releases of WebSphere MQ Integrator.

  Your installation options must include the broker, the Configuration Manager, and the Control Center. The installation of other components of WebSphere MQ Integrator is optional.
- You have created and started a broker on a supported, runtime, operating system (AIX, HP-UX, Solaris, or Windows NT) that specifies queue manager MQSI_SAMPLE_QM.

  The *WebSphere MQ Integrator Installation Guide* for your broker operating system provides instructions that help you to do this.

  You can use a different queue manager if you want. In this case, you must modify the provided sample resources after you have imported them. Refer to the instructions for each example for more details about what you need to do.
- 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 your broker operating system provides instructions that help you to do this.
- You have access to DB2 to create and initialize the database and tables that are required by these examples.

  For more information about initializing a DB2 database, see “Creating the database” on page xiv.
- You have updated your broker domain topology to define the brokers to which you intend to deploy the message flows for the examples that you want to use.

  This document assumes that you are assigning message flows to the broker’s default execution group; if you want to use a different execution group, you can do so during the assignment step documented for each example.

  For more information about updating your topology, see the *WebSphere MQ Integrator Using the Control Center* book.

The *WebSphere MQ Integrator Administration Guide* contains detailed information about using the WebSphere MQ Integrator commands; for example, commands that create, start, or stop brokers.

SupportPac Contents

This SupportPac is supplied in the zip file, *ic03.zip*, that contains all the files that you need to use the examples described in this document:

- ic0301.pdf (this document)
- License agreement `licence2.txt`
- Version description `level.txt`
- Files for example 1 (retail operation):
  - `retail\Counter-MRM.xml` - the subflow used to write items from the MRM receipt message to a database (see Chapter 1, "The retail operation" on page 1 for a definition of MRM)
  - `retail\CounterXML.xml` - the subflow used to write items from the self-defining XML receipt message to a database
  - `retail\mrmreceipt.xml` - a sample MRM receipt message
  - `retail\MRMscenario.xml` - the main message flow using the MRM message
  - `retail\ReceiptMessagesMsgSet.mrp` - the message set defining the receipt messages
  - `retail\RegisterSubscription-MRM.xml` - the message flow used to register a subscription and set a content-based filter on an MRM message
  - `retail\RegsubXML.xml` - the message flow used to register a subscription and set a content-based filter on a self-defining XML message
  - `retail\retail.tst` - the input file to `runqmsc` used to define the MQSeries queues used by the retail message flows
  - `retail\retail.sql` - the input file to DB2 used to define tables within a database
  - `retail\subscribemsg.xml` - a sample message used to register a subscription
  - `retail\subscribemsgMRM.xml` - a sample self-defining XML receipt message
  - `retail\xmlscenario.xml` - the main message flow using the self-defining XML message
- Files for example 2 (loan request application):
  - `loan\credit.cpy` - the COBOL copybook defining the message
  - `loan\LoanRequestMsgSet.mrp` - the message set used to define the message
  - `loan\loanreq.xml` - a sample XML loan request message
  - `loan\Loan Request.xml` - the message flow used for the loan request example
  - `loan\loan.tst` - the input file to `runmqsc` used to define the MQSeries queues used by the loan request message flow
  - `loan\loan.sql` - the input file to DB2 used to define and populate tables in a database
- Files for example 3 (dynamic routing example):
  - `route\RouteToLabel.mrp` - the message set defining the input message for the Route flow
  - `route\JOURNEYAggregation.xml` - the message flow for travel agent example 1
  - `route\trademsg.xml` - a sample XML input message
  - `route\route.sql` - the input file to DB2 used to define and populate tables in a database
  - `route\route.tst` - the input file to `runmqsc` used to define the MQSeries queues used by the message flow
- Files for example 4 (travel agent single-flow example):

  - `travel1\JOURNEYAggregation.xml` - the aggregate message flow
  - `travel1\travelreq.xml` - a sample XML input message
  - `travel1\journey.tst` - the input file to `runmqsc` used to define the MQSeries queues used by the message flow
- Files for example 5 (travel agent double-flow example):
  - `travel2\JOURNEYIn.xml` - the fan-in message flow
  - `travel2\JOURNEYOut.xml` - the fan-out message flow
  - `travel2\JOURNEYReply.xml` - the reply message flow
  - `travel2\travelreq.xml` - a sample XML input message
  - `travel2\journeyagg.tst` - the input file to `runmqsc` used to define the MQSeries queues used by the message flow
- Files for example 6 (price quotation example):
  - `estimates\GetEstimatesOut.xml` - the fan-out message flow
  - `estimates\GetEstimatesIn.xml` - the fan-in message flow
  - `estimates\GetEstimatesReply.xml` - the reply message flow
  - `estimates\GetEstimatesAgg.tst` - the input file to `runmqsc` used to define the MQSeries queues used by the message flow

Using the examples

There are three ways that you can use the supplied examples:

- You can use each complete example as supplied.
- You can use the complete example, but change aspects of its operation to match your specific requirements.
- You can copy the message flow and delete one or more nodes, leaving only those nodes that you want to use within your own message flow. You can then modify the properties of these nodes to match the rest of your message flow.

Creating the database

The message flows that are defined in the examples use a database for the storage and retrieval of the information that passes through the message flows. Before you can use these message flows, you must create a database called MYDB, and set up an ODBC connection for it.

This SupportPac includes files and information that help you to create and initialize MYDB as a DB2 database. You can use a different database supplier if it is supported by WebSphere MQ Integrator; however, the SupportPac does not provide any help for these databases. For further information, refer to the *WebSphere MQ Integrator Installation Guide* for your broker operating system.

You can use a different database name, and different table names, if you want. If you do this, you must change the example resources after you have imported them as indicated in the appropriate descriptions.

Before you can configure and use the message flows, you must create:

1. A database called MYDB.

   If you use DB2 for this database, you can:

   - Start the DB2 Control Center and expand the Object tree in the DB2 Control Center until you find Databases.

     Right-click **Databases** and select **Create—>Database Using Wizard**.

     You must enter a database name and alias, but you can let all the remaining options take their default values.

   - Start a DB2 Command Window and type the following commands:

     ```
     db2 start database manager
     db2 create database MYDB
     db2 connect to MYDB
     db2 bind <INSTHOME>\bnd\@db2cli.list grant public
     db2 connect reset
     ```

     where `<INSTHOME>` is the DB2 instance directory for the userid. For example, it might be `c:\sqllib` on Windows NT, or `/home/db2instl/sqllib` on UNIX.

2. An ODBC connection to the MYDB database.

   If you use DB2 for this database, you can:

   - On Windows NT, select **Start—>Settings—>Control Panel**.

     Double-click the **ODBC** icon to start the *ODBC Data Source Administrator*.

     Select the **System DSN** tab, and click **Add** to add a new IBM DB2 data source for this database.

     For more detailed instructions, see the *WebSphere MQ Integrator for Windows NT Installation Guide*.

- On UNIX, edit the file `/var/odbc/.odbc.ini`.

  The *WebSphere MQ Integrator Installation Guide* for your operating system provides details to help you to complete this task.

Refer to the *WebSphere MQ Integrator Installation Guide* for the operating system on which your broker is running for guidance about how to set up an ODBC connection if your database is not DB2.

Ensure that each broker to which you deploy the message flows has access to this database. The userid and password that the broker is using to access databases on your behalf, specified by the `–u` and `–p` flags respectively on the **mqsicreatebroker** command, must be authorized for read and update access to MYDB.

Using the supplied DB2 scripts

The SupportPac includes DB2 scripts that you can use to create and initialize the tables that are required by each example. The description of each example identifies the supplied script, and tells you how to use it.

The userid from which you use these scripts determines the database schema name. For example, if you are logged on as userid `USER1`, the tables you create have the schema `USER1.TABLENAME`. If the broker is not running under `USER1` (that is, the `–u` flag on the **mqsicreatebroker** command did not specify `USER1`), it attempts to access the schema name that includes its runtime userid and access fails.

The invocation of the DB2 scripts shown in this document assumes that you are using your login userid and password to access the database, and that you specified these values on the **mqsicreatebroker** command. If the broker is running under another userid, you must specify this when you run the DB2 script to ensure that the database schema is created with the correct userid to allow access by the broker.

For example, if you specified `–u BKUSER1` and `–p BKPW1` on the **mqsicreatebroker** command, you must invoke the DB2 scripts specifying these values. For example:

```
db2 connect to MYDB user BKUSER1 using BKPW1
db2 -f loan.sql
```

Invoking the DB2 scripts in this way ensures that the tables are created with the correct schema name to allow access by the broker.

Ensure that, when you select a database table in a node in an example message flow (for example, in “Configuring the Warehouse node for the MRM message” on page 17), you reference the correct schema name. This might require you to modify the message flows that are supplied.

Testing the scenarios

The scenarios are provided as working examples, with import files for messages and message flows. If you import these files as shown, and complete their definition, assignment, and deployment, you can pass messages of the defined format through the flows on your brokers.

Each example includes some information on how you can test its operation.

In addition, you might find the following helpful to you in testing:

- The IH02 SupportPac WebSphere MQ Integrator PUT Utility

Using the examples

You can obtain this SupportPac from the Web site:
http://www.ibm.com/software/mqseries

This SupportPac provides a utility program, MQSIPUT, that allows messages to be constructed in various permitted formats, including various headers, within a file. This allows you to test with messages that can be valid or not valid, without having to write programs to set up the headers or the message content.

- The MQSeries Explorer

  This program allows you to view the contents of messages on a queue. MQSeries 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, that you are interested in.

  Note that you cannot use MQSeries Explorer to put messages to a queue for testing these scenarios, because it cannot use a file as input.

Chapter 1. The retail operation

This chapter describes the business scenario for the retail operation that is discussed in Chapter 3 of the book, *WebSphere MQ Integrator Introduction and Planning*, Version 2.1.

In WebSphere MQ Integrator Version 2.0.1, implementation details for this scenario were provided in Appendix A of the *WebSphere MQ Integrator Using the Control Center* book.

This scenario illustrates the following concepts:

- Publish/Subscribe
- Warehousing and writing to databases
- Message subflows
- Controlling the flow of a message using FlowOrder and Filter nodes
- Manipulating the content of a message or message header using Compute and Extract nodes
- Data conversion
- Tracing message flow activity

The message flows contain at least one example of each of the following IBM Primitive nodes:

- MQInput
- Check
- Warehouse
- Floworder
- Extract
- Filter
- Trace
- Compute
- Publication
- MQOutput
- Database

The SupportPac includes files that support two versions of this scenario:

- The message flow that uses a message set and messages defined using the Message Repository Manager (MRM).

  The MRM is a component of the Configuration Manager that manages message definitions and maintains the message repository in which they are stored. You use the Control Center to define messages to the MRM.
- The message flow that uses a self-defining XML message.

The two versions illustrate the use of different nodes to handle the different messages. The ESQL within the message flow includes some field references that are specific to the message format used.

For more information about using these field references in ESQL, refer to the *WebSphere MQ Integrator ESQL Reference* book.

Introduction

The scenario uses three separate message flows:

- The main flow.

  The main flow, which is described in “The main flow” on page 15, has one input node and four output nodes. It is configured to provide different backend systems with different data, according to their requirements:

  - The **Audit** branch of the flow (described in “Audit flow” on page 16) is used to check that the incoming message belongs to the expected message set (that is, the message is a valid receipt), and can therefore be processed by the rest of the flow.

    If the message is valid, the receipt information is stored in a database for later retrieval.

    A FlowOrder node is used to control the order of processing of the Finance and Stock branches of the flow that follow the Audit flow.
  - The **Finance** branch of the flow (described in “Finance flow” on page 20) extracts some information from the message to suit the requirements of the Finance department.

    A trace record is written to a file to record what information has been extracted. A message containing the extracted information is passed on to the Finance Department. If the message is an MRM message, it is converted into an IMS format before being passed on to the Finance Department.
  - The **Stock** branch of the flow (described in “Stock flow” on page 25) is used to add up all instances of an item sold.

    This information is used by the Distribution group to maintain stock levels.
  - The **Partner** branch of the flow (described in “Partner Flow” on page 28) is used to filter messages that contain more than one purchase of the same item. These messages, known as *multibuys*, are stored in a database and are published to subscribers .
- The update multibuy database flow.

  This flow, which is described in “Updating the database” on page 29, is an example of a message flow that has been created as a subflow that can be embedded within a higher-level message flow.

  This illustrates the use of a repeated sequence of actions that can be included at several points within a larger message flow to provide common function.
- The register subscriptions flow.

  This flow, which is described in “Register subscriptions flow” on page 35, is used to subscribe to publications based on the content of the message.

When you have defined your message and message flows, you must assign them to the broker on which you want the message flow to run, and you must deploy the changes you have made to the broker domain to activate those changes. These steps are described in “Assigning the message set and message flows” on page 38 and “Deploying the configuration” on page 38.

Before you start, you must complete the tasks described in “Preparing to use the example” on page 3.

Preparing to use the example

You must complete the tasks described in the following sections before you can successfully deploy the retail message flows. Complete these steps before you start to define your message set and message flows.

Initializing the database

The message flow in this scenario uses a database for the storage of information passing through the message flow. Before you can configure and use the message flow, you must create a database, `MYDB`, and an ODBC connection to that database. See “Creating the database” on page xiv for further information.

You must also set up the tables `RECEIPTINFO`, `RECEIPTINFO2`, and `MULTIBUY` that are used by the message flow. You can use the file `retail.sql` to create these tables, or you can create the tables yourself. If you use the supplied script, you must review the guidelines given in “Using the supplied DB2 scripts” on page xv for accessing the database and creating and using schema names.

To use the file `retail.sql`, open a DB2 Command Window on the system on which your broker is running, and type:

```
db2 connect to MYDB
db2 -f retail.sql
```

Note that you must type the complete path for the file `retail.sql` if the file is not in your current directory.

You might see the error message DB21034E the first time you run this command file. This is because the file first drops any existing tables, and then creates them. If the tables do not already exist, error messages are generated. You can ignore these messages.

Ensure that each broker to which you deploy this message flow has access to this database.

Setting up MQSeries resources

This scenario gets an input message for the message flow from an MQSeries queue on the queue manager that hosts the broker to which the message flow has been deployed. It puts messages to a number of output queues, depending on the action taken on each message received by the message flow. It also uses a number of queues to handle publications and subscriptions.

The queues required are:
- IN
- FINANCE
- STOCK
- NOTMULTI
- PARTNERS
- FAILED
- SUBS
- SUBIN
- SUBOUT

You can use the file `retail.tst` to create these queues, or you can create the queues yourself.

The retail operation

To use the file `retail.tst`, type the following on a command line on the system on which your broker is running:

```
runmqsc MQSI_SAMPLE_QM <retail.tst
```

You must type the complete path for the file `retail.tst` if the file is not in your current directory. Replace the queue manager name in the command above if the name shown is not correct for your broker.

You can use different names for these queues if you choose. If you do, you must edit `retail.tst` before you use the `runmqsc` command, and you must modify the queue properties of the nodes in the message flow to match your new names. See "Creating and configuring the message flows" on page 13 for more details.

If your brokers and your Configuration Manager are hosted by different queue managers, you must also set up pairs of channels between each broker queue manager and the Configuration Manager queue manager. You also need to start an MQSeries listener for each queue manager. For details of how to do this, refer to the *WebSphere MQ Integrator Installation Guide* for your broker operating system.

Creating and completing the message set

You can work through this example either with a message that is defined in the WebSphere MQ Integrator message repository, and that belongs to the message domain MRM, or with a self-defining XML message that belongs to the message domain XML. The nodes used in the example differ slightly depending on which type of message you use, and the field references in the ESQL in the nodes are different.

The receipt message as an MRM message

If you want to use a predefined message structure, you must use the Control Center to define the logical structure and the physical structure, also known as the *wire format*, of the message to the message repository . The SupportPac includes an import file, `ReceiptMessagesMsgSet.mrp`, that creates the message set and message for you.

To import the message set definition from this file, you must:

1. Open a command prompt window.
2. Stop the Configuration Manager using the command **mqsistop configmgr**.
3. Change to the directory containing the file `ReceiptMessagesMsgSet.mrp`.
4. Import the message set by typing:

   ```
   mqsiimpexpmsgset -i -u mqsiuid -p mqsipw -n MQSIMRDB -f ReceiptMessagesMsgSet.mrp
   ```

   where `mqsiuid` and `mqsipw` are the userid and password that are used for message repository access, and `MQSIMRDB` is the name of the message repository. You must use the same values that you specified for the `–u`, `–p`, and `–n` flags on the **mqsicreateconfigmgr** command.The installation guide for your broker operating system gives more details about the use of the **mqsicreateconfigmgr** command.
5. Restart the Configuration Manager using the command **mqsistart configmgr**. This picks up the new definition
6. Restart the Control Center from the Windows NT **Start** menu.
7. In the Control Center Message Sets view, right-click on the Message Sets root and select **Add to Workspace**. Select the message set `My Receipt Messages`, and click **Finish**.

You can now view the message set in the left-hand pane of the Message Sets view. The components of the message set are all present in the message repository, but are not added to your workspace. If you want to view some or all of these components, you can add them by right-clicking on the component folder (for example, for Messages) and selecting **Add to Workspace**.

The `My Receipt Messages` message set contains the following messages:

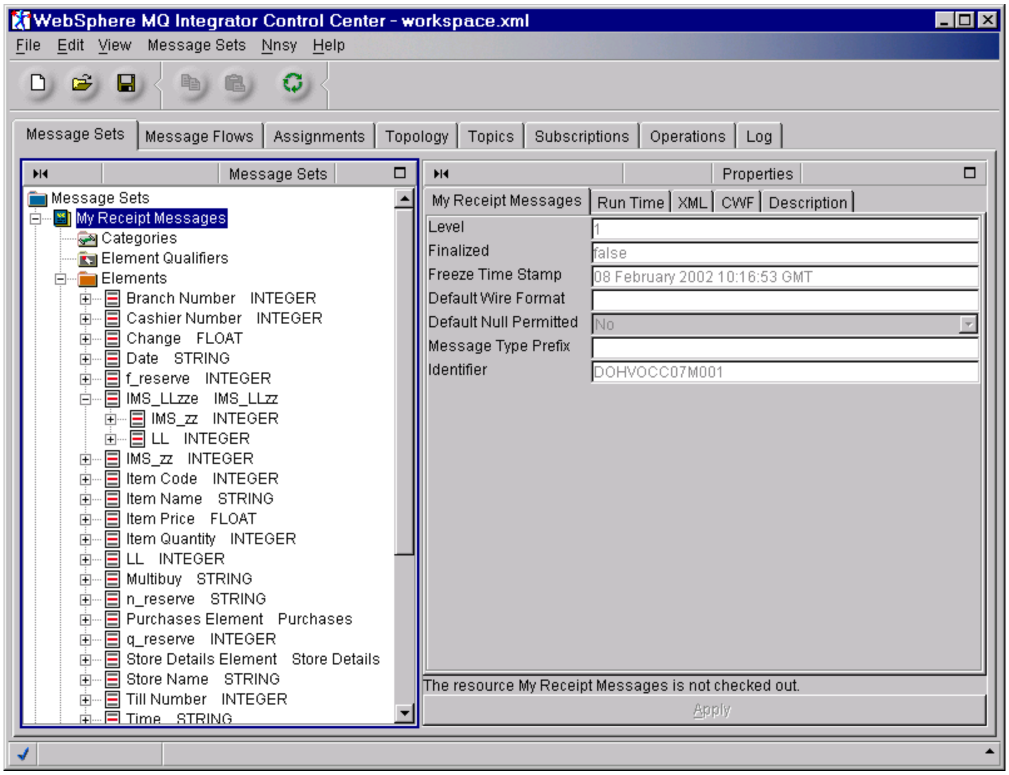
- `Receipt Message`. This contains information from a shop receipt that is required by other departments.
- `Stock Message`. This contains information that is derived from the receipt message and is used to control stock levels.
- `IMS Message`. This defines an IMS variable-length string-format message to allow the receipt message to pass from WebSphere MQ Integrator to IMS on a system running z/OS.

If you prefer to create the messages yourself, follow the instructions provided in “Creating the message set by hand” on page 7.

Associating the MRM receipt message with a message repository definition

When a message received by an MQInput node has a corresponding definition in the message repository, WebSphere MQ Integrator needs to know which parser you are expecting to use for the message (called the message domain), which set the message belongs to (called the message set) and the identifier of the message definition (called the message type).

This is defined by an association, and can be specified in one of two ways. If you use the sample message, `mrmreceipt.xml`, that is supplied by the SupportPac to test this message flow, the association is already made for you, using the first method shown.



Message Sets
- My Receipt Messages
  - Categories
  - Element Qualifiers
  - Elements
    - Branch Number INTEGER
    - Cashier Number INTEGER
    - Change FLOAT
    - Date STRING
    - f_reserve INTEGER
    - IMS_LLzze IMS_LLzz
      - IMS_zz INTEGER
      - LL INTEGER
    - IMS_zz INTEGER
    - Item Code INTEGER
    - Item Name STRING
    - Item Price FLOAT
    - Item Quantity INTEGER
    - LL INTEGER
    - Multibuy STRING
    - n_reserve STRING
    - Purchases Element Purchases
    - q_reserve INTEGER
    - Store Details Element Store Details
    - Store Name STRING
    - Till Number INTEGER
    - Time STRING

Level	1
Finalized	false
Freeze Time Stamp	08 February 2002 10:16:53 GMT
Default Wire Format	
Default Null Permitted	No
Message Type Prefix	
Identifier	DOHVOCC07M001

The resource My Receipt Messages is not checked out.

Figure 1. The message set properties, showing the identifier

If you want to create your own messages, you must make the association in one of the following ways:

1. Define the message domain, the message set, and the message type on the NAMEVALUEDATA part of an MQRFH2 header.

   This method has been used in the sample message file `mrmreceipt.xml`.

   The mcd folder within the message provides basic definition information, the psc folder contains information specific to publications.

   - mcd folder

     Msd The parser to be used for this message. This is MRM in this example. Other values are BLOB, XML, NEON, and NEONMSG. It must be entered in uppercase.

     Set The identifier of the message set to which the message belongs. This is the identifier assigned by WebSphere MQ Integrator when you create the message set in the Control Center. In this example, it is `DOHVOCC07M001` (see Figure 1 on page 6).

Type The identifier of the message definition to which this message maps. It is the identifier you assign when you define the message in the Control Center. In this example, it is receiptmsg. You cannot copy and paste the identifier from the message properties in the Control Center, so make sure that you type it exactly as shown there.

Fmt This is the custom wire format of the message. In this example, it is XML. Other possible values are CWF and PDF.

- psc folder

  Command Specifies that the message is to be published.

  Topic Specifies the topic for publication. In this example, the topic is ″Multibuy″.

2. Define the message domain, the message set, and the message type on the **Default** tab of the MQInput node, as shown in Figure 4 on page 15.

   This method is not used if you import the message flow and use the supplied message which contains an MQRFH2 header.

The receipt message as an XML message

A self-defining XML message can be passed through a message flow without having to be defined as part of a message set in the message repository.

However, if you decide to use an XML message, you cannot use some of the IBM primitive nodes. For example, Extract nodes and Warehouse nodes can only be used with predefined messages. Use Compute nodes or Database nodes instead of these.

An example of an XML message of the format expected by the XML message flow scenario is provided in the file `subscribemsgMRM.xml`. You can use this to test your scenario when you have deployed the message flow. See “Testing the message flow” on page 39 for more details about testing.

Creating the message set by hand

You can use the information here if you want to create the MRM message set used by this scenario yourself. It provides a systematic example that shows you how to create a message for the receipt data. It shows you how to create a message using the bottom-up approach, but you can also use a top-down approach if you prefer.

The message set you create contains three messages called Receipt Message, Stock Message, and IMS Message.

- The purpose of the Receipt Message is to take information from a shop receipt and, through the message flow, feed the information to the people who need it. For example, a financial controller needs to know the sales figures from each branch.
- The Stock Message is used to illustrate how information from one message can be modified and mapped into another message. For example, the stock controller needs to know the total quantity of a particular item for each receipt. The Compute node in the Stock message flow adds the number of items and puts the total into the Stock Message.

- The IMS message contains receipt information in a format acceptable to an IMS system.

The message uses structured compound elements that you populate with simple elements. Each element defines a unit of information.

Refer to the relevant chapter of the *WebSphere MQ Integrator Using the Control Center* book for details on how to create message sets and messages. The information provided here focuses on how you set up the properties to make this example work.

1. Create a message set.

   Give this message set any name. In the import files in the SupportPac, it is called My Receipt Messages. Check that the parser on the Run Time tab is set to MRM. When you click **Finish**, WebSphere MQ Integrator assigns the message set a unique identifier and writes this into the Identifier field of the message set properties. This is the identifier that you must name, either on the MQInput node or in the MQRFH2 header.
2. Create simple elements.

   These are the lowest-level units of information. You can give them any name and identifier you want. Table 1 summarizes the simple elements, the type selected for each one, and the name and identifier used in the import files in the SupportPac.

Table 1. Retail simple elements, types, names, and identifiers

Simple element name	Identifier	Type
Store Name	storename	STRING
Branch Number	branchnum	INTEGER
Cashier Number	cashiernum	INTEGER
Till Number	tillnum	INTEGER
Date	date	STRING
Time	time	STRING
Item Name	itemname	STRING
Item Code	itemcode	INTEGER
Item Price	itemprice	FLOAT
Item Quantity	itemquantity	INTEGER
Total Items	totalitems	INTEGER
Multibuy	multibuy	STRING
Total Sales	totalsales	FLOAT
Change	change	FLOAT
Total Item Quantity	totalitemquantity	INTEGER
f_reserve	f_reserve	INTEGER
IMS_zz	IMS_zz	INTEGER
LL	LL	INTEGER
n_reserve	n_reserve	STRING
q_reserve	q_reserve	INTEGER

The XML descriptor tag for the element in the message must match the identifier used in the definition. For example, the element Store Name has an identifier storename and is represented in the message as <storename>.

Note that, for the elements Date and Time, after you click **Finish** on the element property pages, but before you move on to the next element, you should go to the COBOL tab and change the default settings of the COBOL Language Name property from DATE and TIME to something else. Using a COBOL keyword in these fields is not permitted.

The following explanation of the last few items in the table might help your understanding:

- f_reserve is used to hold the value of the counter when, in the subflow *Counter – MRM1* or *Counter – XML1*, each item in the receipt message is written to a database.
- IMS_zz is a 2-byte reserved field in the IMS message definition.
- LL is a 2-byte field in the IMS message definition that specifies the total length of the IMS variable string item (length of LL (2 bytes) + length of zz (2 bytes) + length of character string itself).
- n_reserve is used to hold the value of the item name when, in the subflow, each item in the receipt message is written to a database.
- q_reserve is used to hold the value of the item quantity when, in the subflow, each item in the receipt message is written to a database.

3. Create element lengths for the elements of type STRING.

   You can give them any name and identifier you want. Table 2 summarizes the STRING elements, the length defined for each one, and the name and identifier used in the import files in the SupportPac.

Table 2. Retail STRING elements, lengths, names, and identifiers

Element name	Element length name	Maximum Length	Element length identifier
Store Name	Store Name Length	20	storenamelen
Date	Date Length	10	datelen
Time	Time Length	10	timelen
Item Name	Item Name Length	40	itemnamelen
Multibuy	Multibuy Length	5	multibuylen
n_reserve	n_reservelen	20	n_reservelen

4. Add the lengths to the corresponding string elements; for example, add Store Name Length to the element Store Name.
5. Create element valid values for some of the elements.

   You can give them any name and identifier you want. Type must be the same type of the element that the valid value is associated with. Table 3 summarizes the INTEGER elements, the minimum and maximum valid value defined for each one, and the name and identifier used in the import files in the SupportPac.

Table 3. Retail INTEGER elements, values, names, and identifiers

Element name	Element valid value name	Element valid value identifier	Type	Valid value range
Branch Number	Branch Number Value	branchnumval	INTEGER	00000000 – 99999999

Table 3. Retail INTEGER elements, values, names, and identifiers (continued)

Element name	Element valid value name	Element valid value identifier	Type	Valid value range
Cashier Number	Cashier Number Value	cashiernumval	INTEGER	000 – 999
Till Number	Till Number Value	tillnumval	INTEGER	000 – 999

6. Add the valid values to the corresponding elements; for example, add Branch Number Value to the element Branch Number.
7. Create compound types.

   These are used as the type for compound elements (higher-level elements) within the message. Transactionlog is used as the type for the message itself, thereby bringing all the lower-level structures together. You can give them any name and identifier you want. Table 4 summarizes the compound types, names, and identifiers used in the import files in the SupportPac.

Table 4. Retail compound type names and identifiers

Compound type name	Identifier
Store Details	storedetails
Purchases	purchases
Totals	totals
Transaction Log	transactionlog
Output Transaction Log	outputtransactionlog
IMS_LLzz	IMS_LLzz
IMS Message Type	imsmsgtype

8. Add elements to the compound types. (Leave *transactionlog outputtransactionlog*, and *imsmsgtype* for now.)

   The order of elements in the message being passed through the message flow **must** match the order of elements in the message definition. This order is defined by the order of elements in the compound types.

   When you add elements to a compound type, they are added in reverse order. For example, selecting Store Name followed by Branch Number produces the order Branch Number followed by Store Name. There is a Reorder option on the Types pull-down to re-sequence the elements within a type.

   To match the message shown in Figure 1 on page 6, add the elements in the sequence shown in Table 5. Use Ctrl+left-click to select multiple elements.

Table 5. Retail elements to add to compound types

Compound type	Elements to be added
Store Details	• Time • Date • Till Number • Cashier Number • Branch Number • Store Name

Table 5. Retail elements to add to compound types (continued)

Compound type	Elements to be added
Purchases	• q_reserve • Item Quantity • Item Price • Item Code • n_reserve • Item Name
Totals	• f_reserve • Change • Total Sales • Multibuy • Total Items • Total Item Quantity
IMS_LLzz	• LL • IMS_zz

9. Create elements with compound types.

   These elements bring together a number of lower-level elements. Because you added the simple elements to the compound type, when you create the compound element, those simple elements are automatically associated with it. You can give them any name and identifier you want.

   Table 6 summarizes the compound element names, types, and identifiers used in the import files in the SupportPac.

   The XML descriptor tag for the element in the message must match the identifier used in the message repository definition. For example, the element Store Details Element has an identifier storedetailselement and is represented in the message as *storedetailselement*.

Table 6. Retail compound type names, identifiers, and types

Compound element name	Identifier	Type
Store Details Element	storedetailselement	Store Details
Purchases Element	purchaseselement	Purchases
Totals Element	totalselement	Totals
IMS_LLzze	IMS_LLzze	IMS_LLzz

10. Add the compound elements Totals Element, Purchases Element, and Store Details Element (in that order) to the compound type Transaction Log. This pulls all the elements of the receipt message together in a single type.
11. Add the elements Total Item Quantity, Purchases Element, Time, Date, Branch Number, and Store Name (in that order) to the compound type Output Transaction Log. This pulls all the elements of the stock message together in a single type.
12. Add the simple elements Total Sales, Date, and Branch number and the compound element IMS_LLzze to the compound type Ims Message Type. This pulls all the elements of the IMS message together in a single type.
13. Create a message of type imsmsgtype. Give it any name or identifier you like. In the import files in the SupportPac, the message name is IMS Message and the identifier is imsmsg. You must name the identifier in the Compute node that sets up the message for transmission to IMS.
14. Create a message of type transactionlog. Give it any name or identifier you like. In the import files in the SupportPac, the message name is Receipt

Message and the identifier is receiptmsg. You must name the identifier either on the MQInput node or in the MQRFH2 header.

15. Create a message of type outputtransactionlog. Give it any name or identifier you like. In the import files in the SupportPac, the message name is Stock Message and the identifier is stockmsg. You must name the identifier either on the MQInput node or in the MQRFH2 header.
16. Make the Purchases Element a repeating element. Make sure that the types transactionlog and outputtransactionlog are checked out. In the Receipt Message, check out the Purchases Element and open its properties pages. On the Connection tab, change **Repeat** to yes. Click **Apply**. Repeat this step for the Stock Message.
17. Set the Custom Wire Format for the IMS variable string definition. Check out the compound type IMS_LLzz, open the properties pages of the LL element within that type, click on the Custom Wire Format tab. Set the Length Count to 2. Repeat for the IMS_zz element.
18. Create a category to contain the messages. This is optional but might be useful if you want to experiment with the functions to generate documentation about the message set. You can give the category any name and identifier you like. In the import files in the SupportPac, the category name is Transaction Log Messages and the identifier is transactionlogmsgs.

    Add the receipt message and the stock message to the category.
19. Save the definitions to the shared repository. Select **File —> Check In —>All (Save to Shared)**.

Creating and configuring the message flows

The files `MRMscenario.xml` and `xmlscenario.xml` contain the definitions of the main retail message flow and the subflow used within the main flow. The first file uses the MRM-defined message, the second uses the XML message. Each base message flow also has an associated subscriptions flow in a separate file. The message flows are shown in Figure 2 and Figure 3.

"Creating and configuring the message flows" describes how to define these message flows. The message flows work on messages of a specific format and content. The definition and manipulation of those messages is described in "Creating and completing the message set" on page 5.

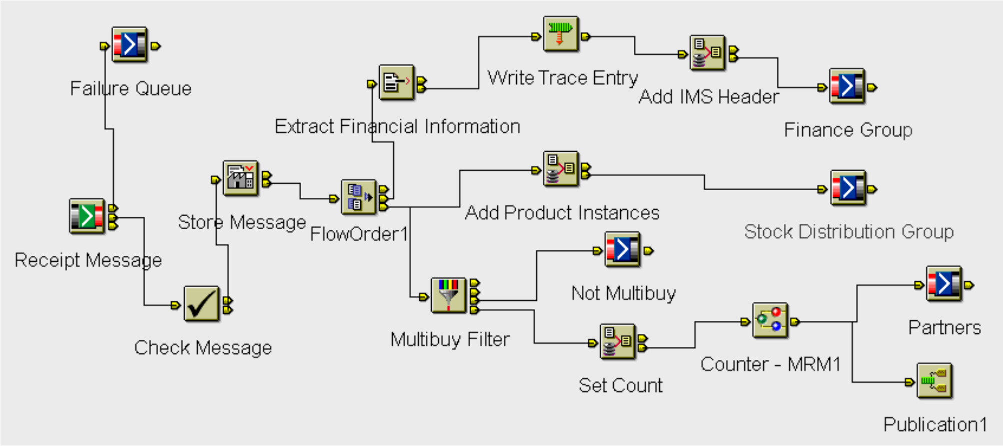


Figure 2. Retail scenario message flow with MRM message

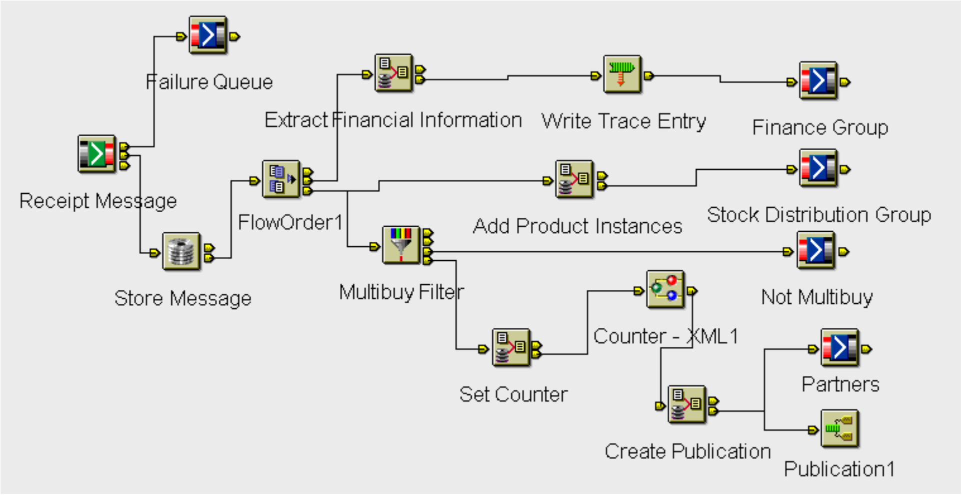


Figure 3. Retail scenario message flow with XML message

You can import and configure either pair of message flows by doing the following:

- Select the Message Flows tab.
- Import the retail message flow definition. To do this:
  1. Select **Import to Workspace...** from the **File** menu.

The **Message Flows** check box is checked by default. This imports only message flows from the files that you identify as the source of the resources to import.

2. Locate the files that you need to run the scenario:
   a. `MRMscenario.xml` and `RegisterSubscription_MRM.xml` if you want to use this scenario with an MRM message
   b. `xmlscenario.xml` and `RegsubXML.xml` if you want to use this scenario with a self-defining XML message

   You can type in the full path and file name in the dialog, or you can click **Browse** and search for these files.
3. Click **Import**. The Control Center processes the file and imports the message flows that are defined within it.

   The file `MRMscenario.xml` contains the main message flow **MRMscenario** and the subflow **Counter – MRM**.

   The file `RegisterSubscription_MRM.xml` contains the message flow **Register subscription – MRM**.

   The file `xmlscenario.xml` contains the main message flow **xmlscenario** and the subflow **Counter – XML**.

   The file `RegsubXML.xml` contains the message flow **Register subscription – XML**.

   A list of the resources that have been imported are displayed in a dialog when the processing ends.
4. Click **OK** to dismiss the dialog.

Each new message flow appears in the left-hand pane, marked by the blue cube icon that indicates that the flow is new. Select each new message flow in turn, and follow the instructions provided in this section to complete the configuration of the nodes within the flow. If you have imported the message flows, the majority of the configuration work is already done for you. If you want to do some additional configuration of a node, first double-click on that node, and then click on the Properties attribute.

- When you have completed the configuration updates that are described in the rest of this section, select **File—>Check In—>All (Save to Shared)** to check in the message flows. You must check in the message flows before you can assign them to a broker.

If you do not use one of the files that are provided, you must create the message flows yourself, and complete their design by dragging and dropping the appropriate nodes into the Message Flow Definition pane, using either Figure 2 on page 13 (if you are using MRM messages) or Figure 3 on page 13 (if you are using XML messages) as a guide. You must connect the nodes and configure them as described in the following sections.

When you review and work through the following configuration information, remember to complete only those sections for the message type (MRM or XML) that you are using. You can ignore all information that refers to the message type that you are not using.

The two examples are independent; you can import both if you want.

The main flow

The main flow starts with an MQInput node that retrieves messages for processing. The messages are then passed on through the branches (described in "Introduction" on page 2) before being sent to various queues for further processing by other departments.

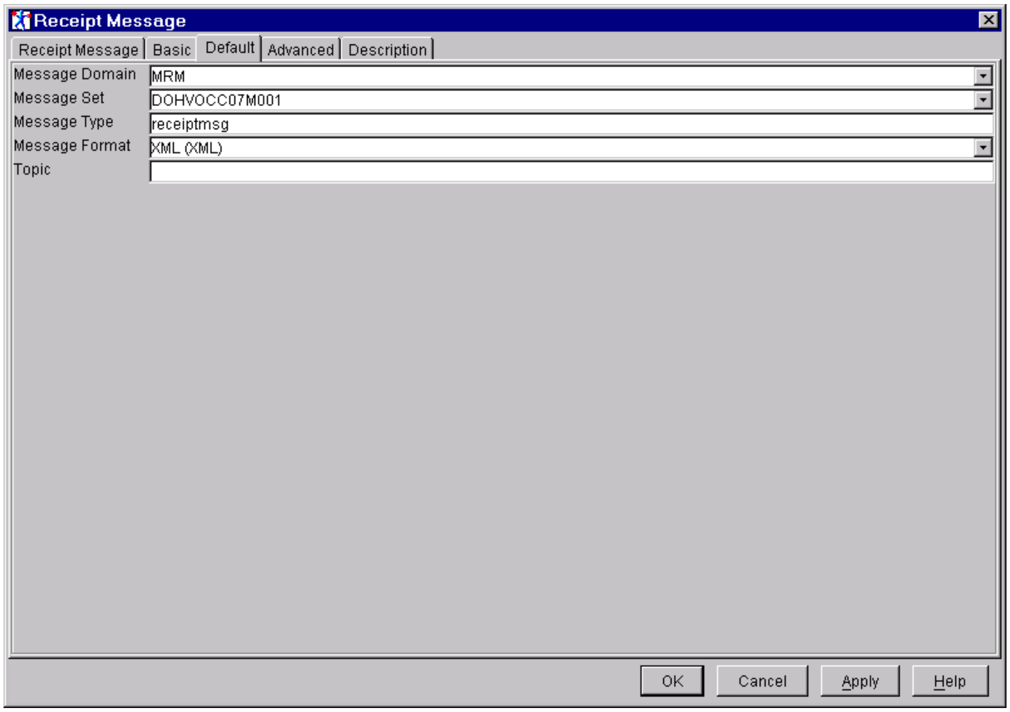
Getting the message

The first node in the message flow, *Receipt Message*, is an MQInput node. This node gets a message from an MQSeries queue on the queue manager hosting the broker.

If you have imported the message flow, this node is fully configured, and identifies input queue IN. If you are using a different queue name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. Select the Basic tab and type the queue name; the default used in the example is IN.
2. If you have an MRM message and you do not specify the message domain, the message set, the message type, the message format, and the topic in the MQRFH2 header, you must specify these on the Default tab of the MQInput node, as shown in Figure 4.



Message Domain: MRM
Message Set: DOHVOCC07M001
Message Type: receiptmsg
Message Format: XML (XML)
Topic:

Figure 4. MQInput node properties

3. Click **OK** to apply your changes.
4. If you are using an MRM message, connect the node's out terminal to the Check node *Check Message*. If you are using an XML message, connect the node's out terminal to the Database node *Store Message*.
5. Connect the node's failure terminal to the MQOutput node *Failure Queue*.

Audit flow

The Audit branch of the message flow for an MRM message contains three nodes: a Check node, *Check Message*, a Warehouse node, *Store Message*, and a FlowOrder node, *FlowOrder1*. The nodes in this part of the message flow check that the incoming message belongs to the expected message set and can therefore be passed through the rest of the message flow, store the receipt information in a database for retrieval later, and control the sequence of processing of subsequent branches.

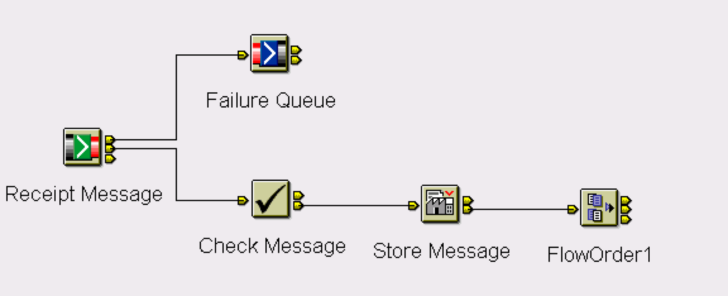
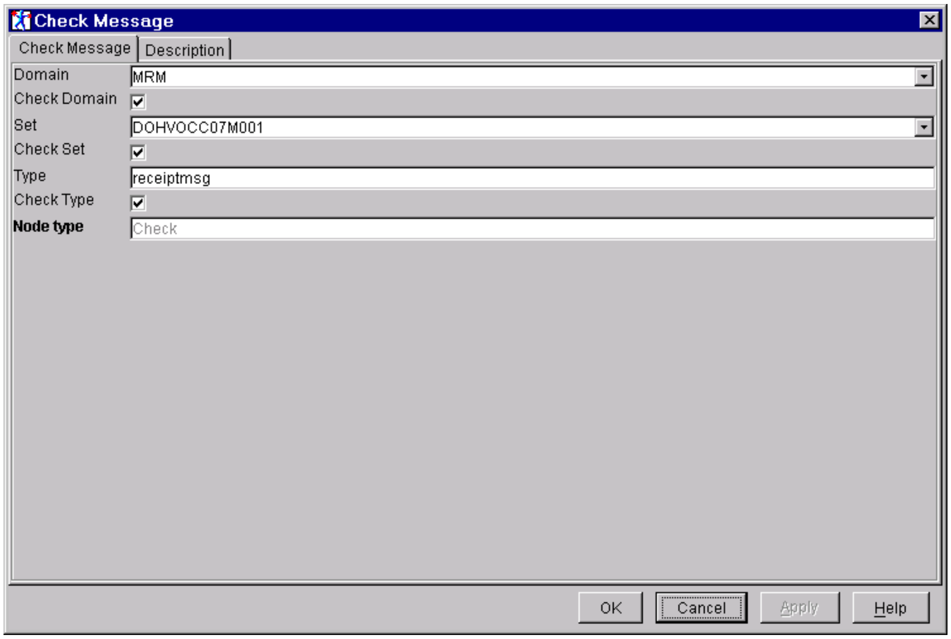


Figure 5. Audit message processing nodes for the MRM message

The Audit branch of the flow for an XML message contains a Database node instead of the Warehouse node, because you can use a Warehouse node only with an MRM message, and the same FlowOrder node. This is illustrated in Figure 3 on page 13. No checking is necessary because there is no defined message set, therefore there is no Check node.

Checking the message



Domain	MRM
Check Domain	☑
Set	DOHVOCC07M001
Check Set	☑
Type	receiptmsg
Check Type	☑
Node type	Check

Figure 6. Check node properties

The Check node *Check Message* is included in the MRM message flow. It is not included in the XML message flow.

If you have imported the MRM message flow, this node is fully configured. If you create the message flow yourself, you must:

1. Configure the node properties as shown in Figure 6 on page 16. The message set number is the identifier for the message set, and the one that is shown here is the identifier for the imported message set. If you have created your own message set following the instructions in "Creating the message set by hand" on page 7, you must type the identifier that you specified for the message.
2. Click **OK** to apply your changes.
3. Connect the match terminal of this node to the node *Store Message*.

   The *Store Message* node is a Warehouse node for the MRM message flow, but a Database node for the XML message flow.

Storing the message

This part of the message flow stores the message in a database for audit purposes. The MRM message flow uses a Warehouse node, but the XML message flow uses a Database node.

Configuring the Warehouse node for the MRM message: You can use a Warehouse node only with an MRM message. The Warehouse node *Store Message* stores the message as a binary object with a timestamp in the MYDB database.

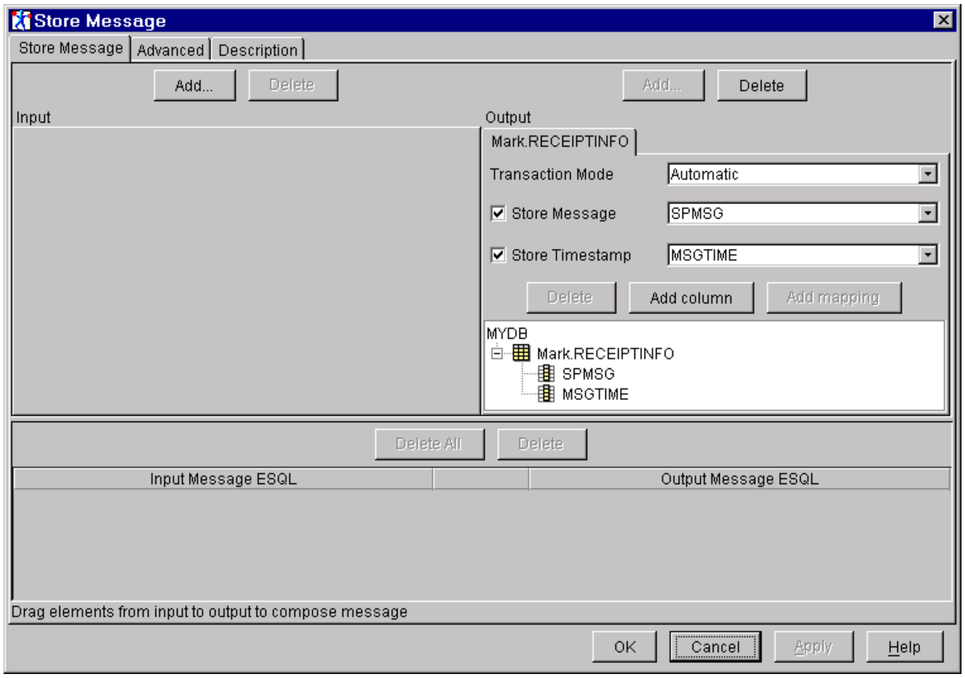
If you have imported the MRM message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** to remove the data source and table RECEIPTINFO from the Output pane. Click **Add** to add the data source MYDB and the correct table schema for the RECEIPTINFO table; for example, BKUSER1.RECEIPTINFO.
2. Add the columns SPMSG and MSGTIME.
3. Check the box to **Store Message** and select the column SPMSG from the drop-down list.
4. Check the box to **Store Timestamp** and select the column MSGTIME from the drop-down list.
5. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click the **Add** button in the Output pane and add the data source MYDB and the correct table schema for the RECEIPTINFO table (for example, BKUSER1.RECEIPTINFO).
2. Add the columns SPMSG and MSGTIME.
3. Check the box to **Store Message** and select the column SPMSG from the drop-down list.
4. Check the box to **Store Timestamp** and select the column MSGTIME from the drop-down list.
5. Click **OK** to apply your changes.
6. Connect the out terminal of this node to the FlowOrder node, *FlowOrder1*.

Note: You do not need to add the input message, *receiptmsg*, because you are storing the entire message, and do not need to refer to elements within the message. However, if you want to change this node to warehouse a subset of the input message, or if you want to include the message to document its use in this node, you must add this on the Input pane.



Transaction Mode: Automatic

☑ Store Message: SPMSG

☑ Store Timestamp: MSGTIME

MYDB
- Mark.RECEIPTINFO
  - SPMSG
  - MSGTIME

Drag elements from input to output to compose message

Figure 7. Warehouse node properties (MRM message)

You can check whether the message is stored in the Warehouse. For example, in a DB2 Command Window type:

```
db2 connect to MYDB
db2 select * from receiptinfo
```

You cannot see the text of the message because of the way it is stored (as a BLOB), but you can see the timestamp at the bottom.

Configuring the Database node for the XML message: If you are using an XML message, you store the message using Database node *Store Message*.

If you have imported the XML message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** to remove the database table RECEIPTINFO2 from the Output pane.
2. Click **Add** and add the data source MYDB and the correct table schema for the RECEIPTINFO2 table (for example, BKUSER1.RECEIPTINFO2).
3. Add all the columns in the table to this pane.
4. Edit the ESQL in the lower part of the dialog to put the correct schema name in the initial clause. For example:

   ```
   INSERT INTO Database.BKUSER1.RECEIPTINFO2 (Storename, Branchnum, Cashiernum,
   Tillnum, Date, Time, Itemname, Itemcode, Itemprice, Itemquantity, Change,
   Multibuy, Totalsales, Totalitems)

   (and so on)
   ```
5. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click **Add** and add the data source MYDB and the correct table schema for the RECEIPTINFO2 table (for example, BKUSER1.RECEIPTINFO2).
2. Add all the columns in the table to this pane.

3. Create a database schema for every element of the message, and itemize every element in the ESQL used in the node to insert values into database columns. You must specify the correct schema in this ESQL . Part of the ESQL is shown below; you must replace BKUSER1 with your required schema value.

   ```
   INSERT INTO Database.BKUSER1.RECEIPTINFO2 (Storename, Branchnum, Cashiernum,
   Tillnum, Date, Time, Itemname, Itemcode, Itemprice, Itemquantity, TotalItems,
   Multibuy, Totalsales, Change)
   VALUES(Body.Message.receiptmsg.transactionlog.storedetailselement.storename,
   Body.Message.receiptmsg.transactionlog.storedetailselement.branchnum,
   Body.Message.receiptmsg.transactionlog.storedetailselement.cashiernum,
   Body.Message.receiptmsg.transactionlog.storedetailselement.tillnum,

   (and so on)
   ```
4. Click **OK** to apply your changes.
5. Connect the out terminal of this node to the FlowOrder node *FlowOrder1*.

Controlling the order of processing

The FlowOrder node *FlowOrder1* does not process a message itself, and has no properties, but it allows you to control the order of processing in the branches of the message flow that follow the node.

The node has two output terminals, first and second, that ensure that the message is propagated to the nodes following on from the first terminal, and that the first branch of processing has completed, before it propagates the message to the second terminal. It has a third output terminal, failure, that is not used in this example.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Connect the first terminal of this node to the Extract node *Extract Financial Information*, the first node in the Finance branch of the flow.

   This ensures that the Finance branch of the flow completes before the message is propagated to the other nodes for further processing.
2. Connect the second terminal of this node to the nodes *Add Product Instances*, which is the first node in the Stock branch of the flow, and *Multibuy Filter*, which is the first node in the Partners branch of the flow.

Finance flow

The Finance department wants to receive only part of the information from the receipt message. Therefore, the required parts of the receipt message (branch number, date, and total sales information) are extracted from the receipt message and used to produce a new message that is passed on for further processing.

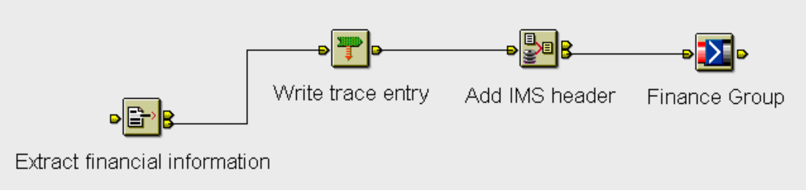


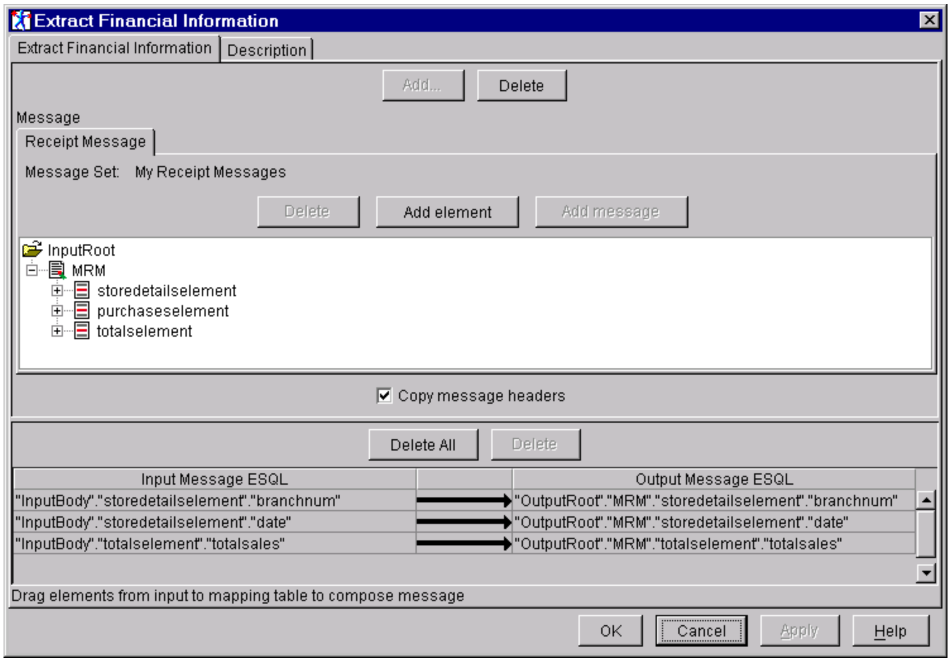
Figure 8. Finance message processing nodes for the MRM message

This is done in the node *Extract Financial Information*. For an MRM message, an Extract node is used. However, an Extract node cannot be used for messages that are not predefined. Therefore, for an XML message, a Compute node is used. The information that is used by the Finance department is recorded by the Trace node *Write Trace Entry*. An IMS header is added to the MRM message by the Compute node *Add IMS Header* before the message is passed to the Finance department through the MQOutput node *Finance Group*. The XML message is passed directly from the Trace node to the MQOutput node *Finance Group*.

Extracting financial information from the MRM message

The Extract node *Extract Financial Information* node takes the information required by the Finance department from the input MRM message to create a new MRM output message.

If you have imported the MRM message flow, this node is fully configured, and its properties appear as shown in Figure 9.



Message Set: My Receipt Messages

☑ Copy message headers

Input Message ESQL		Output Message ESQL
"InputBody"."storedetailselement"."branchnum"	→	"OutputRoot"."MRM"."storedetailselement"."branchnum"
"InputBody"."storedetailselement"."date"	→	"OutputRoot"."MRM"."storedetailselement"."date"
"InputBody"."totalselement"."totalsales"	→	"OutputRoot"."MRM"."totalselement"."totalsales"

Drag elements from input to mapping table to compose message

Figure 9. Extract node properties

If you are creating the message flow yourself, you must:

1. Click **Add** and select the message set called My Receipt Messages and the message called Receipt Message.
2. Expand the elements storedetailselement and totalselement, and drag branchnum, date, and totalsales into the mapping window below.
3. Click **OK** to apply your changes.
4. Connect the out terminal of this node to the Trace node *Write Trace Entry*.

Extracting financial information from the XML message

The Compute node *Extract Financial Information* node takes the information required by the Finance department from the input XML message to create a new XML output message.

If you have imported the XML message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use the following ESQL:
   ```
   DECLARE I INTEGER;
   SET I = 1;
   WHILE I < CARDINALITY(InputRoot.*[]) DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
   END WHILE;
   SET
   OutputRoot.XML.Message.receiptmsg.transactionlog.storedetailselement.branchnum
   = InputRoot.XML.Message.receiptmsg.transactionlog.storedetailselement.branchnum;
   SET OutputRoot.XML.Message.receiptmsg.transactionlog.storedetailselement.date
     = InputRoot.XML.Message.receiptmsg.transactionlog.storedetailselement.date;
   SET OutputRoot.XML.Message.receiptmsg.transactionlog.totalselement.totalsales
     = InputRoot.XML.Message.receiptmsg.transactionlog.totalselement.totalsales;
   ```
3. Click **OK** to apply your changes.
4. Connect the out terminal of the Compute node to the Trace node *Write Trace Entry*.

Use MQSeries Explorer on Windows NT to look at the extracted message; select the message on the output queue, click **Properties**, and then the Data tab.

Writing a trace entry

The Trace node *Write Trace Entry* writes a trace entry according to the pattern that you define for the node properties. This can include any text that appears as comments (documentation), and values that are substituted from the message (for example, field contents). The values to be substituted must be surrounded by the characters `${`*`value`*`}`.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set the Destination to *File*.
2. Type the file path and name. The destination file used by the imported examples is `mytrace` in location `c:\$user\trace`
3. Type the ESQL pattern that is required to write to a file a trace entry that contains the three extracted elements and a simple timestamp.

   a. For the MRM message:

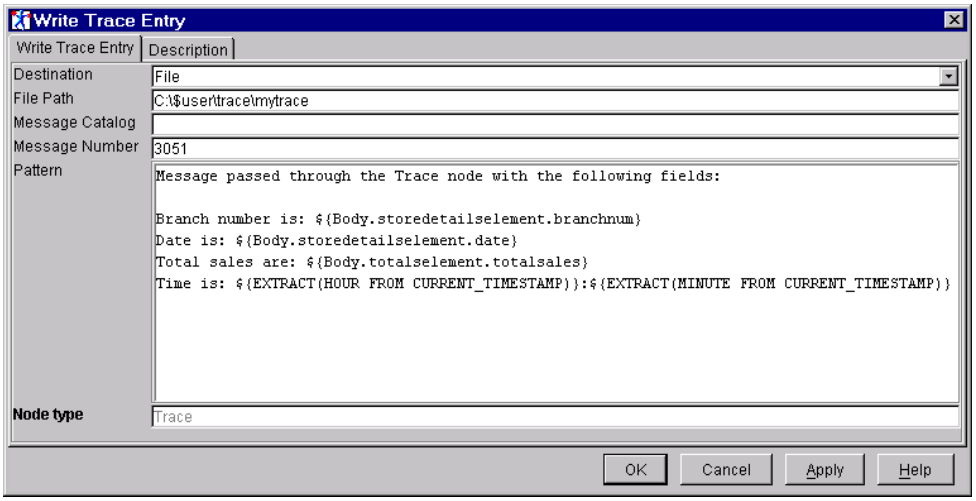
      ```
      Message passed through the Trace node with the following fields:
      Branch number is: ${Body.storedetailselement.branchnum}
      Date is: ${Body.storedetailselement.date}
      Total sales are: ${Body.totalselement.totalsales}
      Time is: ${EXTRACT(HOUR FROM CURRENT_TIMESTAMP)}:${EXTRACT(MINUTE FROM CURRENT_TIMESTAMP)}
      ```

   b. For the self-defining XML message:

      ```
      Message passed through the Trace node with the following fields:
      Branch number is: ${Body.XML.Message.receiptmsg.transactionlog.storedetailselement.branchnum}
      Date is: ${Body.XML.Message.receiptmsg.transactionlog.storedetailselement.date}
      Total sales are: ${Body.XML.Message.receiptmsg.transactionlog.totalselement.totalsales}
      Time is: ${EXTRACT(HOUR FROM CURRENT_TIMESTAMP)}:${EXTRACT(MINUTE FROM CURRENT_TIMESTAMP)}
      ```

4. Click **OK** to apply your changes.
5. Connect the out terminal of this node to:
   a. the Compute node *Add IMS Header*, if you are using an MRM message
   b. the MQOutput node *Finance Group*, if you are using an XML message

Figure 10 illustrates the node properties set up for the MRM message. The property Destination has been changed to File and a location for the trace file has been specified.



Destination: File
File Path: C:\$user\trace\mytrace
Message Catalog:
Message Number: 3051
Pattern:

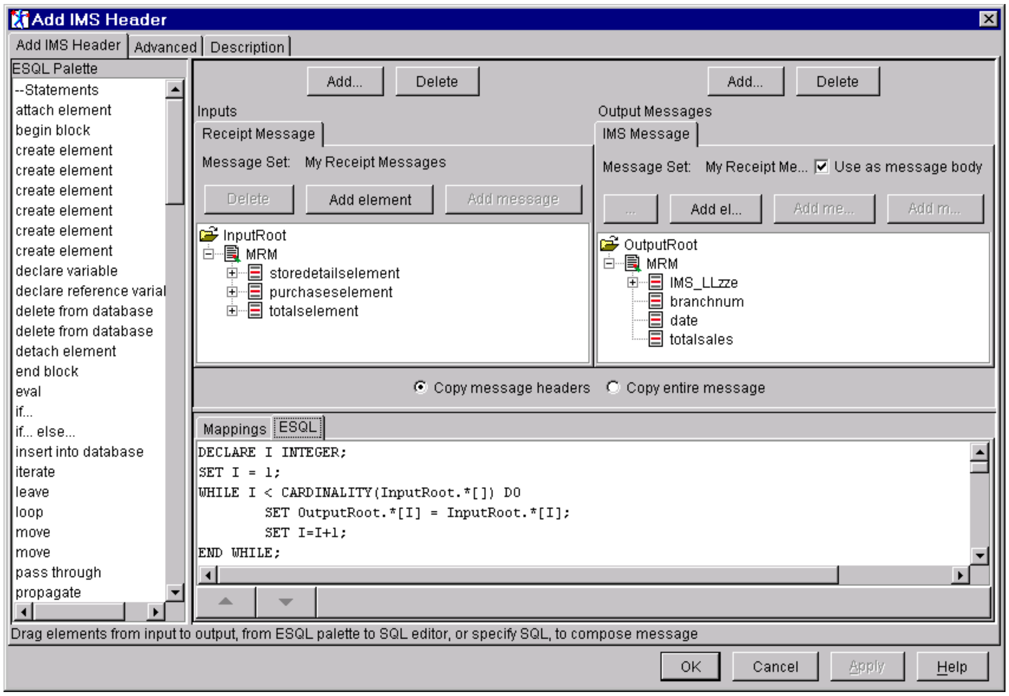
```
Message passed through the Trace node with the following fields:

Branch number is: ${Body.storedetailselement.branchnum}
Date is: ${Body.storedetailselement.date}
Total sales are: ${Body.totalselement.totalsales}
Time is: ${EXTRACT(HOUR FROM CURRENT_TIMESTAMP)}:${EXTRACT(MINUTE FROM CURRENT_TIMESTAMP)}
```

Node type: Trace

Figure 10. Trace node properties

Adding an IMS header to the MRM message

The Compute node *Add IMS Header* restructures the input message into the IMS format (IMS Variable string), removes the MQRFH2 header, adds an IMS header, and resets the encoding and code page for the z/OS system. These actions transform the message to allow it to pass from WebSphere MQ Integrator to IMS.



Message Set: My Receipt Messages

Message Set: My Receipt Me... ☑ Use as message body

● Copy message headers ○ Copy entire message

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I=I+1;
END WHILE;
```

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

Figure 11. Compute node properties

Figure 11 shows the properties of the *Add IMS header* Compute node.

However, this example does not require access to an IMS system to be deployed: the MQOutput node for this flow is defined as a local queue. You can add the extra complexity to this flow if it is appropriate for your needs. The node is included to illustrate how you can transform a message to pass it to a different system that has different requirements on the data content. For further information about data conversion using WebSphere MQ Integrator nodes as an alternative to MQSeries data conversion exits, see the *WebSphere MQ Integrator ESQL Reference* book.

If you want to view the message content after the transformation, you can insert a Trace node after *Add IMS header* and write the entire message contents to a trace file using the pattern `${Root}`.

If you have imported the MRM message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click **Add** to add Receipt Message to the Inputs pane.
2. Click **Add** to add IMS Message to the Output Messages pane.
3. Select the **Use as message body** option on the Output Messages pane.
4. Select **Copy message headers**.
5. Select the ESQL tab, and type the following to set up the message and header contents. You must check that the MessageSet identifier in the ESQL is correct for your message set:

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) -1 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.
SET "OutputRoot"."MQMD"."CodedCharSetId" = 500;
SET "OutputRoot"."MQMD"."Encoding" = 785;
SET "OutputRoot"."MQMD"."Format" = 'MQIMS   ';
--
SET "OutputRoot"."MQIIH"."StrucId" = 'IIH ';
SET "OutputRoot"."MQIIH"."Version" = 1;
SET "OutputRoot"."MQIIH"."StrucLength" = 84;
SET "OutputRoot"."MQIIH"."Encoding" = 785;
SET "OutputRoot"."MQIIH"."CodedCharSetId" = 500;
SET "OutputRoot"."MQIIH"."Format" = 'MQIMSVS ';
SET "OutputRoot"."MQIIH"."Flags" = 0;
SET "OutputRoot"."MQIIH"."LTermOverride" = 'REG1IMS ';
SET "OutputRoot"."MQIIH"."MFSMapName" = '        ';
SET "OutputRoot"."MQIIH"."ReplyToFormat" = 'MQIMSVS ';
SET "OutputRoot"."MQIIH"."Authenticator" = '        ';
SET "OutputRoot"."MQIIH"."TranInstanceId" = X'00000000000000000000000000000000';
SET "OutputRoot"."MQIIH"."TranState" = ' ';
SET "OutputRoot"."MQIIH"."CommitMode" = '0';
SET "OutputRoot"."MQIIH"."SecurityScope" = 'C';
SET "OutputRoot"."MQIIH"."Reserved" = ' ';
-- LLzz must be defined in the output MRM message -------
SET "OutputRoot"."MRM"."IMS_LLzze"."LL" = 46;
SET "OutputRoot"."MRM"."IMS_LLzze"."IMS_zz" = 0;
SET "OutputRoot"."MRM"."branchnum"="InputBody"."storedetailselement"."branchnum";
SET "OutputRoot"."MRM"."date"="InputBody"."storedetailselement"."date";
SET "OutputRoot"."MRM"."totalsales"="InputBody"."totalselement"."totalsales";
-- input message -------
SET "OutputRoot"."Properties"."MessageSet" = 'DOHVOCC07M001';
SET "OutputRoot"."Properties"."MessageType" = 'imsmsg';
-- The message format specified below is the message format identifier specified in the
-- Custom Wire Format of the Message set properties.
SET "OutputRoot"."Properties"."MessageFormat" = 'CWF';
--
```

The WHILE loop in this ESQL removes the MQRFH2 header from the incoming message by specifying `CARDINALITY(InputRoot.*[]) -1`.

Note the use of the CWF message format in the final line of ESQL: this is the wire format used by the output IMS message. Therefore this node illustrates the possibility of switching from one format (XML on input) to another (CWF on output).

6. Click **OK** to apply your changes.
7. Connect the out terminal of this node to the node *Finance Group*.

Outputting the Finance message

The message is passed to the Finance Department on an MQSeries queue, finance, to which the MQOutput node *Finance Group* puts it.

If you have imported the message flow, this node is fully configured, and the output queue is identified as queue FINANCE on queue manager MQSI_SAMPLE_QM. If you are using different values, you must update the node properties.

If you are creating the message flow yourself, you must:

1. Select the Basic tab and type the queue and queue manager names for your output queue. You can leave other values to default.
2. Click **OK** to apply your changes.

Stock flow

The stock branch of the flow is used to add up all instances of an item sold, and pass this information to the Distribution group to maintain stock levels. For example, if a shopper buys two bottles of shampoo, the receipt contains two instances of shampoo.

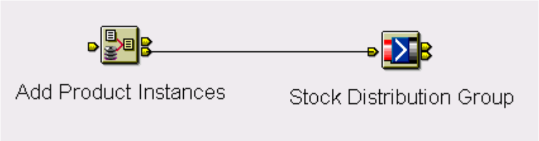


Figure 12. Stock message processing nodes

Using the stock flow with an MRM message

The Compute node *Add Product Instances* extracts the store name, branch number, date, time, and purchases details from the MRM message and constructs a new output message that can be passed on for further processing. It illustrates both how you can use the drag-and-drop capabilities of the node to map selected elements from an input message (Receipt Message) onto a different output message (Stock Message), and how to calculate the total item quantity.

If you have imported the MRM message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click **Add** to add the message Receipt Message in the message set My Receipt Messages to the Inputs pane.
2. Click **Add** to add the message Stock Message in the message set My Receipt Messages to the Output Messages pane.
3. Select **Use as message body**.
4. Select **Copy message headers**
5. Expand storedetailselement and purchaseselement within the input message.

   Drag simple elements storename, branchnum, date, and time of storedetailselement from the input message onto their equivalent in the output message. Do the same for elements itemname, n_reserve,itemcode, itemprice, and itemquantity of purchaseselement. You can see the mappings build up on the Mappings tab.
6. On the ESQL tab, edit the ESQL as shown below. You must check that the MessageSet identifier used in the ESQL is correct for your message set. Much of the ESQL has been generated for you already by the selections you made on the node properties and by the mappings.

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
     SET OutputRoot.*[I] = InputRoot.*[I];
     SET I=I+1;
END WHILE;
SET "OutputRoot"."MRM"."storename" =
 "InputBody"."storedetailselement"."storename";
SET "OutputRoot"."MRM"."branchnum" =
 "InputBody"."storedetailselement"."branchnum";
SET "OutputRoot"."MRM"."date" =
 "InputBody"."storedetailselement"."date";
SET "OutputRoot"."MRM"."time" =
 "InputBody"."storedetailselement"."time";

DECLARE stop INTEGER;
DECLARE countitems INTEGER;
DECLARE current INTEGER;

SET stop = CARDINALITY("InputBody"."purchaseselement"[]);
SET current = 1;
SET countitems = 0;

WHILE current <= stop DO
SET "OutputRoot"."MRM"."purchaseselement"[current] =
  "InputBody"."purchaseselement"[current];
IF "InputBody"."purchaseselement"[current]."itemname" = 'Shampoo' THEN
SET countitems = countitems +
  "InputBody"."purchaseselement"[current]."itemquantity";
END IF;
SET current = current + 1;
END WHILE;

SET "OutputRoot"."MRM"."outputtotalselement"."totalitemquantity" = countitems;
SET "OutputRoot"."Properties"."MessageSet" = 'DOHVOCC07M001';
SET "OutputRoot"."Properties"."MessageType" = 'stockmsg';
```

7. Click **OK** to apply your changes.
8. Connect the out terminal of this node to the node *Stock Distribution Group*.

Using the stock flow with an XML message

The Compute node *Add Product Instances* adds up the instances of each product specified in the XML message and puts the value into a new field (totalitemquantity) in the message being output from the node.

If you have imported the XML message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select the ESQL tab and type the following statements:

```
SET OutputRoot = InputRoot;
DECLARE TotalItemQuantity INTEGER;
SET TotalItemQuantity = (SELECT SUM(CAST(T.itemquantity AS INT))
FROM InputBody.Message.receiptmsg.transactionlog.purchaseselement[] AS T
WHERE CAST(T.itemname AS CHAR) = 'Shampoo');
SET
"OutputRoot"."XML"."Message"."receiptmsg"."transactionlog"."totalselement"."totalitemquantity"
  = TotalItemQuantity;
```

This declares a new element called TotalItemQuantity as an integer and sets its value to the sum of ItemQuantity where the ItemName is (in this example) Shampoo. The TotalItemQuantity element is placed within the Totals compound element in the output message.

Alternatively, you can use the following ESQL using a WHILE loop to output the same message:

```
SET OutputRoot = InputRoot;
DECLARE TotalItemQuantity INTEGER;
SET TotalItemQuantity = 0;
DECLARE current INTEGER;
DECLARE stop INTEGER;
SET current = 1;
SET stop = CARDINALITY(InputBody.Message.receiptmsg.transactionlog.*[]);

WHILE current <= stop DO
  IF CAST(InputBody.Message.receiptmsg.transactionlog.purchaseselement[current].
  itemname AS CHAR) = 'Shampoo' THEN
     SET TotalItemQuantity =  TotalItemQuantity +
     CAST(InputBody.Message.receiptmsg.transactionlog.purchaseselement[current]itemquantity
 AS INTEGER)
  END IF;
  SET current = current + 1;
END WHILE;
SET "OutputRoot"."XML"."Message"."receiptmsg"."transactionlog"."totalselement"."totalitemquant
  = TotalItemQuantity;
```

This loops through the receipt message increasing the value of TotalItemQuantity by one each time it comes across an instance of Shampoo, therefore adding up all instances of the Shampoo product in the receipt. Again, the TotalItemQuantity element is placed within the Totals compound element in the output message.

2. Click **OK** to apply your changes.
3. Connect the out terminal of this node to the node *Stock Distribution Group*.

Outputting the Stock message

The MQOutput node *Stock Distribution Group* puts the message to an MQSeries queue, from which the Stock Distribution Group can retrieve it.

If you have imported the message flow, this node is fully configured, and the output queue is identified as queue STOCK on queue manager MQSI_SAMPLE_QM. If you are using different values, you must update the node properties.

If you are creating the message flow yourself, you must:

1. Select the Basic tab and type the queue and queue manager names for your output queue. You can leave other values to default.
2. Click **OK** to apply your changes.

Partner Flow

The partner branch of the flow is used to track and keep details of products that are selling well. If more than one of the same product is bought on the same transaction, this is called a ‘multibuy’. Each multibuy record is placed into a database for easy access and reference by partners.

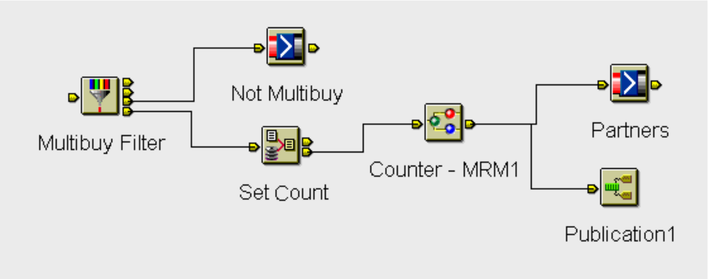


Figure 13. Partner message processing nodes for the MRM message

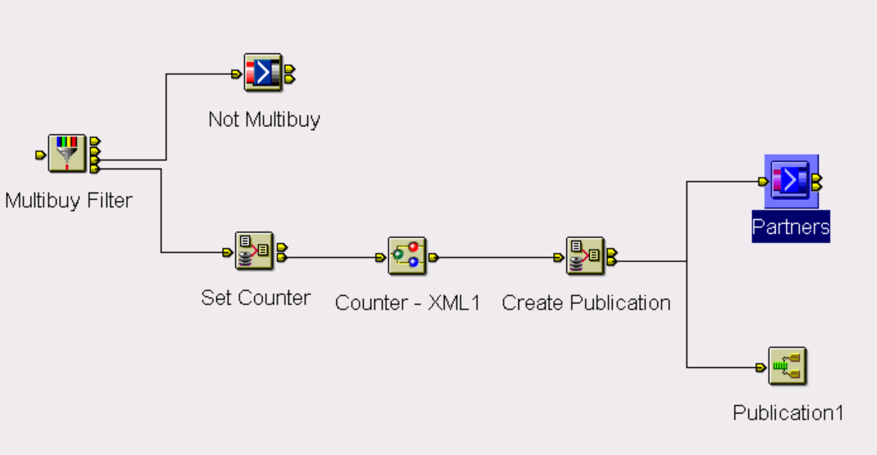


Figure 14. Partner message processing nodes for the XML message

The message flow contains a Filter node to filter ‘multibuy’ records and a Database node to insert the records into the Multibuy database for partners.

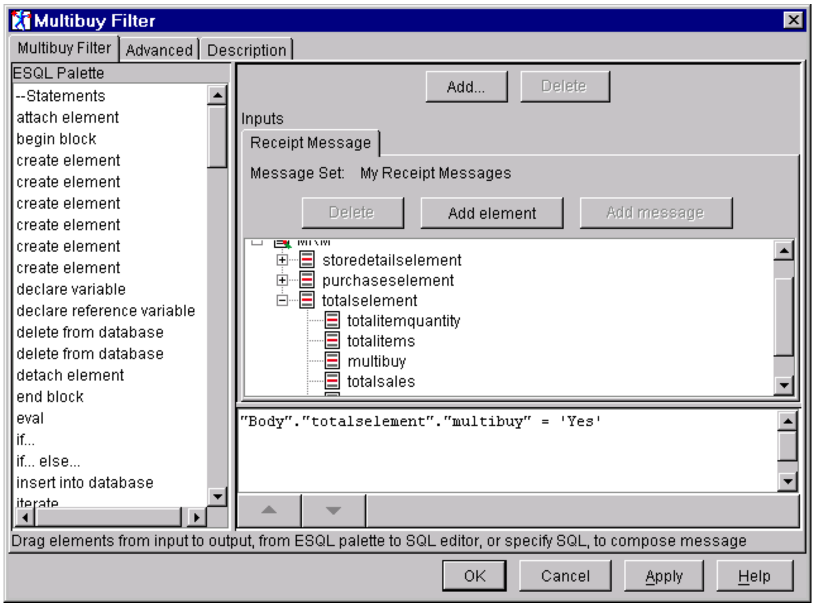
Database updates are completed in a loop within the flow: this allows the database to be updated for every item in the receipt message. The loop is defined as a subflow (*Counter – MRM* for MRM messages, *Counter – XML* for XML messages).

The messages are also published through a Publication node. If the message is an XML message, a Compute node is used to add the required MQRFH2 header information to the message before routing it to the Publication node. The MRM message used in this scenario has the MQRFH2 header in it already.

Partners subscribe to the publications by filtering on the content of the message (the value of the itemname field).

Filtering multibuy records

The Filter node *Multibuy Filter* is set up to filter all messages with the value `Yes` in the `Multibuy` field to the Multibuy database.



Message Set: My Receipt Messages

```
"Body"."totalselement"."multibuy" = 'Yes'
```

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

Figure 15. Filter node properties showing the set up for the MRM message

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Configure the filter node for an MRM message:
   a. Click **Add** above the Inputs pane and select the message set My Receipt Messages and the message Receipt Message.
   b. Expand totalselement. Drag and drop the multibuy element into the filter field in the lower part of the dialog.
   c. Edit the ESQL by adding `='Yes'` to the expression that was generated by the drag and drop (as shown in Figure 15).
2. Configure the filter node for an XML message by entering the following ESQL in the filter field in the lower part of the dialog:

   ```
   Body.Message.receiptmsg.transactionlog.totalselement.multibuy = 'Yes'
   ```
3. Click **OK** to apply your changes.
4. Connect the true terminal of this node to the node *Set Count*.
5. Connect the false terminal of this node to the node *Not Multibuy*.

Handling messages that do not contain multibuys

The MQOutput node *Not Multibuy* identifies an MQSeries queue to which messages are put when their “multibuy” value is “no”.

If you have imported the message flow, this node is fully configured, and the output queue is identified as NOTMULTI on queue manager MQSI_SAMPLE_QM. If you are using different values, you must update the node properties.

If you are creating the message flow yourself, you must:

1. Select the Basic tab and type the queue and queue manager names for your output queue. You can leave other values to take default values.
2. Click **OK** to apply your changes.

Updating the database

Every item in the receipt message for which more than one has been purchased is recorded in the Multibuy database.

This process is handled by a loop to search the entire receipt to record all such purchases. The loop is created as a subflow. Before the loop can be entered, a loop counter is initialized by the Compute node *Set Count*.

Creating the loop subflow: If you have imported the message flow, the subflow for the loop has already been imported and created as part of the main message flow.

If you are creating your own message flow, you must:

1. Right-click on the Message Flows root and create a new, empty message flow. You can choose any name for this subflow. The names used by the import files are *Counter – MRM* (if you are using MRM messages) or *Counter – XML* (if you are using XML messages).
2. Drag and drop the following nodes from the set of IBM Primitives in the tree to the right-hand Message Flow Definition pane:
   a. An Input Terminal node *InTerminal1*. This is the node through which input messages are received by the subflow. It performs no message processing.
   b. A Filter node *Test Counter*. This is configured to test the value of the flag to determine if the message is to pass through the loop once more, or is to return to the main message flow.
   c. A Compute node *Prepare For Database*. This prepares values for corresponding columns in the Multibuy database.
   d. A DataInsert node (for MRM messages) or a Database node (for XML messages) *Update Multibuy Database*. This inserts values into the Multibuy database table.
   e. A Compute node *Reset Counter*. This resets the flag counter and connects back to *Test Counter*.
   f. An Output Terminal node *OutTerminal1*. This is the node through which messages are returned to the main message flow. It performs no message processing.
3. Connect up these nodes as shown in Figure 16 on page 31.
4. Select the main message flow in the left-hand pane. Its contents are displayed in the Message Flow Definition pane.
5. Drag and drop your new subflow into the correct location in the main flow and connect it up in the same way as you do any other node:
   a. Connect the out terminal of the node *Set Counter* to the in terminal of the subflow node *Counter – MRM* or *Counter – XML*.
   b. Connect the out terminal of the subflow node to the in terminal of the node *Publication1* (for MRM messages) or *Create Publication* (for XML messages).
6. Configure each node in the subflow as described in the following sections.

Initializing the loop control flag

The Compute node *Set Counter* sets the initial value for a counter to control the loop.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select **Copy entire message**.
2. If you are using an MRM message:
   a. Click **Add** to add the Receipt Message to the Inputs pane and expand the tree for element totalselement.
   b. Select the ESQL pane.

   c. Type the following ESQL:

      ```
      SET "OutputRoot"."MRM"."totalselement"."f_reserve" = CARDINALITY("InputBody"."purchaseselem
      ```

3. If you are using an XML message, select the ESQL tab and type the following ESQL below the comment line:

   ```
   SET OutputRoot.XML.Message.receiptmsg.transactionlog.totalselement.f_reserve =
     CARDINALITY(InputRoot.XML.Message.receiptmsg.transactionlog.purchaseselement[]);
   ```

4. Click **OK** to apply your changes.
5. Connect the out terminal of the node to the subflow node *Counter – MRM1* or *Counter – XML1*.

Figure 16 illustrates the nodes that make up the loop and their connections.

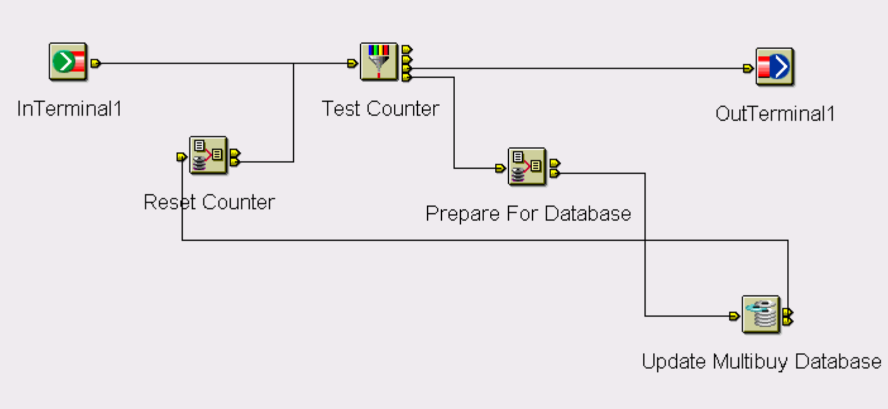


Figure 16. The loop to record data in the database

Testing the value of the loop control flag

Within the counter subflow, the node *Test Counter* ensures the loop is terminated when the total number of multibuy items have been recorded in the database.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set up the ESQL to test the loop counter:
   a. If you are using an MRM message, drag and drop the field f_reserve from the message to the filter field in the lower part of the dialog, and complete the statement as follows:

      ```
      "Body"."totalselement"."f_reserve" > 0
      ```

   b. If you are using an XML message, type the following ESQL in the filter field in the lower part of the dialog:

      ```
      Body.Message.receiptmsg.transactionlog.totalselement.f_reserve > 0
      ```

2. Click **OK** to apply your changes.
3. Connect the true terminal (when the counter indicates the loop must be reiterated) to the node *Prepare For Database*. This ensures the loop is executed one more time.
4. Connect the false terminal (when the counter indicates the loop must be terminated) to node *OutTerminal1*. This returns the message to the main flow node *Publication1* (for MRM messages) or the node *Create Publication* (for XML messages).

Preparing the values for insertion in the database

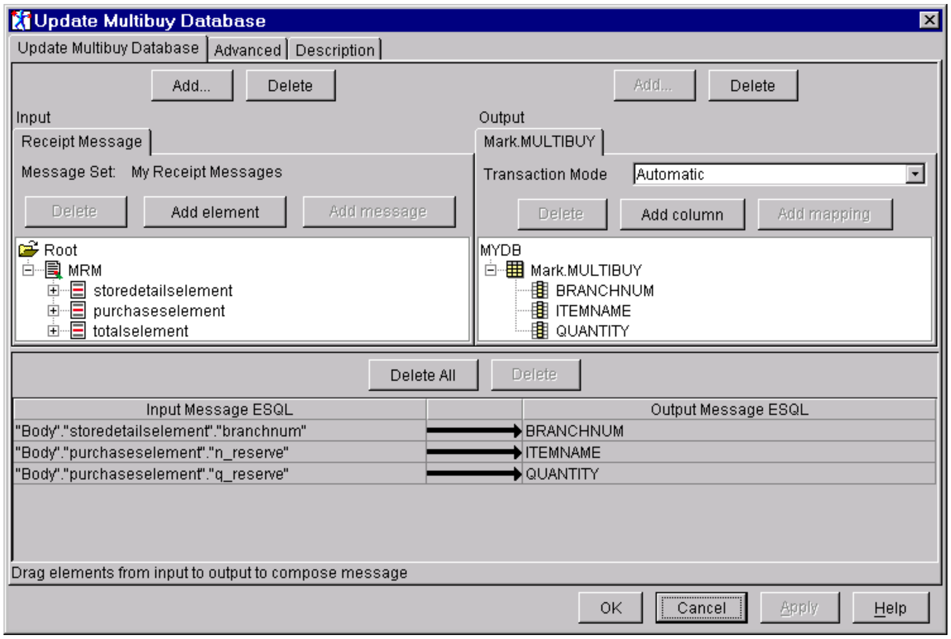
The node *Prepare For Database* is a Compute node that is used to change the data type of some fields in the message before they can be stored in the database.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click **Copy entire message**.
2. Select the ESQL tab and type the ESQL required:
   a. For an MRM message, you need the following statements:
   ```
   DECLARE elementnum INTEGER;
   SET elementnum="InputBody"."totalselement"."f_reserve";
   SET "OutputRoot"."MRM"."purchaseselement"."n_reserve"=
   "InputBody"."purchaseselement"[elementnum]."itemname";
   SET "OutputRoot"."MRM"."purchaseselement"."q_reserve"=
   "InputBody"."purchaseselement"[elementnum]."itemquantity";
   ```
   b. For an XML message, you need the following statements:
   ```
   DECLARE elementnum INTEGER;
   SET OutputRoot = InputRoot;
   SET elementnum =
    CAST(InputRoot.XML.Message.receiptmsg.transactionlog.totalselement.f_reserve AS INTEGER);
   SET OutputRoot.XML.Message.receiptmsg.transactionlog.storedetailselement.n_reserve =
    InputRoot.XML.Message.receiptmsg.transactionlog.purchaseselement[elementnum].itemname;
   SET OutputRoot.XML.Message.receiptmsg.transactionlog.storedetailselement.q_reserve =
    InputRoot.XML.Message.receiptmsg.transactionlog.purchaseselement[elementnum].itemquantity;
   ```
3. Connect the out terminal of this node to the node *Update Multibuy Database*.

Updating the Multibuy database (MRM message)

If you are using an MRM message, the DataInsert node *Update Multibuy Database* inserts information from the message into the database.



Message Set: My Receipt Messages

Transaction Mode: Automatic

Input Message ESQL		Output Message ESQL
"Body"."storedetailselement"."branchnum"	→	BRANCHNUM
"Body"."purchaseselement"."n_reserve"	→	ITEMNAME
"Body"."purchaseselement"."q_reserve"	→	QUANTITY

Figure 17. DataInsert node properties (MRM message)

If you have imported the MRM message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** to remove the database table from the Output pane.

2. Click **Add** to add the data source MYDB and the correct table schema for the MULTIBUY table to the Output pane.
3. Add the columns BRANCHNUM, ITEMNAME, and QUANTITY from the table to this pane.
4. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click **Add** and select the message set My Receipt Messages and the message Receipt Message to add to the Input pane.
2. Click **Add** to add the data source MYDB and the correct table schema for the MULTIBUY table to the Output pane.
3. Add the columns BRANCHNUM, ITEMNAME, and QUANTITY from the table to this pane.
4. Expand storedetailselement and purchaseselement in the input message. Drag and drop branchnum, n_reserve (the itemname), and q_reserve (the quantity) onto the name of the target column.
5. Click **OK** to apply your changes.
6. Connect the out terminal of this node to the node *Reset Counter*.

Updating the Multibuy database (XML message)

If you are using an XML message, the Database node *Update Multibuy Database* updates the database with information from the message.

If you have imported the XML message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** to remove the database table MULTIBUY from the Output pane. Click **Add** to add the data source MYDB and the correct table schema for the MULTIBUY table to the Output pane.
2. Add the columns BRANCHNUM, ITEMNAME, and QUANTITY to this pane.
3. Edit the ESQL to correct the schema name, specifying the required value instead of `BKUSER1` in the following example:
   ```
   INSERT INTO Database.BKUSER1.MULTIBUY(branchnum,itemname,quantity)
   ```
4. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click **Add** to add the data source MYDB and the correct table schema for the MULTIBUY table to the Output pane.
2. Add the columns from the table to this pane.
3. Type the following ESQL, specifying the correct schema name instead of `BKUSER1`:
   ```
   INSERT INTO Database.BKUSER1.MULTIBUY(branchnum,itemname,quantity)
   VALUES(Body.Message.receiptmsg.transactionlog.storedetailselement.branchnum,
   Body.Message.receiptmsg.transactionlog.storedetailselement.n_reserve,
   Body.Message.receiptmsg.transactionlog.storedetailselement.q_reserve);
   ```
4. Click **OK** to apply your changes.
5. Connect the out terminal of this node to the node *Reset Counter*.

Updating the value of the loop control counter

The Compute node *Reset Counter* updates the loop control counter to ensure that the loop is executed the correct number of times.

The retail operation

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click **Copy entire message**.
2. Select the ESQL tab and type the ESQL required:
   a. For an MRM message, you need the following statement:
      ```
      SET "OutputRoot"."MRM"."totalselement"."f_reserve"
       = ("InputRoot"."MRM"."totalselement"."f_reserve")-1;
      ```
   b. For an XML message, you need the following statements:
      ```
      SET OutputRoot = InputRoot;
      SET OutputRoot.XML.Message.receiptmsg.transactionlog.totalselement.f_reserve =
      CAST(InputRoot.XML.Message.receiptmsg.transactionlog.totalselement.f_reserve AS INTEGER)-1;
      ```
3. Click **OK** to apply your changes.
4. Connect the out terminal of this node to the node *Test Counter*.

Completing the Partner flow

When the flow of control returns from the loop subflow into the main flow, the message is ready for publishing.

Creating the publication (XML message only): The Compute node *Create Publication* is used to add an MQRFH2 header to the message to define it as a publication, and to set a topic of “Multibuy”. This addition enables the message to be published in the same way as the MRM message, which already has the MQRFH2 header.

If you have imported the XML message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click **Copy message headers**.
2. On the ESQL tab, use the following ESQL:
   ```
   DECLARE I INTEGER;
   SET I = 1;
   WHILE I < CARDINALITY(InputRoot.*[]) DO
     SET OutputRoot.*[I] = InputRoot.*[I];
     SET I = I+1;
   END WHILE;
   SET OutputRoot.MQRFH2.psc.Command = 'Publish';
   SET OutputRoot.MQRFH2.psc.Topic = 'Multibuy';
   SET OutputRoot.XML = InputBody;
   ```
3. Click **OK** to apply your changes.
4. Connect the out terminal of this node to the node *Publication1* and to the node *Partners*.

Publishing the message: The Publication node *Publication1* routes the messages to the subscribers that have registered an interest in the “Multibuy” topic.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you can leave all properties to assume default values.

Outputting the Partner message: The MQOutput node *Partners* puts the message to an MQSeries queue. This node is used instead of a Publication node so that you can view the messages that are published. You cannot view messages in a Publication node. The MQOutput node therefore lets you test your complete flow by providing a view of the published messages.

If you have imported the message flow, this node is fully configured, and the output queue is identified as the queue PARTNERS on queue manager MQSI_SAMPLE_QM. If you are using different values, you must update the node properties.

If you are creating the message flow yourself, you must:

1. Select the Basic tab and type the queue and queue manager names for your output queue. You can leave other values to take default values.
2. Click **OK** to apply your changes.

Register subscriptions flow

A separate message flow is used to register a subscription. Two versions of this flow are provided. One message flow handles MRM messages; see *Register subscription – MRM* shown in Figure 18. The other message flow handles XML messages; see*Register subscription – XML* shown in Figure 19.

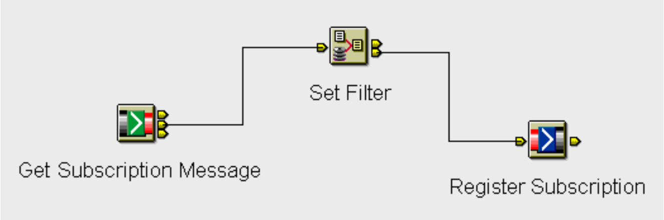


Figure 18. The Register subscriptions flow for the MRM message

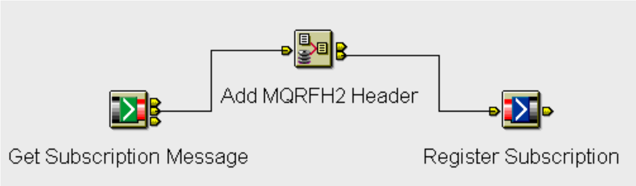


Figure 19. The Register subscriptions flow for the XML message

The example message flow that has been imported shows filtering on the value of a field in the message. This is known as content-based filtering. If you want to subscribe only by topic, remove the filter line. Be aware that if you set up a subscription on a topic of “Multibuy”, you have to delete that subscription if you then want to do a content-based filter from the same subscriber application ID. The second, content-based, subscription does not overwrite the more general topic-based one. You can delete subscriptions from the Control Center Subscriptions view.

Receiving the message

The MQInput node *Get Subscription Message* receives an input message on its defined input queue.

If you have imported the message flow, this node is fully configured and the input queue is identified as queue SUBIN. If you are using a different queue name, you must update the node properties.

The retail operation

If you are creating the message flow yourself, you must:

1. Specify the name of the queue from which the messages are read by the MQInput queue. The default queue name is SUBIN. You can leave all other properties to take default values.
2. Click **OK** to apply your changes.
3. Connect the out terminal of this node to node *Set Filter* (for MRM messages) or to *Add MQRFH2 Header* (for XML messages).

Setting up a subscription filter (MRM message)

The Compute node *Set Filter* adds subscription registration information and additional filter criteria into the MQRFH2 header of the MRM message.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Type the following ESQL (note that all quotes used in this example are single quotes):

   ```
   DECLARE I INTEGER;
   SET I = 1;
   WHILE I < CARDINALITY(InputRoot.*[]) DO
     SET OutputRoot.*[I] = InputRoot.*[I];
     SET I = I+1;
   END WHILE;
   SET OutputRoot.MQRFH2.psc.Command = 'RegSub';
   SET OutputRoot.MQRFH2.psc.Topic = 'Multibuy';

   SET OutputRoot.MQRFH2.psc.QMName = 'MQSI_SAMPLE_QM';
   SET OutputRoot.MQRFH2.psc.QName = 'SUBS';

   SET OutputRoot.MQRFH2.psc.Filter
     = 'purchaseselement.itemname = ''Shampoo''';
   ```

   QName and QMName identify the queue on which you want to receive any publication messages that match the subscription.
2. Click **OK** to apply your changes.
3. Connect the out terminal of this node to the node *Register Subscription*.

Setting up a subscription filter (XML message)

The Compute node *Add MQRFH2 Header* adds an MQRFH2 header to the incoming XML message, and sets subscription registration information and additional filter criteria within the new header.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click on **Copy message headers**.
2. Type the following ESQL (note that all quotes used in this example are single quotes):

   ```
   DECLARE I INTEGER;
   SET I = 1;
   WHILE I < CARDINALITY(InputRoot.*[]) DO
     SET "OutputRoot".*[I] = "InputRoot".*[I];
     SET I = I+1;
   END WHILE;
   SET "OutputRoot"."MQRFH2"."psc"."Command" = 'RegSub';
   SET "OutputRoot"."MQRFH2"."psc"."Topic" = 'Multibuy';

   SET "OutputRoot"."MQRFH2"."psc"."QMName" = 'MQSI_SAMPLE_QM';
   SET "OutputRoot"."MQRFH2"."psc"."QName" = 'SUBS';
   ```

```
SET "OutputRoot"."MQRFH2"."psc"."Filter"
  = 'Message.receiptmsg.transactionlog.purchaseselement.itemname = ''Shampoo''';
```

QName and QMName identify the queue on which you want to receive any publication messages that match the subscription.

3. Click **OK** to apply your changes.
4. Connect the out terminal of this node to the MQOutput node *Register Subscription*.

Registering the subscription

The MQOutput node *Register Subscription* puts the subscription message to the broker’s queue SYSTEM.BROKER.CONTROL.QUEUE. If you have set the subscription message type to 1 (MQMT_REQUEST), the broker sends back a response. If you want to see this response, which should be ″Completion OK″, you must update the MQMD of the subscription message to include a ReplyToQ and ReplyToQmgr. The broker sends the response to the subscription message to this queue.

All published messages that match the criteria set in the MQRFH2 header of the subscription message are also put on to the queue named in the MQRFH2 header; QName identifies the name of the queue, and QMName identifies the name of the queue manager.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select the Basic tab and set the Queue Name to SYSTEM.BROKER.CONTROL.QUEUE and the Queue Manager Name to the name of the queue manager hosting the broker; the default queue manager is MQSI_SAMPLE_QM.
2. Click **OK** to apply your changes.

Assigning the message set and message flows

When your message set and message flow definitions are complete, you must check in all the objects that you have been working with before you can assign them to a broker. To do this, select **File—>Check In—>All (Save to Shared)** to check in all objects at once. For other check-in options, see the *WebSphere MQ Integrator Using the Control Center* book.

Assigning the message set to the broker

If you are using a message defined to the message repository (that is, an MRM message set), you have to assign the message set to the broker. If you are using an XML message, you do not do this.

In the Assignments view, check out the brokers to which you want to deploy this scenario. Drag and drop the message set (My Receipt Messages) onto the name of the broker in the graphic in the Domain Topology pane. When you have finished these updates, check in the brokers. The message set information is sent to the broker in the form of a message dictionary when you deploy your changes. See “Deploying the configuration” for information about deploying.

Assigning message flows to the execution group

Message flows must be assigned to an execution group in a broker before they can process messages.

In the Assignments view, check out an execution group in the broker to which you want to assign the message flows associated with this scenario (for example, the default execution group). Drag and drop the appropriate main flow and its register subscription flow (**MRMscenario** and **Register subscription – MRM** or **xmlscenario** and **Register subscription – XML**) onto the graphic of the execution group in the Domain Topology pane. The loop counter subflow is assigned automatically, because it is part of the main flow. When you have finished these updates, check in the execution group.

Deploying the configuration

Finally, to use the configuration set up, you must deploy it. To do this, select **File —> Deploy —> Complete configuration (all types) —> Normal**. For more deployment options, see the *WebSphere MQ Integrator Using the Control Center* book. You can check whether deployment was successful by refreshing and viewing the messages that are displayed in the Log view.

When the configuration has been successfully deployed, you can put receipt messages and subscription messages to the appropriate queues and see the messages processed through the message flows. Further details of how you can test out your configuration are provided in “Testing the message flow” on page 39.

Testing the message flow

The SupportPac includes two test messages that you can use to test out your deployed message flows.

If you have imported, or have created, the message flows for the MRM message scenario, you can test the message flow by putting the example message, in file `mrmreceipt.xml`, to the input queue (IN) of the message flow. If you are using an XML message, put the message in file `subscribemsgMRM.xml` to the same queue. You must remember to specify the correct queue name if you have chosen not to use the default queue name.

One way of putting a message to the input queue of the message flow is to use the utility program, `mqsiput`, that is provided in SupportPac IH02:

```
mqsiput IN MQSI_SAMPLE_QM mrmreceipt.xml
```

When processing has completed, you can find the processed message on queues FINANCE, STOCK, and PARTNERS.

You can register a subscription on messages containing an itemname of ″Shampoo″ by putting the message in file `subscribemsg.xml`, which is valid for both MRM and XML scenarios, to the SUBIN queue for your Register subscriptions flow. If you do this, you receive a confirmation message on queue SUBOUT, and a ’multibuy’ entry that indicates a successful subscription appears in the Subscription pane of the Control Center window.

If you then put the `mrmreceipt.xml` message on the input queue (IN) of the message flow, a confirmation message is received on queue SUBOUT, and the processed message appears on queue SUBS.

You can edit the input message to change the path that it takes through the message flow. For example, you can change the multibuy value to ″no″. After processing, the message appears on queues FINANCE, STOCK, and NOTMULTI.

If you edit the input message to remove all instances of the item ″Shampoo″, the message is processed by the message flow, and is published, but because it does not match the subscription filter, no message is put to queue SUBS.

Chapter 2. The loan request

This chapter describes a business scenario that implements a loan request.

Highlights of this example include:
- Importing a ’legacy’ message format
- Populating a message with information retrieved from a database

The message flow examples use the following IBM Primitive nodes:
- MQInput
- Database
- Compute
- Filter
- DataUpdate
- MQOutput

Introduction

This example uses a single message flow, illustrated in Figure 20. The message flow processes a message of a specific format and content.

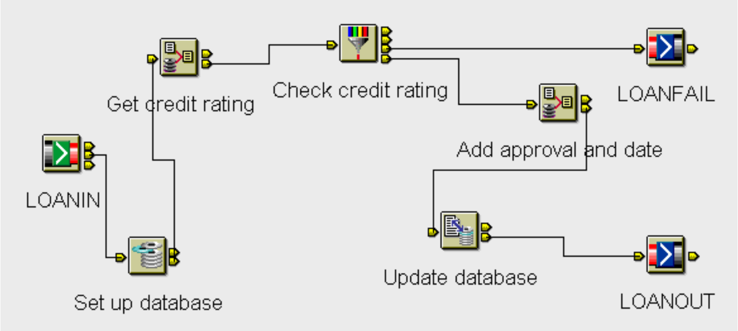


Figure 20. Loan request message flow

The definition of the message is derived from a COBOL copybook that is imported into the Control Center and created as an MRM message. This is described in “Creating and completing the message set” on page 43.

“Creating and configuring the message flow” on page 44 describes how to define and configure the message flow.

Before you create your message and message flow, perform the tasks described in “Preparing to implement the scenario” on page 42. You must perform these tasks before you can successfully deploy your message flow.

Preparing to implement the scenario

Before you define your message set and message flows you must:

1. Initialize the database
2. Set up some MQSeries resources; for example, some MQSeries queues

Initializing the database

The message flow used in this example uses a database for the storage of information that passes through the message flow. Before you can configure and use the message flow, you must create a database called MYDB, and an ODBC connection to that database. See “Creating the database” on page xiv for further information about how to do this.

You must also set up the tables `CREDIT` and `CREDITREF` that are used by the message flow. You can use the file `loan.sql`, which is supplied in this SupportPac, to create these tables and to insert data into them. If you use the supplied script, review the guidelines given in “Using the supplied DB2 scripts” on page xv for accessing the database and creating and using schema names.

To use `loan.sql`, open a DB2 Command Window on the system on which your broker is running, and type:

```
db2 connect to MYDB user BKUSER1 using BKPW1
 (where BKUSER1 and BKPW1 are the values specified for the -u and -p flags
   on the mqsicreatebroker command)
db2 -f loan.sql
```

If the file `loan.sql` is not in your current directory, you must type the complete path for the file.

You might see the error message DB21034E when you run this command file for the first time. This is because the file first drops any existing tables, and then creates the tables it needs. If the tables don’t already exist, error messages are generated. You can ignore these messages.

You must also ensure that each broker to which you deploy this message flow has access to this database.

You can create these tables yourself if you choose, but you must then also provide sample table entries before you can use the message flow. You can review the SQL statements contained in file `loan.sql` to see details of the entries created in the tables.

Setting up MQSeries resources

This example gets an input message for the message flow from an MQSeries queue on the queue manager hosting the broker to which the message flow has been deployed. It puts messages on to one of two output queues, depending on the success or failure of the action taken by the message flow on each message received.

The queues required are:

- LOANIN, the input queue for the message flow
- LOANOUT, the output queue for the message flow
- LOANFAIL, the queue to which request messages are put if the loan request fails

You can use the file `loan.tst` to create these queues, or you can create the queues yourself. To use `loan.tst`, type the following on a command line on the system on which your broker is running:

```
runmqsc MQSI_SAMPLE_QM <loan.tst
```

If the file `loan.tst` is not in your current directory, you must type the complete path name for the file. If you are not using the queue manager `MQSI_SAMPLE_QM`, type the name of your queue manager instead of `MQSI_SAMPLE_QM`.

You can use different names for the queues, but, if you do, you must edit the file `loan.tst` before you use the `runmqsc` command, and you must change the queue properties of the nodes in the message flow to match your new names. See "Creating and configuring the message flow" on page 44 for more details.

Creating and completing the message set

You must import the COBOL copybook into the Control Center to create a message that can be processed by the message flow. Start the Control Center and ensure that your user role is set to *All roles* (use **Preferences** in the **File** menu to view and change the user role).

- Create a new message set in the Control Center (Message Sets view). Specify the name **RequestLoan**, and leave all other values to assume default values.
- Right click on your new message set and select **Import to Message Set** then **COBOL...**.
- Either type the full path of `credit.cpy` in the Cobol Importer dialog, or click **Browse** to locate the file.
- Click **Finish**. The Control Center imports the copybook and creates a set of elements that correspond to the contents of the copybook. It displays an information dialog when it has completed the import.
- Add the following new elements to the Elements folder in your new message set.

  To do this, right click on the Elements folder, select **Add to Workspace**, and select the elements listed:
  - APPROVAL
  - CHECK_1
  - COMMENTS
  - CUST_ACCT_NUM
  - CUST_BANK_NAME
  - DETAILS
  - FIRST_NAME
  - INTEREST_RATE
  - LAST_NAME
  - LOAN_AMOUNT
  - STARTDATE
  - TERM
- Add the following new compound types to the Types folder in your new message set.

  To do this, right click on the Types folder, select **Add to Workspace**, and select the types listed:
  - DETAILS_TYPE
  - FX_LOAN_REQUEST_TYPE
- Add the following new element values to the Element Values folder in your new message set.

The loan request

To do this, right click on the Element Values folder, select **Add to Workspace**, and select the element values listed:

- – APPROVAL_VALUE
- – COMMENTS_VALUE
- – CUST_BANK_NAME_VALUE
- – FIRST_NAME_VALUE
- – LAST_NAME_VALUE
- – STARTDATE_VALUE

- Now create the message:
  1. Right click on the Messages folder within your message set.
  2. Select **Create** then **Message**. The dialog ***Create a new message*** is displayed.
  3. On the Message tab, type the name of the message, FX_LOAN_REQUEST. Type an identifier of `loanrequestmessage`, and select a **Type** of FX_LOAN_REQUEST_TYPE. Let all other fields take their default value.
  4. Click **Finish**. The message is created with the correct contents in the correct order.
  5. Check in all your message resources.

     To do this, select **File** then **Check In** then **List...**. Expand the Message Sets tree and select all the resources shown there. Click **Check In**.

Creating and configuring the message flow

The file `Loan Request.xml` contains the definition of the loan request message flow. To import and configure this message flow:

- Select the Message Flows tab.
- Import the loan request message flow definition:
  1. From the **File** menu, select **Import to Workspace**.
  2. Locate the file `Loan Request.xml` that contains the message flow definition. Either type in the full path and file name in the dialog, or click **Browse** and search for the file.

     The **Message Flows** check box is checked by default. This imports message flows only from the file that you have identified as the source of the resources to be imported.
  3. Click **Import**. The Control Center processes the file and imports the message flows that are defined within it. The file `Loan Request.xml` contains just one message flow, **Loan request**. A list of resources that have been imported are displayed in a dialog when the processing has completed. Click **OK** to dismiss the dialog.
- The new message flow appears in the left-hand pane, with a blue cube beside it to show that it is new. Select this new message flow. You can now configure the nodes within it before checking it in to the configuration repository.
- Select **File—>Check In—>All (Save to Shared)** to check in the message flow.

If you do not use the import file provided, you must create the message flow yourself, and complete its design by dragging and dropping the appropriate nodes into the Message Flow Definition pane. You must connect the nodes and configure them as described in the following sections.

Getting the message

The MQInput node *LOANIN* gets the loan request input messages from the input queue LOANIN.

If you have imported the message flow, this node is partially configured. To complete it, you must:

1. Select the Default tab, and set the message domain to MRM and the message set to the identifier of your new message set; you can select this from the drop-down list. This step is necessary because the identifier is created when you create the message set, and cannot be correctly specified in the message flow for import.
2. If you are not using the default queue name, you must type the correct name on the Basic tab.
3. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. On the Basic tab, type the name of the input queue. The default is LOANIN, but you can use another value if you choose.
2. Select the Default tab, and set the message domain to MRM, the message set to the identifier of your new message set (you can select this from the drop-down list), the message type to the identifier of your new message, loanrequestmessage, and the message format to CWF.
3. Click **OK** to finish.
4. Connect the node's out terminal to the DataInsert node *Set up database*.

Storing message contents in the database

The DataInsert node *Set up database* stores into the database values from within the message into the database.

If you have imported the message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** to remove the database table from the Output pane.
2. Click **Add** in the Output pane and add the data source MYDB and the correct table schema for the CREDIT table.
3. Select **Add column** to add each column in the CREDIT table (LastName, FirstName, BankName, AcctNum, Check1, Approval, StartDate, Term, Rate, Amount, Comments).
4. Map the message elements to the database table columns by dragging the input message elements onto the corresponding column in the output database table.
5. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click **Add** above the Input pane and select your new message set and new message from the drop-down lists on the Add dialog.
2. Click **Add** above the Output pane and type datasource (MYDB) and the correct table schema for the CREDIT table (for example, BKUSER1.CREDIT) to add the database table.
3. Right-click in the Output pane and select **Add column** to add each column in the CREDIT table (LastName, FirstName, BankName, AcctNum, Check1, Approval, StartDate, Term, Rate, Amount, Comments).

4. Map the message elements to the database table columns by dragging the input message elements onto the corresponding column in the output database table.
5. Click **OK** to finish.
6. Connect the node's out terminal to the Compute node *Get credit rating*.

Getting the credit rating

The Compute node *Get credit rating* extracts the credit rating information for the requestor from the database and adds it to the message.

If you have imported the message flow, this node is partially configured. To complete it you must:

1. Click **Delete** to remove the CREDITREF database table from the Inputs pane. Click **Add** and add datasource MYDB and the correct table schema for the CREDIT table to the Inputs pane.
2. Select the ESQL tab and edit the ESQL to specify the correct schema name. For example:
   ```
   SET "OutputRoot"."MRM"."CHECK_1"=THE(SELECT ITEM T.SCORE FROM
       Database.BKUSER1.CREDITREF AS T WHERE T.LastName="InputBody"."LastName");
   ```
3. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click **Add** and add the loan request message FX_LOAN_REQUEST to the Inputs pane.
2. Click **Add** and add datasource MYDB and the correct table schema for the CREDIT table to the Inputs pane.
3. Click **Add** and add the loan request message to the Output Messages pane.
4. Select the **Copy entire message** radio button.
5. On the lower part of this tab, select the ESQL tab and type the following statement, substituting the correct database schema name for BKUSER1:
   ```
   SET "OutputRoot"."MRM"."CHECK_1"=THE(SELECT ITEM T.SCORE FROM
       Database.BKUSER1.CREDITREF AS T WHERE T.LastName="InputBody"."LastName");
   ```
6. Click **OK** to finish.
7. Connect the node's out terminal to the Filter node *Check credit rating*.

Checking the credit rating

The Filter node *Check credit rating* uses the credit rating information with the details of the request in the message to check whether the request can be honored. The node checks that the credit rating is greater than 70.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click **Add** to add the input message to the Inputs pane. (You do not have to do this, but it does ensure that the message used for this node is documented here.)
2. Type the following ESQL on the Check credit rating tab:
   ```
   "Body"."CHECK_1">70
   ```
3. Click **OK** to finish.
4. Connect the node's false terminal to the MQOutput node *LOANFAIL*.
5. Connect the node's true terminal to the Compute node *Add approval and date*.

Adding the date

The Compute node *Add approval and date* modifies the message by adding the start date for the loan, and marking it as approved. The current message is only passed to this node if the contents of the message indicate that the credit rating of the request is acceptable.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Click **Add** in both Inputs and the Output Messages panes to add the loan request message.
2. Select **Copy entire message**.
3. Select the ESQL tab in the lower part of the pane, and type the following statements:

   ```
   SET "OutputRoot"."MRM"."APPROVAL"='YES';
   SET "OutputRoot"."MRM"."STARTDATE"=CAST((CURRENT_DATE + INTERVAL
    '1' MONTH)AS CHAR);
   ```

   The second statement casts the CURRENT_DATE as a character string to convert the date value to a format that can be stored in a column in the database table.
4. Click **OK** to finish.
5. Connect the node’s out terminal to the DataUpdate node *Update database*.

Updating the database

The DataUpdate node *Update database* retrieves information about the approved loan from the message, and updates the loan database.

If you have imported the message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** to delete the database from the Output pane.
2. Click **Add** and add datasource MYDB and the correct table schema for the CREDIT table to the Output pane.
3. Add the columns CHECK1, APPROVAL, and STARTDATE.
4. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click **Add** and add the loan request message on the Input pane.
2. Click **Add** and add datasource MYDB and the correct table schema for the CREDIT table to the Output pane.
3. Add the columns CHECK1, APPROVAL, and STARTDATE.
4. Drag and drop the fields from the input message to the equivalent columns in the database. The following mappings are generated in the Update Mappings tab.

   ```
   "Body"."CHECK_1"—>CHECK1;
   "Body"."APPROVAL"—>APPROVAL;
   "Body"."STARTDATE"—>STARTDATE;
   ```
5. Click **OK** to finish.
6. Connect the node’s out terminal to the MQOutput node *LOANOUT*.

Outputting the message

The MQOutput node *LOANOUT* places the message on the output queue from which another application can retrieve it and take any further action.

If you have imported the message flow, this node is fully configured and identifies queue LOANOUT, on broker MQSI_SAMPLE_BROKER, as the output queue. If you are using a different queue or queue manager name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. Select the Basic tab. Type the name of the queue manager and the queue that you are using for output. You can let all other properties take their default value.
2. Click **OK** to finish.

Handling requests that are not approved

If the loan request is not approved, the check in the Filter node *Check credit rating* fails and the message is routed to the MQOutput node *LOANFAIL*.

If you have imported the message flow, this node is fully configured and identifies output queue LOANFAIL on queue manager MQSI_SAMPLE_QM. If you are using a different queue name, you must update the node properties.

If you are creating the message flow yourself, you must:

1. Select the Basic tab. Type the name of the queue manager and the queue that you are using for failure output. You can let all other properties take their default value.
2. Click **OK** to finish.

Assigning and deploying the scenario

When your message set and message flow definitions are complete, you must check in all the objects that you have been working with before you can assign them to a broker. To do this, select **File—>Check In—>All (Save to Shared)** to check in all objects at once. For other check in options, see the *WebSphere MQ Integrator Using the Control Center* book.

When you have completed message flow configuration, you must assign and deploy the resources so that you can use the message flow. Select the Assignments view in the Control Center and check out the brokers to which you want to deploy this scenario. Drag and drop the message set that contains the FX_LOAN_REQUEST message from the Assignable Resource pane to the brokers pictured in the Domain Topology pane. When you have finished these updates, check in the brokers.

Check out an execution group on each broker to which you are deploying the scenario. Drag and drop the message flow **Loan request** from the Assignable Resource pane to an execution group within each broker. When you have done this, check in the execution groups.

When you have completed your assignments, you can deploy your changes. You can do this in one of the following two ways:

1. Deploy the delta or complete configuration data (all types) from the **File** menu.
2. Deploy only Assignments configuration data by right clicking on the broker to which you want to deploy and selecting **Deploy**.

You can check the success of the deployment by going to the Log view, and refreshing the contents of the log.

For more details about assignment and deployment, refer to the *WebSphere MQ Integrator Using the Control Center* book.

Testing the message flow

The SupportPac includes a test message that you can use to check out your message flow. It is defined in file `loanreq.xml`. The contents of the message assume that you have used `loan.sql` to set up and populate the database. If you have not initialized the database tables using `loan.sql`, the credit check fails and the message is output to the LOANFAIL queue.

You can edit the message in file `loanreq.xml` to replace the existing name (einstein). If you do, you must remember to keep the full length of the name field by inserting the correct number of blanks in this field.

Chapter 3. Dynamic routing

This scenario illustrates how you can use the RouteToLabel node to determine processing of the message based on content of the message itself.

Highlights of this scenario include:

- Dynamically routing messages based on their content
- Using data retrieved from a database with data within a message to calculate values that determine the path of a message through the message flow

The example message flows use the following IBM Primitive nodes:

- MQInput
- Compute
- RouteToLabel
- Label
- DataUpdate
- MQOutput

Introduction

This scenario shows how you might handle brokerage transactions if you want to process trading requests and requests to update customer details differently. It uses a message flow, illustrated in Figure 21, that supports dynamic routing of messages according to their content.

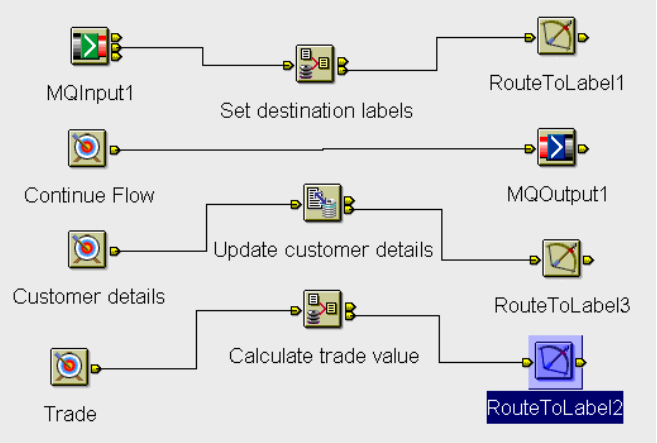


Figure 21. The dynamic routing message flow.

The flow is made up of subflows that are associated with the main flow using RouteToLabel and Label nodes.

- The input flow. This flow, described in “The input flow” on page 55, receives each message from an input queue, constructs a destination list for the message, and passes the message to a RouteToLabel node that decides on further processing.
- The customer details flow. This flow, described in “The customer details flow” on page 56, handles incoming messages that have a value of ″custdetails″ in the

request field of the message. It performs processing specific to requests to update customer details, for example, updating a customer details database. It ends in a RouteToLabel node to send the message on to the next destination in the list.

- The trade flow. This flow, described in “The trade flow” on page 58, handles incoming messages that have a value of ″lowtrade″ in the request field of the message and performs processing specific to low-value trading requests. It ends in a RouteToLabel node to send the message on to the next destination in the list.
- The completion flow. This flow, described in “The completion flow” on page 59, receives messages that have been processed and are now ready to continue through the message flow. In this example, the message is put directly to an MQOutput node, but it is likely that you might create a message flow that has a sequence of nodes that process both types of message before reaching the output node.

The use of RouteToLabel and Label nodes makes a simpler message flow than the message flow you would need if you used a sequence of Filter nodes that identify and route the message for different processing, or a sequence of nodes that each performs an action on a subset of the total number of messages processed by the message flow.

The message flow provided is sufficient to illustrate the concept of dynamic routing using the RouteToLabel and Label nodes. Your requirements are probably for a more complex flow, but you can build on the techniques adopted in this scenario to create more comprehensive message flows that handle a wider range of options in message processing, For example, a likely extension to this supplied message flow is to add subflows that handle a larger number of request types.

The message flow processes messages of a specific format and content. The definition of the messages is described in “Defining the message” on page 53.

“Creating and configuring the message flow” on page 54 describes how to define the message flow.

Before you create your message and message flow, perform the tasks described in “Preparing to implement the scenario”, which are required before you can successfully deploy your message flow.

Preparing to implement the scenario

You must perform the tasks described in the following sections before you can successfully deploy your message flow. Perform these tasks before you start to define your message set and message flows.

Initializing the database

The message flow in this scenario uses a database for the storage of information passing through the message flow. Before you can configure and use the message flow, you must create a database called MYDB, and an ODBC connection to that database. See “Creating the database” on page xiv for further information.

You must also set up the tables `STOCKPRICE` and `CUSTOMER` that are used by the message flow. You can use the file `route.sql` to create these tables if you choose, or you can create them yourself. If you use the supplied script, you must review the guidelines given in “Using the supplied DB2 scripts” on page xv for accessing the database and creating and using schema names.

To use `route.sql`, open a DB2 Command Window on the system on which your broker is running, and type:

```
db2 connect to MYDB user BKUSER1 using BKPW1
 (where BKUSER1 and BKPW1 are the values specified for the -u and -p flags
   on the mqsicreatebroker command)
db2 -f route.sql
```

You must enter the complete path for the file `route.sql` if it is not in your current directory.

You might see error DB21034E the first time you run this command file. This is because the file first drops any existing tables, and then creates them. If the tables don’t already exist, error messages are generated. You can ignore these messages.

You must also ensure that each broker to which you deploy this message flow has access to this database.

Setting up MQSeries resources

This scenario gets an input message for the message flow from an MQSeries queue on the queue manager hosting the broker to which the message flow has been deployed. It puts messages on to an output queue when message processing has been completed.

The queues required are:
- LABELIN
- LABELOUT

You can use the file `route.tst` to create these queues if you choose, or you can create them yourself. To use `route.tst`, type the following on a command line on the system on which your broker is running:

```
runmqsc MQSI_SAMPLE_QM <route.tst
```

You must enter the complete path for the file `route.tst` if it is not in your current directory. Replace the queue manager name in the command above if the name shown is not correct for your broker.

You can use different names for these queues if you choose. If you do, you must edit `route.tst` before you use the `runmqsc` command, and you must modify the queue properties of the nodes in the message flow to match your new names. See “Creating and configuring the message flow” on page 54 for more details.

Defining the message

The message used in this brokerage example is an MRM message (that is, it is defined to the message repository) with a message format of XML. If you prefer, you could use other formats (for example, a self-defining XML message).

The message has a request field that indicates whether the message contains ″lowtrade″ or ″custdetails″ information. Each type is routed to a different sequence of nodes before being completed by a common flow.

To use the predefined message structure, you must import the definition that is provided in the file `RouteToLabelMsgSet.mrp`. To import this file, you must:

Dynamic routing

1. Open a command prompt window.
2. Stop the Configuration Manager using the **mqsistop configmgr** command.
3. Change to the directory that contains the file `RouteToLabelMsgSet.mrp`.
4. Import the message set by typing:

   ```
   mqsiimpexpmsgset -i -u mqsiuid -p mqsipw -n MQSIMRDB -f RouteToLabel.mrp
   ```

   where `–u` and `–p` are the userID and password that are used for message repository access, and `–n` specifies the name of the message repository (you specified these parameters using the `–n`, `–u`, and `–p` flags on the **mqsicreateconfigmgr** command). The installation guide for your broker operating system gives more details about the use of this command.
5. Restart the Configuration Manager using the command **mqsistart configmgr**. This picks up the new definition
6. Restart the Control Center from the Windows NT *Start* menu.
7. In the Control Center Message Sets view, right-click on the Message Sets root and select **Add to Workspace**.
8. Select the message set `RouteToLabelMsgSet`, and click **Finish**.

You can now view the message set in the left-hand pane of the Message Sets view. The components of the message set are all present in the message repository, but are not added to your workspace. If you want to view some or all of these components, you can add them by right-clicking on the component folder (for example, for Messages) and selecting **Add to Workspace**.

The `RouteToLabelMsgSet` message set contains the following messages:
- `Trade message`. This contains trade and customer details.

If you prefer to create the message yourself, instructions are provided in “Creating the message set by hand” on page 61.

Creating and configuring the message flow

The file `DynamicRouting.xml` contains the definition of the dynamic routing message flow. You can import and configure this message flow as follows:
- Select the Message Flows tab.
- Import the dynamic routing message flow definition:
  1. Select **Import to Workspace...** from the **File** menu.
  2. Locate file `DynamicRouting.xml` that contains the message flow definition (you can type in the full path and file name in the dialog, or you can click **Browse** and search for the file).

     The **Message Flows** check box is checked by default. This imports message flows only from the file that you have identified as the source of the resources to import.
  3. Click **Import**. The Control Center processes the file and imports the message flows that are defined within it. The file `DynamicRouting.xml` contains just one message flow, **DynamicRouting**. A list of resources that have been imported are displayed in a dialog when the processing has completed.
  4. Click **OK** to dismiss the dialog.
- The new message flow appears in the left hand pane, with a blue cube beside it to show that it is new. Select this new message flow: you can now configure the nodes within it before checking it in to the configuration repository. The majority of the configuration work is already done in the imported flow.

- Select **File—>Check In—>All (Save to Shared)** to check in the message flows.

If you choose not to use the import file provided, you must create the message flow and complete its design by dragging and dropping the appropriate nodes into the Message Flow Definition pane. You must connect the nodes and configure them as described in the following sections.

You must ensure that you include all related Label nodes and the subflows that follow from the Label nodes within the Message Flow Definition pane of the message flow or subflow that contains the RouteToLabel node or nodes that route messages to these Label nodes. This ensures that all the Label nodes are included when you deploy the message flow.

The Label node is not connected to a prior node: if you create a subflow that starts with any other type of node, the subflow defined in the Message Flow definition pane is ignored when the message flow is deployed. Subflows that start with a Label node are not ignored.

For more information about destination lists, see the *WebSphere MQ Integrator ESQL Reference* book. If you intend to derive destination values from the message itself, or from a database, you might also need to cast values from one type to another. Casts are described in more detail in the *WebSphere MQ Integrator ESQL Reference* book.

The input flow

This receives each message from an input queue, constructs a destination list for the message, and passes the message to a RouteToLabel node that decides on further processing.

Getting the message

The MQInput node *MQInput1* retrieves input messages from the input queue.

If you have imported the message flow, this node is fully configured and identifies input queue LABELIN. If you are using a different queue name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. On the Basic tab, enter the input queue name. The default is LABELIN.
2. On the Default tab, enter the message domain (MRM), message set identifier, message type (trademessage), and message format (XML).
3. Click **OK** to finish.
4. Connect the out terminal to the node *Set destination labels*.

Setting the destination labels

The Compute node *Set destination labels* sets up the DestinationList for the message received.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Create a destination list in the message.

   The destinations are set up as a list of label names. The label names can be any string value, and can be explicitly specified in the compute node or taken or cast from any field in the message or from a database. A label name in the destination list must, however, match the *Label Name* property of a corresponding Label Node.

On the lower part of the properties dialog, select the ESQL tab and enter the following below the comment line:

```
SET "OutputDestinationList"."Destination"."RouterList"."DestinationData"[1].
"labelname" = 'continue';
SET "OutputDestinationList"."Destination"."RouterList"."DestinationData"[2].
"labelname" = "InputRoot"."MRM"."request";
```

2. Select the *Advanced* tab on the Compute node properties dialog, and select the option *LocalEnvironment and Message* from the drop-down list.
3. Click *Copy entire message* radio button.
4. Click **OK** to finish.
5. Connect the out terminal to the node *RouteToLabel1*.

Configuring the RouteToLabel node

The RouteToLabel node *RouteToLabel1* uses the destination list within a message (created in the *Set destination labels* node) to route the message to a target node of type Label that matches the label within the destination list item.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set Mode to Route to Last.
2. Click **OK** to finish.

When Mode is set to Route to Last in the RouteToLabel node properties, a message is routed to the last label in the destination list. The destination list created by the node *Set destination labels* is created assuming the mode is set this way.

The RouteToLabel node matches the value of the ″request″ field in the message. A message with a value of ″lowtrade″ in the request field is routed to the Label node with a Label Name property of ″lowtrade″. A message with a value of ″custdetails″ in the request field is routed to the Label node with the Label Name property of ″custdetails″.

If a message fragment performing the dynamically routed work itself ends in a RouteToLabel node, the message is passed to the next destination in the list. In this example, the message is passed to the Label node with a Label Name property of ″continue″, and continues along the common part of the message flow.

Setting up the labelled message flow routes

There are three routes, each beginning with a Label node. The *Label Name* property on each Label node must match either the *labelname* specified in the Compute node that set the destination labels, or the value of a field in an incoming message. These routes do not have connections between them and the RouteToLabel nodes: this looks unusual, and is the only situation in which an unconnected node is processed in a message flow. (If you leave other nodes unconnected, they are ignored.)

The customer details flow

This flow starts with a Label node *Customer details* that has the Label Name property set to *custdetails*. It handles incoming messages that have a value of ″custdetails″ in the request field of the message, and performs processing specific to requests to update customer details in a customer details database.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set the Label Name property to *custdetails*.
2. Click **OK** to finish.
3. Connect the out terminal to the node *Update customer details*.

Updating the customer details

The DataUpdate node *Update customer details* interacts with the database to make the changes contained within the message. A DataUpdate node is selected because the input message is an MRM message: using the DataUpdate node allows you to drag and drop message elements onto the database columns.

(You can use a Database node if you prefer, but if you do you must construct the ESQL yourself. If you are using an XML message, you must use a Database node and construct the ESQL yourself, because the elements are not defined and therefore cannot be dragged.)

If you have imported the message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** and remove the datasource from the Output pane.
2. Click **Add** to add datasource MYDB and the correct table schema for the CUSTOMER table (for example, BKUSER1.CUSTOMER) to the Output pane.
3. Add the columns CUSTACCTNUM, BANKACCTNUM, HOUSENUM, STREET, TOWN, and POSTCODE to the Output pane.
4. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. On the Input pane, add the message `Trade message`.
2. On the Output pane, add the table CUSTOMER (in datasource MYDB), and add the columns CUSTACCTNUM, BANKACCTNUM, HOUSENUM, STREET, TOWN, and POSTCODE.
3. Drag the message element custacctnum from the Input pane and drop it onto the name CUSTACCTNUM in the Output pane. This element is used as the key into the database information.
4. Select the Update Mappings tab. Drag the message elements housenum, street, town, and postcode into the Update Mappings pane. These elements are used to update the details in the database.
5. Click **OK** to finish.
6. Connect the out terminal to the node *RouteToLabel3*.

Completing message processing

This message flow fragment itself ends in a RouteToLabel node, *RouteToLabel3*. The message is passed to the next destination in the list. In this scenario, the message is passed to the Label node with a Label Name property of ″continue″, and continues along the common part of the message flow.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set Mode to Route to Last.
2. Click **OK** to finish.

The trade flow

This flow starts with a Label node *Trade* that has the Label Name property set to *lowtrade*. It handles incoming messages that have a value of ″lowtrade″ in the request field of the message, and performs processing specific to low-value trading requests.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set the Label Name property to *lowtrade*.
2. Click **OK** to finish.
3. Connect the out terminal to the node *Calculate trade value*.

Calculating the trade value

The Compute node *Calculate trade value* retrieves the stock price from a DB2 database, and calculates the value of the trade request based on that price and the quantity of stock requested. It also updates the message with the calculated value.

If you have imported the message flow, this node is partially configured. To complete it, you must:

1. Click **Delete** to remove the database table STOCKPRICE from the Inputs pane.
2. Click **Add** to add datasource MYDB and the correct table schema for the STOCKPRICE table (for example, BKUSER1.STOCKPRICE) to the Inputs pane.
3. Add the columns STOCK and PRICE.
4. In the lower part of the properties dialog, select the ESQL tab and edit the following statement to insert the correct schema name. For example:

   ```
   SET "OutputRoot"."MRM"."stockprice"=THE(SELECT ITEM T.PRICE FROM
        Database.BKUSER1.STOCKPRICE AS T WHERE T.STOCK="InputRoot"."MRM"."stock");
   SET "OutputRoot"."MRM"."tradevalue"= ("OutputRoot"."MRM"."stockprice")*
          ("InputRoot"."MRM"."stockquantity");
   ```
5. Click **OK** to apply your changes.

If you are creating the message flow yourself, you must:

1. Click **Add** to add the message Trade message to the Inputs pane.
2. Click **Add** to add datasource MYDB and the correct table schema for the STOCKPRICE table (for example, BKUSER1.STOCKPRICE) to the Inputs pane. Add the columns STOCK and PRICE.
3. Click **Add** to add the message Trade message to the Output Messages pane.
4. Select **Copy entire message**.
5. In the lower part of the properties dialog, select the ESQL tab and enter the following below the comment line (substituting the correct schema name for your database):

   ```
   SET "OutputRoot"."MRM"."stockprice"=THE(SELECT ITEM T.PRICE FROM
        Database.BKUSER1.STOCKPRICE AS T WHERE T.STOCK="InputRoot"."MRM"."stock");
   SET "OutputRoot"."MRM"."tradevalue"= ("OutputRoot"."MRM"."stockprice")*
          ("InputRoot"."MRM"."stockquantity");
   ```
6. Click **OK** to finish.
7. Connect the out terminal to the node *RouteToLabel2*.

Completing message processing

This message flow fragment itself ends in a RouteToLabel node, *RouteToLabel2*. The message is passed to the next destination in the list. In the example above, the message is passed to the Label node with a Label Name property of ″continue″, and continues along the common part of the message flow.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set Mode to Route to Last.
2. Click **OK** to finish.

The completion flow

This flow starting with a Label node *Continue Flow* which has the Label Name property set to *continue*. It receives messages that have been processed and are now ready to continue through the message flow. In this example, the message is put directly to an output node, but it is likely that you might create a message flow that has a sequence of nodes that process both types of message before reaching the output node.

Typically, the Label Name of this fragment is the one at index [1] of the destination list. This means that, when all the dynamically routed work is complete, the flow either finishes or continues with common processing.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Set the Label Name property to *continue*.
2. Click **OK** to finish.
3. Connect the out terminal to the node *MQOutput1*.

Outputting the message

The MQOutput node *MQOutput1* places the message on the output queue from which another application can retrieve it and take any further action.

If you have imported the message flow, this node is fully configured and identifies queue LABELOUT on queue manager MQSI_SAMPLE_QM. If you are using a different queue, you must enter the correct identifiers on the Basic tab.

If you are creating the message flow yourself, you must:

1. Select the Basic tab. Enter the name of the queue manager and the queue that you are using for output. You can let all other properties take their default value.
2. Click **OK** to finish.

Assigning and deploying the scenario

When your message set and message flow definitions are complete, you must check in all the objects that you have been working with before you can assign them to a broker. To do this, select **File—>Check In—>All (Save to Shared)** to check in all objects at once. (For other check in options, see the *WebSphere MQ Integrator Using the Control Center* book.)

You must now assign and deploy the resources so that you can use the message flow. Select the Assignments view in the Control Center and check out the brokers to which you want to deploy this scenario. Drag and drop the RouteToLabel message set that contains the trade message from the Assignable Resources pane to the brokers pictured in the Domain Topology pane. When you have finished these updates, check in the brokers.

Check out an execution group on each broker to which you are deploying the scenario. Drag and drop the message flow **RouteToLabel** from the Assignable Resource pane to an execution group within each broker. When you have done this, check in the execution groups.

When you have completed your assignments, you can deploy your changes. You can do this in one of the following two ways:

1. Deploy the delta or complete configuration data (all types) from the **File** menu.
2. Deploy only Assignments configuration data by right clicking on the broker to which you want to deploy and selecting **Deploy**.

You can check the success of the deployment by going to the Log view, and refreshing the contents of the log.

For more details about assignment and deployment, refer to the *WebSphere MQ Integrator Using the Control Center* book.

Testing the message flow

The SupportPac includes a test message that you can use to test out your deployed message flow.

You can test this message flow by putting the XML message `trademsg.xml` to the input queue for this message flow on a broker to which it has been deployed.

You can edit the XML message file (for example, using Notepad) to change the value of the request field from ″lowtrade″ to ″custdetails″. When you put this updated message to the input queue of the message flow, the message passes through the Customer Details route.

Creating the message set by hand

You can create the MRM message set by hand if you prefer, by following the following guidance:

- Create the message set `RouteToLabel`.
- Create the elements:

Table 7. Routing simple elements, types, names, and identifiers

Simple element name	Identifier	Type
request	request	STRING
stock	stock	STRING
Stock quantity	stockquantity	INTEGER
Stock Price	stockprice	FLOAT
Trade value	tradevalue	FLOAT
Customer Account Number	custacctnum	INTEGER
Bank Account Number	bankacctnum	INTEGER
House Number	housenum	INTEGER
Street	street	STRING
Town	town	STRING
Postcode	postcode	STRING

- Create lengths for the string elements:

Table 8. Routing STRING elements, lengths, names, and identifiers

Element name	Element length name	Maximum Length	Element length identifier
request	reqlen	12	reqlen
stock	stocklen	40	stocklen
Street	Street Length	20	streetlen
Town	Town Length	20	townlen
Postcode	Postcode Length	10	postcodelen

  Add the lengths to the corresponding string elements. For example, add `stocklen` to the element `stock`.
- Create the compound types:

Table 9. Routing compound type names and identifiers

Compound type name	Identifier
Address	Address
Trade message	trademsg

- Add elements to the compound types. The order of elements in the message being passed through the message flow **must** match the order of elements in the message definition. This order is defined by the order of elements in the compound types.

  When you add elements to a compound type, the most recent is added at the top, so you must add these in reverse order. Table 10 on page 62 lists the elements to add in the order in which you must add them. (There is a Reorder option on the Types pull-down to re-sequence the elements within a type.)

Table 10. Routing elements to add to compound types

Compound type	Elements to be added
Address	– Postcode – Town – Street – House Number
Trade message	– Address – Bank Account Number – Customer Acct Number – Trade value – Stock Price – Stock quantity – stock – request

- Create the message `Trade message`, with identifier `trademessage`, of type `Trade message`. This definition matches the XML message supplied in `trademsg.xml`. The message in this file has an MQRFH2 header in it already: if you create your own message definition, you must make sure that the message set identifier, message identifier, and element identifier in the message input file match your MRM definitions.
- Save the definitions to the shared repository. Select **File —> Check In —> All (Save to Shared)**.

Chapter 4. Travel agent (single-flow) example

This example is the first of three examples which illustrate the concept of message aggregation.

Message aggregation extends the request/reply metaphor by enabling a single message, or request, to be sent to more than one node, receiving back replies from some, or all, of these nodes, and then combining these replies into a single reply message.

To use message aggregation, you must construct two message flows; the ’fan-out’ flow sends the input message to the nodes from which a reply is required; the ’fan-in’ flow receives the replies from these nodes and combines into one, aggregated, reply.

To support message aggregation, three IBM Primitive nodes are provided. These are the:

- AggregateControl node
- AggregateReply node
- AggregateRequest node

The AggregateControl node marks the beginning of a fan-out of the requests that are part of an aggregation.

The AggregateReply node marks the end of a fan-in of the requests that are part of an aggregation. It collects the replies and combines them into a single, compound, reply.

The AggregateRequest node records the fact that request messages have been sent, and collects information that helps the AggregateReply node to construct the compound reply message.

Introduction

In this example, we show how a travel agent might handle requests to arrange a journey that requires a taxi to be booked and a hotel room to be reserved. The message flow that is illustrated in Figure 22 is used.

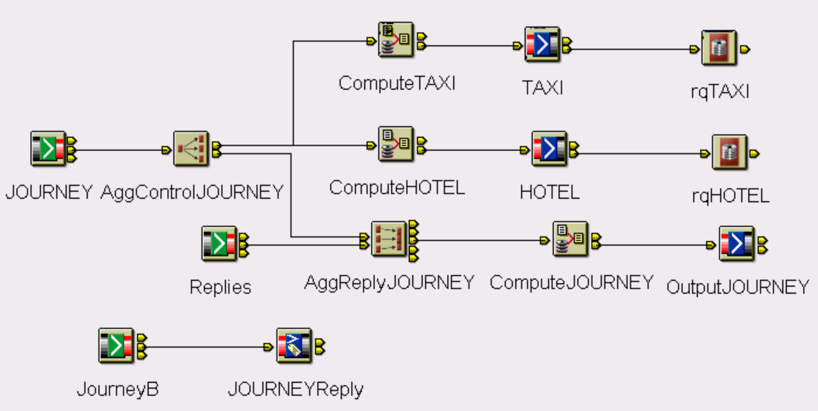


Figure 22. The aggregation message flow.

This message flow is made up of the following subflows:

- The fan-out flow. This flow, which is shown in the upper part of Figure 22, is described in “Creating the travel requests for the journey” on page 66.
- The fan-in flow. This flow, which is shown in the middle part of Figure 22, is described in “Collecting the replies from the journey requests” on page 69.
- The request/reply flow. This flow, which is shown in the lower part of Figure 22, is described in “The request/reply flow” on page 70.

The message flow illustrates the concept of message aggregation using the AggregateControl, AggregateRequest, and AggregateReply nodes. Your requirements might be for a more complex flow than the one shown here, but you can use the techniques shown in this example to create more complicated, or larger, message flows.

The message flow works with messages of a specific format and content. “Defining the message” on page 65 describes how to define the messages that are required.

“Creating and configuring the message flow” on page 66 describes how to define the message flow.

Before you can successfully deploy your message flow, you must set up some MQSeries resources. Refer to “Preparing to implement the example” for more information about what you need to do.

Preparing to implement the example

You must complete the tasks described in the following sections before you can successfully deploy your message flow. Do what is described here before you start to define your message set and message flows.

Setting up MQSeries resources

This example gets an input message for the message flow from an MQSeries queue, JOURNEYIN, on the queue manager that hosts the broker to which the message flow has been deployed. The queue manager puts messages onto an output queue, JOURNEYOUT, when message processing has been completed.

A requests queue, REQUEST, and a reply-to queue, REPLIES, are also required.

The following queues are required :
- JOURNEYIN
- JOURNEYOUT
- REPLIES
- REQUEST

Use the supplied file `journey.tst` to create these queues, or create the queues yourself.

To use `journey.tst`, type the following on a command line on the system on which your broker is running:

```
runmqsc MQSI_SAMPLE_QM <journey.tst
```

If the file, `journey.tst`, is not in your current directory, but is somewhere else on your computer, you must enter the complete path name. Replace the queue manager name in the command above if the name shown is not correct for your broker.

You can use other names for these queues if you want. If you do, you must edit `journey.tst` before you use the `runmqsc` command, and you must modify the queue properties of the nodes in the message flow to match your new names. See "Creating and configuring the message flow" on page 66 for more details.

Defining the message

The self-defining XML message `travelreq.xml` is provided. To import this file, you must:

1. Open a command prompt window.
2. Stop the Configuration Manager using the **mqsistop configmgr** command.
3. Change to the directory containing the file `travelreq.xml`.
4. Import the message by typing:

   ```
   mqsiimpexpmsgset -i -u mqsiuid -p mqsipw -n MQSIMRDB -f travelreq.xml
   ```

   where `mqsiuid` and `mqsipw` are the userID and password that are used for message repository access, and `MQSIMRDB` specifies the name of the message repository. You specified these parameters using the `–n`, `–u`, and `–p` flags on the **mqsicreateconfigmgr** command.
5. Restart the Configuration Manager using the command **mqsistart configmgr**. This picks up the new definition.
6. Restart the Control Center from the Windows NT *Start* menu.
7. In the Control Center Message Sets view, right-click on the Message Sets root and select **Add to Workspace**. Select the message `travelreq.xml`, and click **Finish**.

The message set appears in your workspace.

Creating and configuring the message flow

The file `JOURNEYAggregation.xml` contains the definition of the aggregate message flow. Do the following to import and configure this message flow:

1. Select the Message Flows tab.
2. Import the message flow definition:
   a. Select **Import to Workspace...** from the **File** menu.
   b. Locate file `JOURNEYAggregation.xml` that contains the message flow definition. You can type in the full path and file name in the dialog, or you can click **Browse** and search for the file.
   c. The **Message Flows** check box is checked by default. This will import message flows only from the file you have identified as the source of the resources to import.
   d. Click **Import**. The Control Center processes the file and imports the message flows that are defined within it. A list of resources that have been imported are displayed in a dialog when the processing has completed.
   e. Click **OK** to dismiss the dialog.
3. The new message flow appears in the left hand pane, with a blue cube beside it to show that it is new. Select this new message flow. You can now configure the nodes within it before checking it in to the configuration repository. The majority of the configuration work is already done in the imported flow.
4. Select **File—>Check In—>All (Save to Shared)** to check in the message flows.

If you choose not to use the import file provided, you must create the message flow yourself and complete its design by dragging and dropping the appropriate nodes into the Message Flow Definition pane. You must connect the nodes and configure them as described in the following sections.

Creating the travel requests for the journey

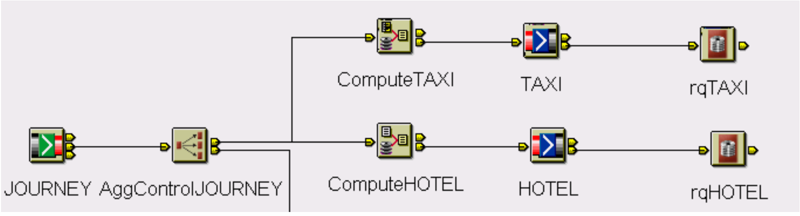


Figure 23. The fan-out flow.

The fan-out flow receives a message from the MQInput node, *JOURNEY*, and uses an AggregateControl node, *AggControlJOURNEY* to fan the message out through two Compute nodes, *ComputeTAXI* and *ComputeHOTEL*, to two MQOutput nodes, *TAXI* and *HOTEL*, and on to two AggregateRequest nodes, *rqTAXI* and *rqHOTEL*. It also sends a control message to the corresponding AggregateReply node on the fan-in flow.

Getting the message

The MQInput node *JOURNEY* retrieves input messages from the input queue.

If you have imported the message flow, this node is fully configured and identifies input queue JOURNEYIN. If you are using a different queue name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. On the Basic tab, enter the input queue name. The default is JOURNEYIN.
2. Click **OK** to finish.
3. Connect the out terminal to the node *AggControlJOURNEY*.

Starting the aggregation

The AggregateControl node, *AggControlJOURNEY*, connects the MQInput node *JOURNEY* to the Compute nodes *ComputeTAXI* and *ComputeHOTEL*.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select the Aggregate Name field and enter the name *JOURNEY*.

   The AggregateName field identifies the function of the aggregation and is used to associate this fan-out message flow with the corresponding fan-in message flow.
2. Select the *Advanced* tab on the Compute node properties dialog, and select an option that includes *Destination* from the drop-down list. For example, Destination And Message.
3. Click **OK** to finish.
4. Connect the out terminal to the two Compute nodes, *ComputeTaxi* and *ComputeHotel*.
5. Connect the control output terminal to the control input terminal of the AggregateReply node.

Booking a taxi

The Compute node, *ComputeTAXI* extracts appropriate information from the message and constructs the request to book a taxi.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use the following ESQL:

```
DECLARE I INTEGER;
DECLARE J INTEGER;
SET J = CARDINALITY (InputRoot.*[]);
SET I = 1;
WHILE I<J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I+1;
END WHILE;
IF CAST(InputRoot.travelreqmsg.carreqd AS INTEGER) = 1
  THEN SET OutputRoot.XML.travelreqmsg.dates_t
 = InputRoot.XML.travelreqmsg.dates_t;
  ELSE RETURN FALSE;
END IF;
```

3. Click **OK** to apply your changes.
4. Connect the out terminal of the Compute node, *ComputeTAXI* to the MQOutput node, *TAXI*.

Reserving a hotel room

The Compute node, *ComputeHOTEL* extracts appropriate information from the message and constructs the request to reserve a hotel room.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use the following ESQL:

```
DECLARE I INTEGER;
DECLARE J INTEGER;
SET J = CARDINALITY (InputRoot.*[]);
SET I = 1;
WHILE I<J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I+1;
END WHILE;
IF CAST(InputRoot.travelreqmsg.hotel_t.required AS INTEGER) = 1
  THEN SET OutputRoot.XML.travelreqmsg.hotel_t
 = InputRoot.XML.travelreqmsg.hotel_t;
       SET OutputRoot.XML.travelreqmsg.dates_t
 = InputRoot.XML.travelreqmsg.dates_t;
  ELSE RETURN FALSE;
END IF;
```

3. Click **OK** to apply your changes.
4. Connect the out terminal of the Compute node, *ComputeHOTEL* to the MQOutput node, *HOTEL*.

Passing on the journey requests

Each MQOutput node, *TAXI* or *HOTEL*, puts the request message received onto its output queue and also passes it on to the AggregateRequest node.

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the Basic properties tab, set a Queue Manager Name (MQSI_SAMPLE_QUEUE is the default name) and a Queue Name (for example, REQUEST).
3. On the Request properties tab, check the Request option box and set a Reply-to queue manager and a Reply-to queue (for example, REPLIES).
4. Click **OK** to apply your changes.
5. Connect the out terminal of the two MQOutput nodes, *TAXI* and *HOTEL*, to the AggregateRequest nodes, *rqTAXI* and *rqHOTEL*, respectively.

Recording the journey requests sent

The two AggregateRequest nodes, *rqTAXI* and *rqHOTEL*, are used to record that a request has been sent.

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the Basic properties tab, set the Folder Name (to TAXI and HOTEL respectively).
3. Click **OK** to apply your changes.

The AggregateRequest node records that a request has been sent. The Folder Name is used by the corresponding AggregateReply node in the fan-in flow.

Collecting the replies from the journey requests

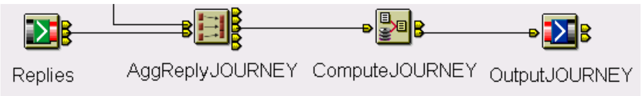


Figure 24. The fan-in flow.

The fan-in flow routes a reply message from the MQInput node, *Replies*, through the AggregateReply node, *AggReplyJOURNEY*, and the Compute node, *ComputeJOURNEY*, to the MQOutput node, *OutputJOURNEY*. A control message is also routed to the AggregateReply node, *AggReplyJOURNEY*, from the AggregateControl node in the fan-out flow.

Getting the message

The MQInput node *Replies* retrieves reply messages from the REPLIES queue.

If you have imported the message flow, this node is fully configured.

If you are creating the message flow yourself, you must:

1. On the Basic tab, set the queue name to REPLIES.
2. Click **OK** to finish.
3. Connect the out terminal to the AggregateReply node *AggReplyJOURNEY*.

Setting the aggregation node

The AggregateReply node *AggReplyJOURNEY* connects the MQInput node *Replies* to the Compute node *ComputeJOURNEY*.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select the Aggregation Name field and enter the name *JOURNEY*.
2. Click **OK** to finish.
3. Connect the out terminal to the node *ComputeJOURNEY*.

Each reply message received by the AggregateReply node is stored in the broker database. When all the replies have been received, an aggregated reply message is created and passed to the Compute node *ComputeJOURNEY*.

Extracting the information for output

The Compute node, *ComputeJOURNEY*, extracts appropriate information from the aggregated reply message and constructs a suitable message to output from the flow.

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use the following ESQL:

   ```
   SET "OutputRoot"."MQMD"."Version" = 2;
   SET "OutputRoot"."MQMD"."xyz" = abc;
   .
   .
   .
   SET "OutputRoot"."XML"."Journey"."TaxiTime" = InputRoot.ComIbmAggregateReplyBody.TAXI.Time;
   ```
3. Click **OK** to apply your changes.

4. Connect the out terminal of the Compute node, *ComputeJOURNEY*, to the MQOutput node, *OutputJOURNEY*.

Outputting the journey information

The MQOutput node, *OutputJOURNEY*, puts the output message onto the output queue *JOURNEYOUT*.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select **Copy message headers**.
2. On the Basic properties tab, set a Queue Manager Name (MQSI_SAMPLE_QUEUE is the default name) and the Queue Name (JOURNEYOUT).
3. Click **OK** to apply your changes.

The request/reply flow

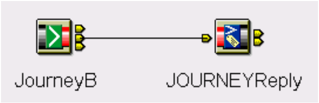


Figure 25. The request/reply flow.

This request/reply flow illustrates the route of a reply message from the MQInput node, *JourneyB*, to the MQReply node, *JOURNEYReply*. It shows the message flow that might be used at the taxi company or hotel to handle a request for a taxi or a hotel room.

Getting the message

The MQInput node *JourneyB* retrieves input messages from the input queue.

If you have imported the message flow, this node is fully configured and identifies input queue JOURNEYOUT. If you are using a different queue name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. On the Basic tab, enter the input queue name, JOURNEYOUT.
2. Click **OK** to finish.
3. Connect the out terminal to the MQReply node, *JOURNEYReply*.

Handling the journey requests

The MQReply node *JOURNEYReply* puts the message on the queue, REPLIES, that is specified in the message header.

Assigning and deploying the scenario

When your message set and message flow definitions are complete, you must check in all the objects that you have been working with before you can assign them to a broker. To do this, select **File—>Check In—>All (Save to Shared)** to check in all objects at once. For other check in options, see the *WebSphere MQ Integrator Using the Control Center* book.

You must now assign and deploy the resources so that you can use the message flow. Select the Assignments view in the Control Center and check out the brokers to which you want to deploy this scenario.

Check out an execution group on each broker to which you are deploying the scenario. Drag and drop the message flow **JOURNEYAggregation** from the Assignable Resource pane to an execution group within the broker. When you have done this, **Check In** the execution group.

When you have completed your assignments, you can deploy your changes. You can do this in one of the following two ways:

1. Deploy the delta or complete configuration data (all types) from the **File** menu.
2. Deploy only Assignments configuration data by right clicking on the broker to which you want to deploy and selecting **Deploy**.

You can check the success of the deployment by going to the Log view, and refreshing the contents of the log.

For more details about assignment and deployment, refer to the *WebSphere MQ Integrator Using the Control Center* book.

Testing the message flow

The SupportPac includes a test message that you can use to test your deployed message flow.

You can test this message flow by putting the XML message `travelreq.xml` to the input queue for this message flow on a broker to which it has been deployed.

Chapter 5. Travel agent (double-flow) example

This example illustrates the use of message aggregation where the fan-in and fan-out message flows are in separate message flows. It shows the use of the AggregateControl, AggregateRequest, and AggregateReply nodes in a more complicated message flow than that shown in the previous scenario.

Introduction

In this example, a travel agent arranges a journey that requires a hotel room and a rental car to be reserved, and the correct currency to be obtained.

The example uses the following message flows:

- JourneyOut. This fan-out flow is shown in Figure 26 on page 76 and is described in "Creating the travel requests for the journey" on page 75.

  It receives a message from the MQInput node, *JOURNEY*, and uses an AggregateControl node, *AggJOURNEY* to fan the message out through five Compute nodes, *ComputeTRAVEL*, *ComputeHOTEL*, *ComputeCARDATES*, *ComputeCURRENCY*, and *SaveMQMDData*, to five MQOutput nodes, *TRAVELLER*, *HOTEL*, *CAR*, *CURRENCY*, and *MQMD_Details*, and five AggregateRequest nodes, *rqTRAVEL*, *rqHOTEL*, *rqDATES*, *rqCURRENCY*, and *rqMQMD*.
- JourneyIn. This fan-in flow is shown in Figure 27 on page 79 and is described in "Collecting the replies from the journey requests" on page 79.

  It routes reply messages from MQInput node, *Replies*, through the AggregateReply node, *AggReplyJOURNEY*, and the Compute node, *ComputeJOURNEY*, to the MQOutput node, *OutputJOURNEY*.

  Other parts of this message flow show how timeout messages, error messages, and undefined messages, are handled.
- JourneyReply. This flow is shown in Figure 28 on page 80 and is described in "The reply flow" on page 80.

  It illustrates the route of the reply message from the MQInput nodes, *TRAVEL*, *HOTEL*, *CAR*, *CURRENCY*, and *SAVED_MQMD*, through optional Compute nodes to the MQReply node, *JOURNEYReply*.

The message flow works with messages of a specific format and content.

"Defining the message" on page 74 describes how to define the messages that are required, and "Creating and configuring the message flows" on page 75 describes how to define the message flows.

Before you can successfully deploy your message flow, you must set up some MQSeries resources. Refer to "Preparing to implement the example", for more information about what you need to do.

Preparing to implement the example

You must complete the tasks described in the following sections before you can successfully deploy your message flow. Do what is described here before you start to define your message set and message flows.

Setting up MQSeries resources

In this example, an input message is obtained from the message flow from an MQSeries queue, JOURNEYIN, on the queue manager that hosts the broker to which the message flow has been deployed.

The queue manager adds messages to the output MQSeries queue, JOURNEYOUT, when message processing has been completed.

The queues required are:
- JOURNEYIN
- JOURNEYOUT
- REPLIES
- CONTROL
- TRAVEL
- HOTEL
- CAR
- CURRENCY
- SAVED_MQMD
- TIMEOUT
- FAILURE
- UNKNOWN

Use the supplied file `journeyagg.tst` to create these queues, or create the queues yourself.

To use `journeyagg.tst`, type the following on a command line on the system on which your broker is running:

```
runmqsc MQSI_SAMPLE_QM <journeyagg.tst
```

If the file, `journeyagg.tst`, is not in your current directory, you must enter the complete path name. Replace the queue manager name in the command above if the name shown is not correct for your broker.

You can use other names for these queues if you want. If you do, you must edit `journeyagg.tst` before you use the `runmqsc` command, and you must modify the queue properties of the nodes in the message flow to match your new names. See "Creating and configuring the message flows" on page 75 for more details.

Defining the message

The self-defining XML message `Travelreq.xml` is provided. To import this file, you must:

1. Open a command prompt window.
2. Stop the Configuration Manager using the **mqsistop configmgr** command.
3. Change to the directory containing the file `RouteToLabel.mrp`.
4. Import the message set by typing:

   ```
   mqsiimpexpmsgset -i -u mqsiuid -p mqsipw -n MQSIMRDB -f Travelreq.xml
   ```

   where `mqsiuid` and `mqsipw` are the userID and password that are used for message repository access, and `MQSIMRDB` specifies the name of the message repository. You specified these parameters using the `–n`, `–u`, and `–p` flags on the **mqsicreateconfigmgr** command.
5. Restart the Configuration Manager using the command **mqsistart configmgr**. This picks up the new definition.

6. Restart the Control Center from the Windows NT *Start* menu.
7. In the Control Center Message Sets view, right-click on the Message Sets root and select **Add to Workspace**. Select the message `Travelreq.xml`, and click **Finish**.

Creating and configuring the message flows

The files `JourneyIn.xml`, `JourneyOut.xml`, and `JourneyReply.xml`contain the definitions of the message flows. Do the following to import and configure the message flows:

1. Select the Message Flows tab.
2. Import the message flow definitions:
   a. Select **Import to Workspace...** from the **File** menu.
   b. Locate files `JourneyIn.xml`, `JourneyOut.xml`, and `JourneyReply.xml` that contain the message flow definitions. You can type in the full path and file names in the dialog, or you can click **Browse** and search for the files.
   c. The **Message Flows** check box is checked by default. This will import message flows only from the files you have identified as the source of the resources to import.
   d. Click **Import**. The Control Center processes the files and imports the message flows that are defined within them. A list of resources that have been imported are displayed in a dialog when the processing has completed.
   e. Click **OK** to dismiss the dialog.
3. The new message flows appear in the left hand pane, with a blue cube beside each one to show that it is new. Select each new message flow in turn. You can now configure the nodes within it before checking it in to the configuration repository. The majority of the configuration work is already done in the imported flow.
4. Select **File—>Check In—>All (Save to Shared)** to check in the message flows.

If you choose not to use the import files provided, you must create the message flows yourself and complete their design by dragging and dropping the appropriate nodes into the Message Flow Definition pane. You must connect the nodes and configure them as described in the following sections.

Creating the travel requests for the journey

This receives each message from an input queue, JOURNEYIN, and passes the message to an AggregateControl node that decides on further processing.

Travel agent (double flow)

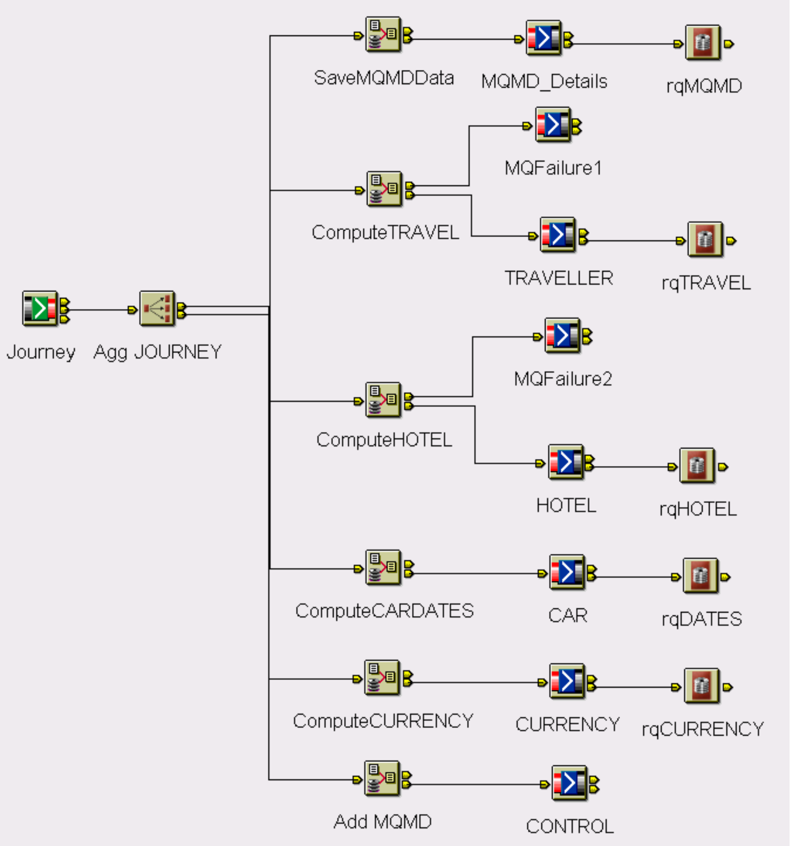


Figure 26. The fan-out message flow.

Getting the message

The MQInput node *JOURNEY* retrieves input messages from the input queue.

If you have imported the message flow, this node is fully configured and identifies input queue JOURNEYIN. If you are using a different queue name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. On the Basic tab, enter the input queue name. The default is JOURNEYIN.
2. Click **OK** to finish.
3. Connect the out terminal to the AggregateControl node *AggJOURNEY*.

Starting the aggregation

The AggregateControl node *AggJOURNEY* connects the MQInput node *JOURNEY* to the five Compute nodes *SaveMQMDData*, *ComputeTRAVEL*, *ComputeHOTEL*, *ComputeCARDATES*, and *ComputeCURRENCY*.

If you have imported the message flow, the Aggregate Control node is fully configured. If you are creating the message flow yourself, you must:

1. Select the Aggregate Name field and enter the name *AggJOURNEY*.
2. Click **OK** to finish.
3. Connect the output terminal to the five Compute nodes listed.
4. Connect the control terminal to the Compute node *AddMQMD*.

Configuring the Compute nodes

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use the following ESQL:
   - For the SaveMQMDData node:

     ```
     DECLARE I INTEGER;
     DECLARE J INTEGER;
     SET J = CARDINALITY (InputRoot.*[]);
     SET I = 1;
     WHILE I<J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I+1;
     END WHILE;
     SET OutputRoot.XML.INMQMD.CorrelId = CAST(InputRoot.MQMD.CorrelId AS CHARACTER);
     SET OutputRoot.XML.INMQMD.MsgId = CAST(InputRoot.MQMD.MsgId AS CHARACTER);
     SET OutputRoot.XML.INMQMD.Version = InputRoot.MQMD.Version;
     SET OutputRoot.XML.INMQMD.ReplyToQ = InputRoot.MQMD.ReplyToQ;
     SET OutputRoot.XML.INMQMD.ReplyToQMgr = InputRoot.MQMD.ReplyToQMgr;
     ```

   - For the ComputeTRAVEL node:

     ```
     DECLARE I INTEGER;
     DECLARE J INTEGER;
     SET J = CARDINALITY (InputRoot.*[]);
     SET I = 1;
     WHILE I<J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I+1;
     END WHILE;
     SET I = CAST(InputRoot.XML.travelreqmsg.travellernum AS INTEGER);
     SET J = 1;
     WHILE J<=I DO
        SET OutputRoot.XML.travelreqmsg.travellerdetails_e[J]
            = InputRoot.XML.travelreqmsg.travellerdetails_e[J];
        SET J = J+1;
     END WHILE;
     ```

   - For the ComputeHOTEL node:

     ```
     DECLARE I INTEGER;
     DECLARE J INTEGER;
     SET J = CARDINALITY (InputRoot.*[]);
     SET I = 1;
     WHILE I<J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I+1;
     END WHILE;
     IF CAST(InputRoot.travelreqmsg.hotel_t.required AS INTEGER) = 1
       THEN SET OutputRoot.XML.travelreqmsg.hotel_t
      = InputRoot.XML.travelreqmsg.hotel_t;
            SET OutputRoot.XML.travelreqmsg.dates_t
      = InputRoot.XML.travelreqmsg.dates_t;
       ELSE RETURN FALSE;
     END IF;
     ```

   - For the ComputeCARDATES node:

     ```
     DECLARE I INTEGER;
     DECLARE J INTEGER;
     SET J = CARDINALITY (InputRoot.*[]);
     SET I = 1;
     WHILE I<J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I+1;
     END WHILE;
     IF CAST(InputRoot.travelreqmsg.carreqd AS INTEGER) = 1
     ```

```
    THEN SET OutputRoot.XML.travelreqmsg.dates_t
   = InputRoot.XML.travelreqmsg.dates_t;
    ELSE RETURN FALSE;
  END IF;
```

- For the ComputeCURRENCY node:

  ```
  DECLARE I INTEGER;
  DECLARE J INTEGER;
  SET J = CARDINALITY (InputRoot.*[]);
  SET I = 1;
  WHILE I<J DO
     SET OutputRoot.*[I] = InputRoot.*[I];
     SET I = I+1;
  END WHILE;
  IF CAST(InputRoot.travelreqmsg.currencyreqd AS INTEGER) = 1
    THEN SET I = CARDINALITY(InputRoot.XML.travelreqmsg.currencyreqd_t[]);
         SET J = 1;
         WHILE J<=I DO;
            SET OutputRoot.XML.travelreqmsg.currencyreqd_t[J]
              = InputRoot.XML.travelreqmsg.currencyreqd_t[J];
            SET J = J+1;
         END WHILE;
    ELSE RETURN FALSE;
  END IF;
  ```

- For the Add MQMD node:

  ```
  SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
  SET OutputRoot.MQMD.StrucId = MQMD_STRUC_ID;
  SET OutputRoot.MQMD.XML = InputRoot.XML;
  ```

3. Click **OK** to apply your changes.
4. Connect the out terminal of the five Compute nodes, *SaveMQMDData*, *ComputeTRAVEL*, *ComputeHOTEL*, *ComputeCARDATES*, and *ComputeCURRENCY*, to the MQOutput nodes, *MQMD_Details*, *TRAVELLER*, *HOTEL*, *CAR*, and *CURRENCY*, respectively.
5. Connect the error terminal of the Compute node, *ComputeTRAVEL*, to the MQOutput node, *MQFailure1*, and connect the error terminal of the Compute node, *ComputeHOTEL*, to the MQOutput node, *MQFailure2*.

Passing on the travel requests

If you have imported the message flow, the MQOutput nodes are fully configured.

If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the Basic properties tab, select a default queue manager and the following queues:
   - For the MQMD_Details node: SAVED_MQMD
   - For the TRAVELLER node: TRAVEL
   - For the HOTEL node: HOTEL
   - For the CAR node: CAR
   - For the CURRENCY node: CURRENCY
   - For the CONTROL node: CONTROL
   - For the MQFailure1 node: FAILURE
   - For the MQFailure2 node: FAILURE1
3. Click **OK** to apply your changes.
4. Connect the out terminal of the five MQOutput nodes, *MQMD_Details*, *TRAVELLER*, *HOTEL*, *CAR*, and *CURRENCY*, to the AggregateRequest nodes, *rqMQMD*, *rqTRAVEL*, *rqHOTEL*, *rqDATES*, and *rqCURRENCY*, respectively.

Recording the journey requests sent

The AggregateRequest nodes are used to record the journey requests that have been sent.

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the BASIC tab, select the following folder names:
   - For the rqMQMD node: MsgRequestMQMD
   - For the rqTRAVEL node: MsgRequest1
   - For the rqHOTEL node: MsgRequest2
   - For the rqDATES node: MsgRequest3
   - For the rqCURRENCY node: MsgRequest4
3. Click **OK** to apply your changes.

Collecting the replies from the journey requests

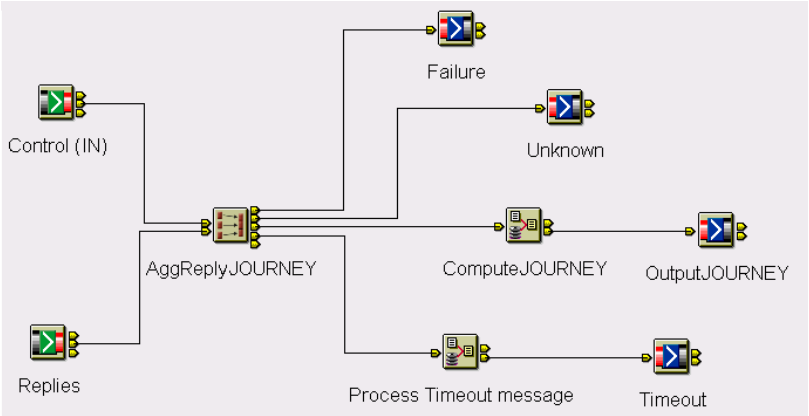


Figure 27. The fan-in message flow.

Setting the aggregation node

The AggregateReply node *AggReplyJOURNEY* connects the MQInput node *Replies* to the Compute node *ComputeJOURNEY*.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select the Aggregate Name field and enter the name *JOURNEY*.
2. Click **OK** to finish.
3. Connect the output terminal to the Compute node, *ComputeJOURNEY*.
4. Connect the failure terminal to the MQOutput node, *Failure*.
5. Connect the unknown terminal to the MQOutput node, *Unknown*.
6. Connect the timeout terminal to the Compute node, *Process Timeout message*.

Extracting information to be output

The Compute node, *ComputeJOURNEY*, extracts appropriate information from the aggregated reply message and constructs a suitable message to output from the flow.

Travel agent (double flow)

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use the following ESQL:
3. Click **OK** to apply your changes.
4. Connect the out terminal of the Compute node, *ComputeJOURNEY*, to the MQOutput node, *OutputJOURNEY*.

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

Outputting information about the journey

If you have imported the message flow, the MQOutput nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the BASIC tab, select a default queue manager and the following queues:
   - For the Failure node: FAILURE
   - For the Unknown node: UNKNOWN
   - For the OutputJOURNEY node: JOURNEYOUT
   - For the Timeout node: TIMEOUT
3. Click **OK** to apply your changes.

The reply flow

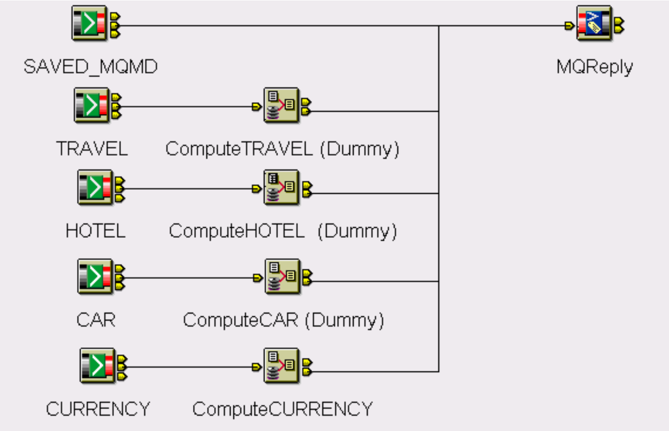


Figure 28. The reply message flow.

Getting the message

The MQInput nodes, *SAVED_MQMD*, *TRAVEL*, *HOTEL*, *CAR*, and *CURRENCY* retrieve messages from the REPLIES queue.

If you have imported the message flow, these nodes are fully configured and identify the input queue REPLIES. If you are using a different queue name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. On the Basic tab, enter the input queue name. The default is REPLIES.
2. Click **OK** to finish.

3. Connect the out terminals to the Compute nodes *ComputeTRAVEL*, *ComputeHOTEL*, *ComputeCAR*, and *ComputeCURRENCY*.

Setting the Compute nodes

The Compute nodes, *ComputeTRAVEL*, *ComputeHOTEL*, *ComputeCAR*, and *ComputeCURRENCY*, can be used to add value to the input message before creating the output message.

If you have imported the message flow, these nodes are dummy Compute nodes. The input message is copied unchanged into the output message. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use your own ESQL to add value to the input message and create an output message.
3. Click **OK** to apply your changes.
4. Connect the out terminal to the MQOutput node, *JOURNEYReply*.

Responding to the travel request

The MQReply node *JOURNEYReply* puts the message on the queue, REPLIES, that is specified in the message header.

Assigning and deploying the scenario

When your message set and message flow definitions are complete, you must check in all the objects that you have been working with before you can assign them to a broker. To do this, select **File—>Check In—>All (Save to Shared)** to check in all objects at once. (For other check in options, see the *WebSphere MQ Integrator Using the Control Center* book.)

You must now assign and deploy the resources so that you can use the message flow. Select the Assignments view in the Control Center and check out the brokers to which you want to deploy this scenario.

Check out an execution group on each broker to which you are deploying the scenario. Drag and drop the message flow **JOURNEYAggregation** from the Assignable Resource pane to an execution group within the broker. When you have done this, **Check In** the execution group.

When you have completed your assignments, you can deploy your changes. You can do this in one of the following two ways:

1. Deploy the delta or complete configuration data (all types) from the **File** menu.
2. Deploy only Assignments configuration data by right clicking on the broker to which you want to deploy and selecting **Deploy**.

You can check the success of the deployment by going to the Log view, and refreshing the contents of the log.

For more details about assignment and deployment, refer to the *WebSphere MQ Integrator Using the Control Center* book.

Testing the message flow

The SupportPac includes a test message that you can use to test out your deployed message flow.

You can test this message flow by putting the XML message `travelreq.xml` to the input queue for this message flow on a broker to which it has been deployed.

Chapter 6. Estimations example

This example illustrates the use of message aggregation where the fan-in and fan-out message flows are in separate message flows. Like the previous two examples, this example shows the use of the AggregateControl, AggregateRequest, and AggregateReply nodes.

Introduction

In this example, you request pricing quotations, or price estimates, for work to repair a fence that has been damaged by recent storms.

The quotation request is sent to five different companies, some of whom might not reply. The example shows how you can aggregate the quotations received into one reply message in which the quotations are sorted in ascending order of preferred supplier (based upon the quoted price).

The example uses the following message flows:
- GetEstimatesOut. This fan-out flow is shown in Figure 29 on page 85 and is described in “Requesting estimates” on page 85. The message flow sends requests for quotations to five different suppliers and specifies a time that will be waited before replies received are analyzed.
- GetEstimatesIn. This fan-in flow is shown in Figure 31 on page 87 and is described in “Collecting the estimates” on page 87. The message flow gathers together the information sent back from the five suppliers, makes sure that at least three replies have been received, and sorts the replies in ascending order of the price quotations.
- GetEstimatesReply. This flow is shown in Figure 30 on page 87 and is described in “The reply flow” on page 86.

“Creating and configuring the message flows” on page 84 describes how to define the message flows.

Before you can successfully deploy your message flow, you must set up some MQSeries resources. Refer to “Preparing to implement the example”, for more information about what you need to do.

Preparing to implement the example

You must complete the tasks described in the following sections before you can successfully deploy your message flow. Do what is described here before you start to define your message set and message flows.

Setting up MQSeries resources

In this example, an input message is obtained from the message flow from an MQSeries queue on the queue manager that hosts the broker to which the message flow has been deployed. The queue manager puts messages on to an output queue when message processing has been completed.

The queues required are:
- ESTREQUEST

Estimations example

- DEVER
- PANDA
- HAYDOCK
- CLOVER
- DREWS
- CONTROL
- REPLIES
- FAILURE
- UNKNOWN
- ESTIMATESOUT
- TIMEOUT

Use the supplied file `GetEstimatesAgg.tst` to create these queues, or create the queues yourself.

To use `GetEstimatesAgg.tst`, type the following on a command line on the system on which your broker is running:

```
runmqsc MQSI_SAMPLE_QM < GetEstimatesAgg.tst
```

You must enter the complete path for the file `GetEstimatesAgg.tst` if it is not in your current directory. Replace the queue manager name in the command above if the name shown is not correct for your broker.

You can use different names for these queues if you choose. If you do, you must edit `GetEstimatesAgg.tst` before you use the `runmqsc` command, and you must modify the queue properties of the nodes in the message flow to match your new names. See "Creating and configuring the message flows" for more details.

Creating and configuring the message flows

The files `GetEstimatesIn.xml`, `GetEstimatesOut.xml`, and `GetEstimatesAggReply.xml`contain the definitions of the message flows. Do the following to import and configure the message flows:

- Select the Message Flows tab.
- Import the dynamic routing message flow definition:
  1. Select **Import to Workspace...** from the **File** menu.
  2. Locate files `GetEstimatesIn.xml`, `GetEstimatesOut.xml`, and `GetEstimatesAggReply.xml` that contain the message flow definitions (you can type in the full path and file names in the dialog, or you can click **Browse** and search for the files).
  3. The **Message Flows** check box is checked by default. This will import message flows only from the files you have identified as the source of the resources to import.
  4. Click **Import**. The Control Center processes the files and imports the message flows that are defined within them. Each file contains just one message flow. A list of resources that have been imported are displayed in a dialog when the processing has completed.
  5. Click **OK** to dismiss the dialog.
- The new message flows appear in the left hand pane, with a blue cube beside each one to show that it is new. Select each new message flow in turn. You can now configure the nodes within it before checking it in to the configuration repository. The majority of the configuration work is already done in the imported flow.

- Select **File—>Check In—>All (Save to Shared)** to check in the message flows.

If you choose not to use the import files provided, you must create the message flows yourself and complete their design by dragging and dropping the appropriate nodes into the Message Flow Definition pane. You must connect the nodes and configure them as described in the following sections.

Requesting estimates

A fan-out flow is used to request estimates from several potential suppliers of fencing. The message is received from the MQInput node *EstRequest*. The AggregateControl node *AggGetEstimates* is used to fan the request out through MQOutput nodes to each of the potential suppliers of fencing.

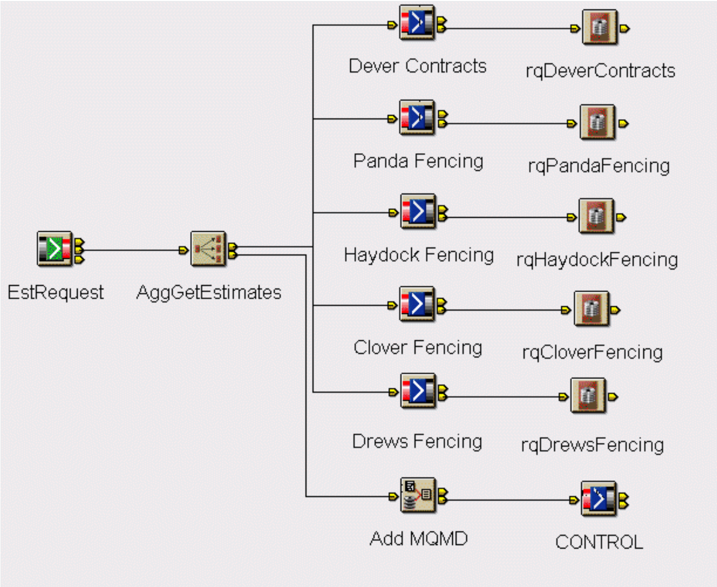


Figure 29. The fan-out message flow.

Getting the message

The MQInput node *EstRequest* retrieves input messages from the input queue.

If you have imported the message flow, this node is fully configured and identifies input queue ESTREQUEST. If you are using a different queue name, you must update the node properties. All other properties are left to take default values.

If you are creating the message flow yourself, you must:

1. On the Basic tab, enter the input queue name. The default is ESTREQUEST.
2. Click **OK** to finish.
3. Connect the out terminal to the AggregateControl node *AggGetEstimates*.

Starting the aggregation

The AggregateControl node *AggGetEstimates* connects the MQInput node *EstRequest* to the five MQOutput nodes *Dever Contracts*, *Panda Fencing*, *Haydock Fencing*, *Clover Fencing*, and *Drews Fencing*. The five MQOutput nodes correspond to the five companies from which an estimate has been requested.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select the Aggregate Name field and enter the name *AggGetEstimates*.
2. Click **OK** to finish.
3. Connect the output terminal to the five MQOutput nodes listed.
4. Connect the control terminal to the Compute node *AddMQMD*.

Requesting the estimates

If you have imported the message flow, these nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the BASIC tab, select a default queue manager and the following queues:
   - For the Dever Contracts node: DEVER
   - For the Panda Fencing node: PANDA
   - For the Haydock Fencing node: HAYDOCK
   - For the Clover Fencing node: CLOVER
   - For the Drews Fencing node: DREWS
3. Click **OK** to apply your changes.
4. Connect the out terminal of the five MQOutput nodes, *Dever Contracts*, *Panda Fencing*, *Haydock Fencing*, *Clover Fencing*, and *Drews Fencing*, to the AggregateRequest nodes, *rqDever Contracts*, *rqPanda Fencing*, *rqHaydock Fencing*, *rqClover Fencing*, and *rqDrews Fencing*, respectively.

Recording the estimates requested

If you have imported the message flow, these AggregateRequest nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the BASIC tab, select the following folder names:
   - For the rqDeverContracts node: DeverContracts
   - For the rqPandaFencing node: PandaFencing
   - For the rqHaydockFencing node: HaydockFencing
   - For the rqCloverFencing node: CloverFencing
   - For the rqDrewsFencing node: DrewsFencing
3. Click **OK** to apply your configuration.

The reply flow

The fan-in message flow shown in Figure 30 on page 87 is used to gather together the estimates from the suppliers of fencing.

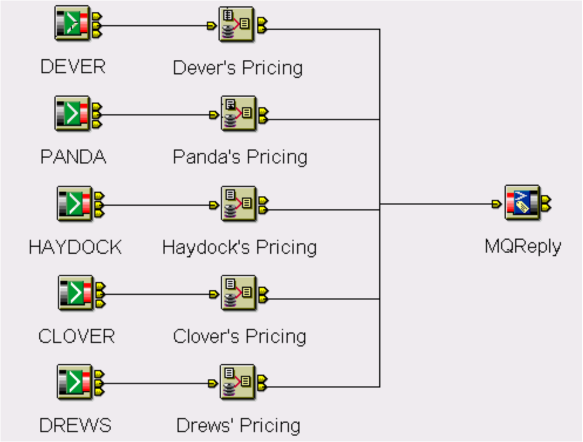


Figure 30. The reply message flow.

Getting the message

The MQInput nodes *DEVER*, *PANDA*, *HAYDOCK*, *CLOVER*, and *DREWS* retrieve input messages from the input queues, DEVER, PANDA, HAYDOCK, CLOVER, and DREWS respectively.

If you have imported the message flow, these nodes are fully configured and identify the input queues specified above. If you want to use different queue name, you must update the node properties accordingly. All other properties are left to take default values.

If you are creating the message flow yourself, for each MQInput node you must:

1. On the Basic tab, enter the input queue name.
2. Click **OK** to finish.
3. Connect the out terminal to the appropriate Compute node: *Dever's Pricing*, *Panda's Pricing*, *Haydock's Pricing*, *Clover's Pricing*, or *Drews' Pricing*.

Collecting the estimates

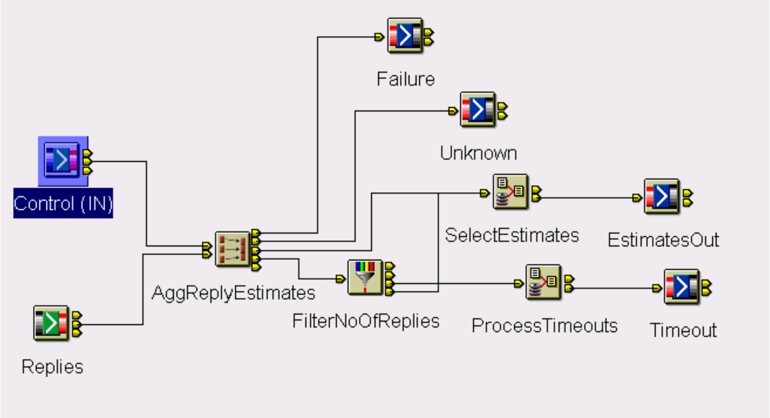


Figure 31. The fan-in message flow.

Estimations example

Getting the message

The MQInput node *Replies* retrieves reply messages from the REPLIES queue.

If you have imported the message flow, this node is fully configured.

If you are creating the message flow yourself, you must:

1. On the Basic tab, set the queue name to REPLIES.
2. Click **OK** to finish.
3. Connect the out terminal to the AggregateReply node *AggReplyEstimates*.

The MQInput node *Control (IN)* retrieves reply messages from the CONTROL queue.

If you have imported the message flow, this node is fully configured.

If you are creating the message flow yourself, you must:

1. On the Basic tab, set the queue name to CONTROL.
2. Click **OK** to finish.
3. Connect the out terminal to the AggregateReply node *AggReplyEstimates*.

Setting the aggregation node

The AggregateReply node *AggReplyEstimates* connects the MQInput node *Replies* to the Compute node *SelectEstimates*.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. Select the Aggregate Name field and enter the name *AggGetEstimates*.
2. Click **OK** to finish.
3. Connect the output terminal to the Compute node, *SelectEstimates*.
4. Connect the failure terminal to the MQOutput node, *Failure*.
5. Connect the unknown terminal to the MQOutput node, *Unknown*.
6. Connect the timeout terminal to the Filter node, *FilterNoOfReplies*.

Analyzing the replies

The Filter node *FilterNoOfReplies* receives an aggregated reply message which contains all the estimates received by the time specified in the Timeout property of the AggregateControl node.

The Filter node then decides which of the replies in the aggregated reply message meet the criteria specified in the original requests for estimates from each of the five suppliers. There must be valid replies from at least three of the suppliers for the request to be forwarded through the output terminal; otherwise, the failure terminal is chosen.

If you have imported the message flow, this node is fully configured. If you are creating the message flow yourself, you must:

1. On the Basic tab, enter the queue name REPLIES.
2. Click **OK** to finish.
3. Connect the output terminal to the Compute node, *Select Estimates*.
4. Connect the failure terminal to the Compute node, *ProcessTimeouts*.

Preparing the final response

If you have imported the message flow, the Compute nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.
2. On the ESQL tab, use the following ESQL:
   - For the SelectEstimates node:

```
DECLARE i INTEGER;
DECLARE j INTEGER;
DECLARE k INTEGER;
DECLARE MyInput REFERENCE TO InputRoot.ComIbmAggregateReplyBody;
--
-- Set the MQMD
SET OutputRoot.MQMD.StrucId = MQMD_STRUC_ID;
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
--
-- Copy Input message to Output message adding the company name
-- taken from the folder name
--
SET i = CARDINALITY (MyInput.*[]);
SET j = 1;
WHILE j<=i DO
   SET OutputRoot.XML.Estimates.Company[j].Name = FIELDNAME (MyInput.*[j]);
   SET OutputRoot.XML.Estimates.Company[j].Data = MyInput.*[j].XML.GetEstimates;
END WHILE;
DECLARE MyOutput REFERENCE TO OutputRoot.XML.Estimates;
--
-- and now sort the output into ascending order
--
SET j = 1;
WHILE j<i DO
   SET k = j+1;
   WHILE k<=i DO
      IF CAST (MyOutput.Company[k].Data.quotation AS DECIMAL)
         < CAST (MyOutput.Company[j].Data.quotation AS DECIMAL)
         THEN SET MyPutput.Company[i+1].Name = MyOutput.Company[k].Name;
              SET MyPutput.Company[i+1].Data = MyOutput.Company[k].Data;
              SET MyPutput.Company[k].Name = MyOutput.Company[j].Name;
              SET MyPutput.Company[k].Data = MyOutput.Company[j].Data;
              SET MyPutput.Company[j].Name = MyOutput.Company[i+1].Name;
              SET MyPutput.Company[j].Data = MyOutput.Company[i+1].Data;
      END IF;
      SET k = k+1;
   END WHILE;
   SET j = j+1;
END WHILE;
--
-- and delete the temporary i+1 slot
--
DETACH MyOutput.Company[i+1].Name;
DETACH MyOutput.Company[i+1].Data;
```

   - For the ProcessTimeouts node:

```
SET OutputRoot.MQMD.StrucId = MQMD_STRUC_ID;
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
SET OutputRoot.XML.Test = InputRoot.ComIbmAggregateReplyBody;
```

3. Click **OK** to apply your changes.
4. Connect the out terminal of the two Compute nodes, *SelectEstimates* and *ProcessTimeouts*, to the MQOutput nodes, *EstimatesOut* and *Timeout*, respectively.

Providing the estimates

If you have imported the message flow, the MQOutput nodes are fully configured. If you are creating the message flow yourself, for each node you must:

1. Select **Copy message headers**.

2. On the BASIC tab, select a default queue manager and the following queues:
   - For the Failure node: FAILURE
   - For the Unknown node: UNKNOWN
   - For the EstimatesOut node: ESTIMATESOUT
   - For the Timeout node: TIMEOUT
3. Click **OK** to apply your changes.

Assigning and deploying the scenario

When your message set and message flow definitions are complete, you must check in all the objects that you have been working with before you can assign them to a broker. To do this, select **File—>Check In—>All (Save to Shared)** to check in all objects at once. (For other check in options, see the *WebSphere MQ Integrator Using the Control Center* book.)

You must now assign and deploy the resources so that you can use the message flow. Select the Assignments view in the Control Center and check out the brokers to which you want to deploy this scenario. Drag and drop the message set from the Assignable Resources pane to the brokers pictured in the Domain Topology pane. When you have finished these updates, check in the brokers.

Check out an execution group on each broker to which you are deploying the scenario. Drag and drop the message flows from the Assignable Resource pane to an execution group within each broker. When you have done this, check in the execution groups.

When you have completed your assignments, you can deploy your changes. You can do this in one of the following two ways:

1. Deploy the delta or complete configuration data (all types) from the **File** menu.
2. Deploy only Assignments configuration data by right clicking on the broker to which you want to deploy and selecting **Deploy**.

You can check the success of the deployment by going to the Log view, and refreshing the contents of the log.

For more details about assignment and deployment, refer to the *WebSphere MQ Integrator Using the Control Center* book.

Testing the message flow

The SupportPac includes a test message that you can use to test out your deployed message flow.

You can test this message flow by putting the XML message `GetEstimatesMsg.xml` to the input queue for this message flow on a broker to which it has been deployed.

Appendix. 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.

Notices

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
SupportPac
IBM
VisualAge
MQSeries
WebSphere

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, 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 AIX Version 2.1 Installation Guide*, GC34-5841
  IBM *WebSphere MQ Integrator for HP-UX Version 2.1 Installation Guide*, GC34-5907
  IBM *WebSphere MQ Integrator for Sun Solaris Version 2.1 Installation Guide*, GC34-5842
  IBM *WebSphere MQ Integrator for Windows NT Version 2.1 Installation Guide*, GC34-5600
  IBM *WebSphere MQ Integrator for z/OS Version 2.1 Installation Guide*, GC34-5919

  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 ESQL Reference*, 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 Programming Guide*, SC34-5603

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

  This book is a reference book for WebSphere MQ Integrator Version 2.1 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.
- IBM *WebSphere MQ Integrator Version 2.1 Working with Messages*, GC34-6039

  This book explains the message model used by WebSphere MQ Integrator, and discusses the message domains that are supported, and the structure of messages within those domains. It also provides guidance for customizing message flows to process the messages in each supported domain.

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.

Printed in U.S.A.