WebSphere® MQ Integrator Enabler

Model Office Reference Manual

NOTE:

Before using this information and the product it supports, read the information in *Notices* on page 66.

Fifth Edition (June 2002)

Printed in USA.

Contents

About this book

Who should read this book

Anyone who is working on an WMQI Enabler implementation can review this reference manual for examples of the various phases of a working WMQI Enabler solution. Adapter developers may find it helpful to read the discussions on the front-end and back-end sample applications, as they simulate the work an adapter might do.

Terminology used in this book

All new terms introduced in this book are defined in the *Glossary*.

This book uses the following shortened names:

- MQSeries®: a general term for IBM MQSeries messaging products.
- Model Office: a general term used to describe an WMQI Enabler development and test environment that includes sample front-end and back-end test applications with implemented use cases.
- Use Case: a general term that refers to the business scenarios used in the Model Office.

How to get additional information

Visit the following home page at:

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

By following this link you can find:

- The latest information about MQSeries family of products.
- Download SupportPacks.
- Access FAQs.
- Access MQSeries family publications library.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments or suggestions about this book or any other *WebSphere MQ Integrator Enabler* documentation:

- 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 44-1962-816151
  - From within the U.K., use 01962-816151

- 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

Chapter 1
Introduction

This manual is designed to serve as a reference tool for the implementation of WebSphere MQ Integrator Enabler (WMQI Enabler).

Adapter programs are needed to convert XML messages (for example, Open Application Group (OAG) or Insurance Application Architecture (IAA) messages) into actions to be used by an issuing or receiving application and are not included as part of the Model Office. Adapter programs are very specific to an application and key IBM Business Partners are already supporting XML messages. The WMQI Enabler Application Integration Guide lays out design considerations and the services that need to be provided by these Adapter programs. Plus the Industry Reference Manuals provide applicable notes on the Adapter Services that may be unique by industry or XML language.

Identification of what the business environment is to do and accomplish is the essential starting point of an implementation project. It is this statement of the business requirements that will drive the customization of the Model Office business scenarios or creation of new business scenarios. These scenarios will be used to develop the required system interaction diagrams and flows that will constitute the new WMQI Enabler implementation.

The WMQI Enabler architecture uses key functions of MQSeries Integrator in conjunction with MQSeries Workflow to provide the advanced WMQI Enabler features that can be used to simplify the integration of applications across the enterprise. Using the WMQI Enabler product can significantly increase productivity by providing the working architecture and technologies. This starting point allows the project team to focus their work and effort on the business events and data needed to make applications inter-operate.

Most of all, the WMQI Enabler solution will provide integration development with a standardized, repeatable approach.

Also provided are instructions for utilizing the testing tool that is provided with the product for testing of the various XML messages. Another quick start for implementations may be found with the description of the WMQI Enabler - Configurator for building the Hub_Only messages and maintaining those messages. For more information on these messages see *Appendix C* of the ***Installation and Setup Guide*** and *Chapter 5* of the ***Development Guide***.

Chapter 2
Options for XML languages

WMQI Enabler use is optimized by a common integration dialect so that numerous applications or systems can communicate in an understandable format. The above XML languages provide that for the referenced industries. In many enterprises today, however, the boundaries of one industry are becoming inter-meshed with other industries and the communication requirements are more complex. In addition, many enterprises desire to maintain a consistent meaning for the data utilized in their enterprise.

In order to communicate, two applications must share the same understanding of the world. For example, if one application processes a person's address as a single string of data, and another processes it as comprising street, town, state and country, they have a built-in impedance to communication.

In order to use the IDM or Model Based XML (MB-XML) approach, a normalized model is needed that guarantees unique semantics and content structure, i.e., a common reference point. For example, when two applications communicate about a Postal Address, they can agree on what it means, what data it contains, and how it is related to other concepts (such as people and policies).

Recognizing this issue, WMQI Enabler has included an Interface Design Model (IDM) for each of the industries that takes the data involved with each of these dialects and has created a common model. This common model ensures that the use of data throughout the enterprise has a common understanding or meaning.

A sample model is provided with WMQI Enabler that can be used to create such "multi-industry" messages. Messages created by this model are not OAG, IAA, or IFX XML but are created as MB-XML for use throughout an enterprise, and represent the fourth XML option contained in WMQI Enabler.

Chapter 3
Message setup quick start

WMQI Enabler is intended to process messages from a front-end system or application to a single or multiple back-end systems or applications, and to return response messages to the front-end system or application. The intent of this chapter is to highlight the steps needed to accomplish the generation of these messages along with the testing of the messages to act as a quick start to the implementation of WMQI Enabler .

As these steps are presented, an indication of where to find more complete documentation will also be offered.

While WMQI Enabler supports many variations, for explanation purposes the following specifications have been used:

- WMQI Enabler has been installed successfully.
- **SetDestinationIDM** workflow process template is used in MQSeries Workflow.
- The front-end system symbolic is **FrontEnd**.
- The back-end system symbolic is **BESystem**.
- The back-end system is set to use QueueManager **MQSIQM**.
- The back-end system is set to use queue **BESYSTEM**.
- The userid is set to **USERID**.
- Store-forward is disabled.
- Sequence validation is disabled.
- Session validation is enabled.
- System interaction check is disabled.
- The test message name is set to **HubTest**.
- MQTester is installed in the **C:\tester** directory.
- MQTester is used to process the test message.
- The TestSuite name is set to **ExampleTestSuite.xml**.

The message construction process is broken down as follows:

1. Identify source and target systems.
2. Define message profile requirements.
3. Build message profile message.
4. Setup a workflow in MQSeries Workflow.
5. Define System Symbolic(s) and SDR requirements.
6. Build SDR message.
7. Build SystemRestart message.
8. Build a request message.
9. Build a response message.
10. Create a TestSuite.

The message debugging/execution process is as follows:

1. Define any new MQSeries entities.
2. Start MQTester.
3. Debug message profile message.
4. Debug back-end system SDR message.
5. Debug back-end SystemRestart message.
6. Debug HubTest request message.
7. Debug HubTest response message.
8. Revise ExampleTestSuite.xml for ease of use.

The message construction and debugging/execution are further described, in the following sections.

Message construction process

Identify source and target systems

Before a TestSuite can be executed, the source and target systems that participate in the message request/response must be identified. For this example, the participating systems are FrontEnd and BESystem. BESystem is a new system; FrontEnd already exists in the model office configuration.

Define message profile requirements

Message profile requirements that need to be defined are as follows:

- The name of the message is set to HubTest.
- SetDestinationIDM workflow process template is used in MQSeries Workflow.
- The back-end system symbolic is BESystem.
- The userid is set to USERID.
- Store-forward is disabled.
- Sequence validation is disabled.
- Session validation is enabled.
- System interaction check is disabled.
- For the remaining setting use the defaults found in existing message profiles.

The Message Profile is specified in the **Development Guide**.

Build Message profile message

The message profile may also be generated using the WMQI Enabler - Configurator , which is documented in the ***Model Office Reference Manual****, Chapter 5.*

The simplest way to define a new message profile is to use an existing message profile as a template and update the message profile.

The example below uses an existing **UpdateMessageProfile** as a template where the points mentioned above in “Define message profile requirements” were modified.

The resulting message is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Message id="M5551920" sessionId="1654651" version="1.4"
bodyType="HUBONLYOFFLINE" timeStampCreated="2000-31-Aug"
timeStampExpired="" sourceLogicalId="FrontEnd"
authenticationId="SysAdmin" crfPublish="true"
crfCmdMode="alwaysRespond" txnScope="all">
```

Sample messages are found on the WMQI Enabler Product CD.

```
<COMMAND cmdType="UpdateMessageProfile">
      <MessageTypeName>HubTest</MessageTypeName>
      <MQSISessionValidationFlag>False</MQSISessionValidationFlag>
  <MQSIMessageSequenceValidationFlag>False</MQSIMessageSequenceValidat
ionFlag>
  <MQSISystemInteractionCheckFlag>False</MQSISystemInteractionCheckFla
g>
      <WorkFlowManagementFlag>True</WorkFlowManagementFlag>
      <WorkFlowQueueManager>FMCQM</WorkFlowQueueManager>
    <WorkFlowDataStructureName>ProcessTemplateExecute</WorkFlowDataStr
uctureName>
    <WorkFlowProcessName>SetDestinationIDM</WorkFlowProcessName>
       <WorkFlowQueue>FMC.FMCGRP.EXE.XML</WorkFlowQueue>
       <WorkFlowSymbolic>Workflow1</WorkFlowSymbolic>
       <WorkFlowReplyToQueueManager>MQSIQM</WorkFlowReplyToQueueManage
r>
       <WorkFlowReplyToQueue>MQWF_END</WorkFlowReplyToQueue>
       <HubQueueManager>MQSIQM</HubQueueManager>
    <DefaultDestinationSymbolic>WorkFlowDefault</DefaultDestinationSym
bolic>
       <MessageTypeDependency></MessageTypeDependency>
       <TraceFlag>False</TraceFlag>
        <SystemInteractionList>
           <Item>
              <SystemSymbolic>BESystem</SystemSymbolic>
              <MessageTypeName>HubTest</MessageTypeName>
              <RequiredInteractionFlag>True</RequiredInteractionFlag>
              <SystemBackup></SystemBackup>
          </Item>
    </SystemInteractionList>
  </COMMAND>
</Message>
```

Setup a workflow in MQSeries Workflow

More information on MQSeries Workflow usage can be found in the ***Development Guide****, Chapter 7.*

Define a workflow process template using MQSeries Workflow. Workflow is normally used to define the step required for message processing.

In this example, we are using **SetDestinationIDM**, but other workflows can also be defined.

The figure below shows the **SetDestinationIDM** workflow process template taken from MQSeries Workflow:

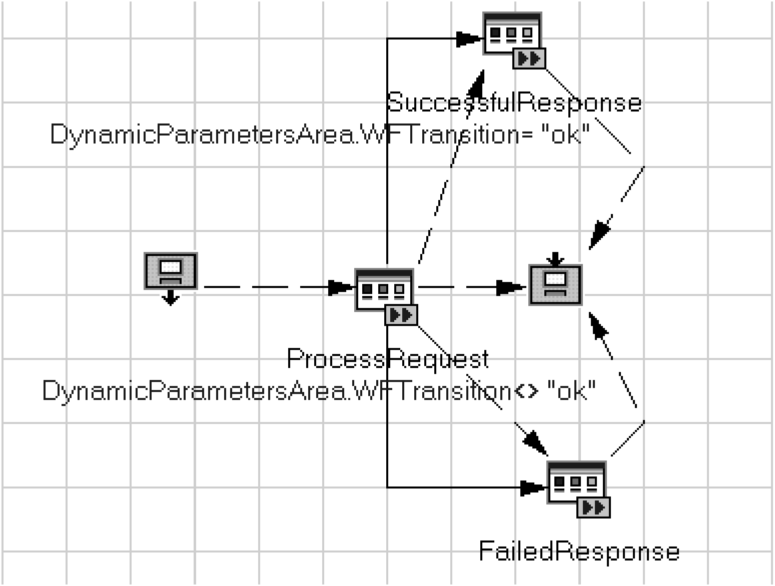

SuccessfulResponse
DynamicParametersArea.WFTransition= "ok"
ProcessRequest
DynamicParametersArea.WFTransition<> "ok"
FailedResponse

Figure 1: SetDestinationIDM workflow process template.

Define System Symbolic(s) and SDR requirements

There is one new system being added to the Hub.

- Its symbolic is **BESystem**.
- Its QueueManager is **MQSIQM**.
- Its queue is **BESYSTEM**.

Build SDR message

*SDR is discussed in the **Development Guide**, Chapter 2 and 6.*

The simplest way to define a new SDREntry is to use an existing SDREntry as a template and update the SDR.

The example below uses an existing **UpdateSDRentry** as a template where the points mentioned above in Define the message profile requirements were modified.

The SDR may also be generated using the WMQI Enabler - Configurator, which is documented in the ***Model Office Reference Manual****.*

The resulting message is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Message id="M5551920" sessionId="1654651" version="1.4"
bodyType="HUBONLYOFFLINE" timeStampCreated="2000-31-Aug"
timeStampExpired="" sourceLogicalId="BESystem"
authenticationId="SysAdmin" crfPublish="true"
crfCmdMode="alwaysRespond" txnScope="all">
       <COMMAND cmdType="UpdateSDREntry">
           <SystemSymbolic>BESystem</SystemSymbolic>
           <Queue>BESYSTEM</Queue>
           <QueueManager>MQSIQM</QueueManager>
       </COMMAND>
</Message>
```

Build SystemRestart message

The simplest way to define a new System Restart message is to use an existing SystemRestart message as a template and update the SystemRestart message.

The example below uses an existing **SystemRestart** message as a template where the points mentioned above in “Define message profile requirements” were modified.

Sample messages are found on the WMQI Enabler Product CD.

The resulting message is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?><Message id="M5551920"
sessionId="1654651" version="1.4" bodyType="HUBONLYONLINE"
timeStampCreated="2000-31-Aug" timeStampExpired=""
sourceLogicalId="BESystem" authenticationId="SysAdmin"
crfPublish="true" crfCmdMode="alwaysRespond" txnScope="all">
    <COMMAND cmdType="SystemRestart">
    </COMMAND>
</Message>
```

Build a request message

The contents of request/response messages are, by definition, message specific. For this example, a simple message, HubTest, is sent into the hub by system symbolic FrontEnd. The COMMAND section of the HubTest message is a simple placeholder.

The HubTest message includes a bodyCategory=HubTest. This message setting tells WMQI Enabler to use the HubTest Message Profile to determine what kind processing the message supposed to go through. The HubTest message profile calls out a WorkflowProcessName of SetDestinationIDM. The SetDestinationIDM workflow uses the contents of the message profile to change the message destinationLogicalId to the system symbolic noted in the interaction portion of message profile. For HubTest, that destination symbolic is BESystem.

The resulting request message is illustrated below:

Sample request XML messages are located on the WMQI Enabler Product CD and discussed in the ***Model Office Reference Manual*** *in Chapter 4.*

```
<?xml version="1.0" encoding="UTF-8"?><Message id="M5441920"
version="1.4" bodyType="OAG" timeStampCreated="2000-10-22-08.00.00"
sourceLogicalId="FrontEnd" authenticationId="USERID"
crfCmdMode="alwaysRespond">
    <CrfActionGroup bodyCategory="HubTest" crfPublish="false"
crfCmdMode="alwaysRespond" >
    </CrfActionGroup>
    <COMMAND>
        <HubTest id="CMD1" cmdType="request" cmdMode="alwaysRespond"
echoBack="false">
        </HubTest>
    </COMMAND>
</Message>
```

Build a response message

The contents of request/response messages are, by definition, message specific. For this example, a simple message, HubTest, is sent into the hub by system symbolic FrontEnd. The COMMAND section of the HubTest message is a simple placeholder.

Sample response XML messages are located on the WMQI Enabler Product CD and discussed in the ***Model Office Reference Manual*** *in Chapter 4.*

The HubTest message includes a bodyCategory=HubTest. This message setting instructs the Hub to use the HubTest Message Profile to determine what processing the message is subjected to. The HubTest message profile calls out a WorkflowProcessName of SetDestinationIDM. This workflow process sends the request message to the appropriate back-end, in this case BESYSTEM. BESYSTEM responds with a response message of the HubTest message. SetDestinationIDM uses the cmdStatus value in the response message to direct workflow processing. In either the **ok** or **notOk** settings, SetDestinationIDM routes the response message into MQWF_OUT with ProcessReply=True in the response container. ProcessReply=True tells the Hub to route the response back to the originating system symbolic with the MQSeries correlationId set to the original request message's MQSeries messageId.

The resulting response message is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?><Message id="M5551921"
version="1.4" bodyType="IAA-XML" timeStampCreated="2000-12-14
14:52:38.937" sourceLogicalId="BESystem" authenticationId="USERID"
crfPublish="false" crfCmdMode="alwaysRespond">
     <CrfActionGroup bodyCategory="HubTest" crfPublish="false"
crfCmdMode="alwaysRespond" >
     </CrfActionGroup>
     <COMMAND>
        <HubTest id="CMDRSP1" cmdType="response" refidRequest="CMDRSP1"
refidMsg="M5441920" cmdStatus="ok">
        </HubTest>
    </COMMAND>
</Message>
```

Create a TestSuite

The TestSuite name is **ExampleTestSuite.xml**. The TestSuite is placed in MQTester's home directory, which is assumed to **C:\tester** in this example.

Sample TestSuites may be found on the WMQI Enabler Product CD.

Again, the simplest way to generate a TestSuite is to copy an existing suite and modify it to fit your needs. By convention, the folders used to hold requests and responses mirror the system symbolics sending/receiving those messages. **HubTest** uses a requesting symbolic **FrontEnd** and a response symbolic **BESystem**.

Chapter 4 in the ***Model Office Reference Manual*** *documents the use of MQTester.*

TestSuites should be written to allow for their execution without any residual from any previous testsuites affecting results. For example, **HubTest** does not require sessionValidation. A previous TestSuite may have executed a Logon. To ensure that an open session is closed, ExampleTestSuite includes a Logoff as its first request message. SDR definitions, MessageProfile, and SystemRestart messages created in the previous sections follow the Logoff. Once the correct Setup messages have been executed, the HubTest request message follows.

HubTest's path through the WMQI Enabler is as follows:

1. MQTester places the HubTest request message onto the HUB_IN message queue.
2. WMQI Enabler uses the HubTest message profile data to route the message to MQSeries Workflow.
3. MQSeries Workflow executes the SetDestinationIDM Process.
4. MQSeries Workflow places its output on the MQWF_OUT message queue.
5. WMQI Enabler alters the destinationLogicalId to BESystem (parameters provided by the SetDestinationIDM workflow).

The process flows of MQTester are found in Chapter 4 of the ***Model Office Reference Manual****.*

6. WMQI Enabler routes messages to BESYSTEM (the message queue noted in BESYSTEM's SDR entry).
7. MQTester validates the HubTest request message.
8. MQTester places a HubTest response Message onto the HUB_RWF_IN message queue, after first setting the appropriate correlation information.
9. WMQI Enabler uses information in the WFCO tables to extract cmdStatus from the HubTest response message and insert that data into the workflow ActivityImplInvokeResponse container.
10. WMQI Enabler routes the message back to MQSeries Workflow.
11. MQSeries Workflow continues its execution of the SetDestinationIDM Process.
12. MQSeries Workflow places its output on the MQWF_OUT MQ message queue.
13. WMQI Enabler uses the ProcessReply entry in the ActivityImplInvoke container to route the message to the originating System (FrontEnd). FEIN is used as the destination queue based FrontEnd's SDR entry. The correlationID of the message placed on the FEIN message queue matches the original HubTest request message "messageID".
14. MQTester validates the HubTest response message.

The resulting ExampleTestSuite.xml is illustrated below:

```
<?xml version='1.0'?>
<!DOCTYPE TestSuite SYSTEM "TestSuite.dtd" >
<!-- Created by IBM's XMLGenerator -->
<TestSuite TestSuiteName="Example">
<UseCaseGroup UseCaseGroupName="Example" SpecialCharacters="##">
<ResultPath>c:\tester\Example\FrontEnd\</ResultPath>
<!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->
<MQSender SenderName="FrontEnd">
<UserId>USERID</UserId>
<RequestQMgr>MQSIQM</RequestQMgr>
<RequestQ>HUB_IN</RequestQ>
<ResponseQMgr>MQSIQM</ResponseQMgr>
<ResponseQ>FEIN</ResponseQ>
<RequestPath>
  <Path>c:\tester\Example\FrontEnd\Request\</Path>
  <File>Logoff.xml</File>
  <File>UMPHubTest.xml</File>
  <File>UpdateSDREntryFE.xml</File>
  <File>UpdateSDREntryBESystem.xml</File>
  <File>SystemRestartFE.xml</File>
  <File>SystemRestartBESystem.xml</File>
  <File>HubTest.xml</File>
</RequestPath>
 <ResponseActualPath Verify="No">
  c:\tester\Example\FrontEnd\responseActual
 </ResponseActualPath>
 <ResponseComparePath Compare="Yes">
  c:\tester\Example\FrontEnd\responseCompare
 </ResponseComparePath>
</MQSender>
<!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->
 <MQReceiver ReceiverName="BESystem">
  <RequestQMgr>MQSIQM</RequestQMgr>
  <RequestQ>BESYSTEM</RequestQ>
  <ResponseQMgr>MQSIQM</ResponseQMgr>
  <ResponseQ>HUB_RWF_IN</ResponseQ>
  <RequestActualPath Verify="No">
   c:\tester\Example\BESystem\requestActual
  </RequestActualPath>
  <RequestComparePath Compare="No">
   c:\tester\Example\BESystem\requestCompare
  </RequestComparePath>
  <ResponsePath>
   c:\tester\Example\BESystem\response
  </ResponsePath>
  </MQReceiver>
 </UseCaseGroup>
</TestSuite>
```

Message debugging/execution process

Define any new MQSeries entities

The SDR entry **BackEnd** makes use of queue **BESYSTEM**. That queue is not created as part of the application TestDrive.

Use the **MQSeries Explorer** tool to create the queue, **BESYSTEM**, on queue manager **MQSIQM**.

Start MQTester

This example assumes MQTester is installed in **C:\tester**.

1. Go to the **C:\tester** directory.
2. Execute the file **MQtester.bat**
3. On the click on the **File** menu and select **Open**
   or click the **Open icon** to load a TestSuite.
4. Locate and Open the TestSuite file ExampleTestSuite.xml.

Debug message profile message

Execute **ExampleTestSuite.xml** in MQTester. Correct any errors in the **UMPHubTest.xml** test case. Potential problems may include malformed XML, invalid message contents, write protected log files, and incorrect values in MQTester or missing directories in the file system.

UMPHubTest is a HUBONLYOFFLINE message. WMQI Enabler processes this message and responds to the requesting system (FrontEnd). The UMPHubTest message processed out of WMQI Enabler is placed into the **FrontEnd/responseActual** directory.

The ExampleTestSuite has its compare setting set to **Yes**. Since no UMPHubTest responseCompare file exists, no comparison can be performed.

Typically, a new test case's response is known in advance and can be built prior to test execution. In this example, WMQI Enabler responseActuals will be used as the starting point for building the responseCompare files.

The UMPHubTest responseActual file is changed to include a ## before the authenticationId. This indicator, "##", allows the message to be utilized in other TestSuites that may use a different authenticationId as the requester.

The resulting UMPHubTest responseCompare file is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE Message SYSTEM &amp;quot;MQSFSE_2001.dtd&amp;quot;-->
<Message id="M5551920" version="1.4" bodyType="HUBONLYOFFLINE"
timeStampCreated="2000-31-Aug" timeStampExpired=""
sourceLogicalId="FrontEnd" authenticationId="##SysAdmin"
crfPublish="true" crfCmdMode="alwaysRespond" txnScope="all">
    <COMMAND cmdType="UpdateMessageProfile">
        <MessageTypeName>HubTest</MessageTypeName>
        <MQSISessionValidationFlag>False</MQSISessionValidationFlag>
        <MQSIMessageSequenceValidationFlag>False</MQSIMessageSequenceV
alidationFlag>
        <MQSISystemInteractionCheckFlag>False</MQSISystemInteractionCh
eckFlag>
        <WorkFlowManagementFlag>True</WorkFlowManagementFlag>
        <WorkFlowQueueManager>FMCQM</WorkFlowQueueManager>
    <WorkFlowDataStructureName>ProcessTemplateExecute</WorkFlowDataStr
uctureName>
    <WorkFlowProcessName>SetDestinationIDM</WorkFlowProcessName>
        <WorkFlowQueue>FMC.FMCGRP.EXE.XML</WorkFlowQueue>
        <WorkFlowSymbolic>Workflow1</WorkFlowSymbolic>
        <WorkFlowReplyToQueueManager>MQSIQM</WorkFlowReplyToQueueManag
er>
        <WorkFlowReplyToQueue>MQWF_END</WorkFlowReplyToQueue>
        <HubQueueManager>MQSIQM</HubQueueManager>
        <DefaultDestinationSymbolic>WorkFlowDefault</DefaultDestinatio
nSymbolic>
        <MessageTypeDependency/>
        <TraceFlag>False</TraceFlag>
    <SystemInteractionList>
         <Item>
             <SystemSymbolic>BESystem</SystemSymbolic>
             <MessageTypeName>HubTest</MessageTypeName>
             <RequiredInteractionFlag>True</RequiredInteractionFlag>
             <SystemBackup/>
         </Item>
    </SystemInteractionList>
     <Result>Success</Result></COMMAND>
<!--filename=UMPHubTest.xml--></Message>
```

Debug back-end system SDR message

The UpdateSDREntryBESystem responseActual file is changed to include a ## before the authenticationId. This indicator, "##", allows the message to be utilized in other TestSuites that may use a different authenticationId as the requester.

The resulting UpdateSDREntryBESystem responseCompare file is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE Message SYSTEM &amp;quot;MQSFSE_2001.dtd&amp;quot;-->
<Message id="M5551920" version="1.4" bodyType="HUBONLYOFFLINE"
timeStampCreated="2000-31-Aug" timeStampExpired=""
sourceLogicalId="BESystem" authenticationId="##SysAdmin"
crfPublish="true" crfCmdMode="alwaysRespond" txnScope="all">
    <COMMAND cmdType="UpdateSDREntry">
         <SystemSymbolic>BESystem</SystemSymbolic>
         <Queue>BESYSTEM</Queue>
        <QueueManager>MQSIQM</QueueManager>
    <Result>Success</Result></COMMAND>
<!--filename=UpdateSDREntryBESystem.xml--></Message>
```

Debug back-end SystemRestart message

The RestartBESystem responseActual file is changed to include a ## before the authenticationId. This indicator, "##", allows the message to be utilized in other TestSuites that may use a different authenticationId as the requester.

The resulting RestartBESystem responseCompare file is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE Message SYSTEM &amp;quot;MQSFSE_2001.dtd&amp;quot;-->
<Message id="M5551920" version="1.4" bodyType=" HUBONLYONLINE "
timeStampCreated="2000-31-Aug" timeStampExpired=""
sourceLogicalId="BESystem" authenticationId="##SysAdmin"
crfPublish="true" crfCmdMode="alwaysRespond" txnScope="all">
     <COMMAND cmdType="SystemRestart">
     <Result>Success</Result></COMMAND>
<!--filename=SystemRestartBESystem.xml--></Message>
```

Debug HubTest request message

Execute ExampleTestSuite. Correct any errors in the **HubTest.xml** test case. Potential problems may include malformed XML, invalid message contents, write protected log files, and incorrect values in MQTester or missing directories in the file system.

HubTest is an OAG message. WMQI Enabler processes the message according to the values set in the message profile. The HubTest message processed out of WMQI Enabler is placed on **BESYSTEM** message queue. MQTester places the resulting message into the **BESystem/responseActual** directory. The ExampleTestSuite has its compare setting set to **Yes**. Since no HubTest responseCompare file exists, no comparison can be performed.

Typically, a new test case's response is known in advance and can be built prior to test execution. In this example, WMQI Enabler responseActuals will be used as the starting point for building the responseCompare files.

The HubTest responseActual file is changed to include a ## before the authenticationId. This indicator, "##", allows the message to be utilized in other testsuites that may use a different authenticationId as the requester.

The resulting BESystem/HubTest responseCompare file is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Message id="M5441920" version="1.4" bodyType="OAG"
timeStampCreated="2000-10-22-08.00.00" sourceLogicalId="FrontEnd"
authenticationId="##USERID" crfCmdMode="alwaysRespond"
destinationLogicalId="BESystem">
    <CrfActionGroup bodyCategory="HubTest" crfPublish="false"
crfCmdMode="alwaysRespond" destinationLogicalId="BESystem">
    </CrfActionGroup>
    <COMMAND>
       <HubTest id="CMD1" cmdType="request" cmdMode="alwaysRespond"
echoBack="false">
      </HubTest>
    </COMMAND>
<!--filename=HubTest.xml--></Message>
```

Debug HubTest response message

Execute ExampleTestSuite. Correct any errors in the **HubTest.xml** test case. Potential problems may include malformed XML, invalid message contents, write-protected log files, and incorrect values in MQTester or missing directories in the file system.

HubTest is an OAG message. MQTester, unless instructed otherwise, uses the incoming request name as the response filename. ExampleTestSuite does not override the response filename, so MQTester responds with **BESystem/response/HubTest.xml**.

The HubTest message processed out of WMQI Enabler is placed on the **FEIN** message queue. MQSeries Tester places the resulting message into the **FrontEnd/requestActual**.

The HubTest requestActual file is changed to include a ## before the authenticationId. This indicator, "##", allows the message to be utilized in other TestSuites that may use a different authenticationId as the requester.

The resulting FrontEnd/HubTest responseCompare file is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Message id="M5551921" version="1.4" bodyType="OAG"
timeStampCreated="2000-12-14 14:52:38.937" sourceLogicalId="BESystem"
authenticationId="##USERID" crfPublish="false"
crfCmdMode="alwaysRespond">
     <CrfActionGroup bodyCategory="HubTest" crfPublish="false"
crfCmdMode="alwaysRespond">
     </CrfActionGroup>
     <COMMAND>
         <HubTest id="CMDRSP1" cmdType="response"
refidRequest="CMDRSP1" refidMsg="M5441920" cmdStatus="ok">
         </HubTest>
    </COMMAND>
</Message>
```

Revise TestSuite for ease of use

ExampleTestSuite.xml executes the complete set of test cases each time it is run. While that is no harm in executing all the test cases on each run, there is value in separating the setup test cases from the request/response messages that actually do *work*.

A slightly longer, yet more usable, TestSuite is illustrated below:

```
<?xml version='1.0'?><!DOCTYPE TestSuite SYSTEM "TestSuite.dtd" >
<!-- Created by IBM's XMLGenerator -->
<TestSuite TestSuiteName="Example">
 <UseCaseGroup UseCaseGroupName="ExampleSetup" SpecialCharacters="##">
  <ResultPath>c:\tester\Example\FrontEnd\</ResultPath>
  <!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->
  <MQSender SenderName="FrontEnd">
   <UserId>USERID</UserId>
   <RequestQMgr>MQSIQM</RequestQMgr>
   <RequestQ>HUB_IN</RequestQ>
   <ResponseQMgr>MQSIQM</ResponseQMgr>
   <ResponseQ>FEIN</ResponseQ>
   <RequestPath>
   <Path>c:\tester\Example\FrontEnd\Request\</Path>
   <File>Logoff.xml</File>
   <File>UMPHubTest.xml</File>
   <File>UpdateSDREntryFE.xml</File>
   <File>UpdateSDREntryBESystem.xml</File>
   <File>SystemRestartFE.xml</File>
   <File>SystemRestartBESystem.xml</File>
  </RequestPath>
  <ResponseActualPath Verify="No">
   c:\tester\Example\FrontEnd\responseActual
  </ResponseActualPath>
  <ResponseComparePath Compare="Yes">
   c:\tester\Example\FrontEnd\responseCompare
  </ResponseComparePath>
 </MQSender>
</UseCaseGroup>
<UseCaseGroup UseCaseGroupName="Example" SpecialCharacters="##">
 <ResultPath>c:\tester\Example\FrontEnd\</ResultPath>
 <!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->
 <MQSender SenderName="FrontEnd">
  <UserId>USERID</UserId>
  <RequestQMgr>MQSIQM</RequestQMgr>
  <RequestQ>HUB_IN</RequestQ>
  <ResponseQMgr>MQSIQM</ResponseQMgr>
  <ResponseQ>FEIN</ResponseQ>
  <RequestPath>
   <Path>c:\tester\Example\FrontEnd\Request\</Path>
   <File>HubTest.xml</File>
  </RequestPath>
  <ResponseActualPath Verify="No">
   c:\tester\Example\FrontEnd\responseActual
  </ResponseActualPath>
  <ResponseComparePath Compare="Yes">
   c:\tester\Example\FrontEnd\responseCompare
  </ResponseComparePath>
 </MQSender>
 <!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->
```

```
  <MQReceiver ReceiverName="BESystem">
  <RequestQMgr>MQSIQM</RequestQMgr>
  <RequestQ>BESYSTEM</RequestQ>
  <ResponseQMgr>MQSIQM</ResponseQMgr>
  <ResponseQ>HUB_RWF_IN</ResponseQ>
  <RequestActualPath Verify="No">
    c:\tester\Example\BESystem\requestActual
   </RequestActualPath>
   <RequestComparePath Compare="No">
    c:\tester\Example\BESystem\requestCompare
   </RequestComparePath>
   <ResponsePath>
    c:\tester\Example\BESystem\response
   </ResponsePath>
  </MQReceiver>
 </UseCaseGroup>
</TestSuite>
```

Chapter 4
MQTester

MQTester provides a generic method for processing messages through the WMQI Enabler product to confirm that the associated WMQI Enabler message flows are working as intended. This tool is intended to provide a mechanism to verify the constructed message flows in the WMQI Enabler Model Office and the WMQI Enabler product early in the implementation process, even prior to the final construction of adapter programs.

The MQTester tool simulates an WMQI Enabler environment by creating process threads of two types:

MQSender	Simulates a front-end application, submits a request message, and waits for a response. Optionally, a datagram can be sent and no response will be expected.
MQReceiver	Simulates a back-end application, receives a request, and sends the appropriate response. If a datagram is received, no response is returned.

Different configurations with single or multiple back-end or front-end applications can be simulated and used in conjunction with WMQI Enabler. No actual or real processing is done, but rather the message request is tied to the message response with data mapping where appropriate.

The MQTester is a Java application developed specifically to test the WMQI Enabler product, although it can be configured to operate in a non-WMQI Enabler environment. This testing tool allows users to specify XML-formatted message files that should be sent to and received from the WMQI Enabler product from the perspectives of a user-defined front-end and back-end MQSeries applications. In addition, users can build compare files for comparing selected output values and also have the ability to specify text or attribute values that must be transferred from the request to the response.

Role of MQTester

The MQTester has been constructed to allow the generation of test suites of scenarios to verify the passage of messages from simulated front-end to back-end systems.

Through the construction of TestSuites, the business purpose of the WMQI Enabler product can be thoroughly exemplified. Pre-defined TestSuites are provided with the product that may be exercised to demonstrate the working of the message flow, and, also, provide samples that may serve as guides to the establishment of other TestSuites. Since the establishment of adapter programs is not required in order to use the testing tool, the construction of messages can be leveraged and enhanced with the additional opportunities for message verification.

The MQTester provides a tool that is both generic and flexible in application. Multiple configurations are possible as well as the routing of multiple messages. The use of multiple machines is also provided. MQTester is the product recommended tool to be employed for product installation verification and use in the implementation process.

Installing MQTester

The MQTester tool is installed by performing these steps:

1. Create a directory called **C:\tester** for the MQTester tool.

   (If if you do not wish to install on the **C**: drive see step 3).

2. Unzip the MQTester.zip file supplied on the product CD into the **\tester** directory.

Presently, only a Windows version of the MQTester exists.

This application also requires Java 1.2 runtime or higher.

The Tester tool is shipped with a sample TestSuite folder which will be placed in the **\tester** directory. The Hub sample requires the WMQI Enabler product and the No_Hub sample does not, as it just hooks the front-end to the back-end directly.

The instructions below show how to run the No_Hub sample. The sample TestSuite directories represent a suggested (but not required) directory structure for the MQTester files:

```
\NoHub
        NoHub.log*
        \frontEnd
                \request
                        AddPerson.xml
                \responseActual
                \responseCompare
                        AddPerson.xml
        \backEnd
                \response
                        AddPerson.xml
                \requestActual
                \requestCompare
                        AddPerson.xml
```

Figure 2: Suggested MQTester directory structure.

*NoHub.log is the results file.

3. If you installed in a directory other than **C:\tester**, edit the TestSuite XML file and change all references from **C:\tester** to your drive and directory.

If you run the sample TestSuite files as they are, you will need an MQSeries configuration as follows:

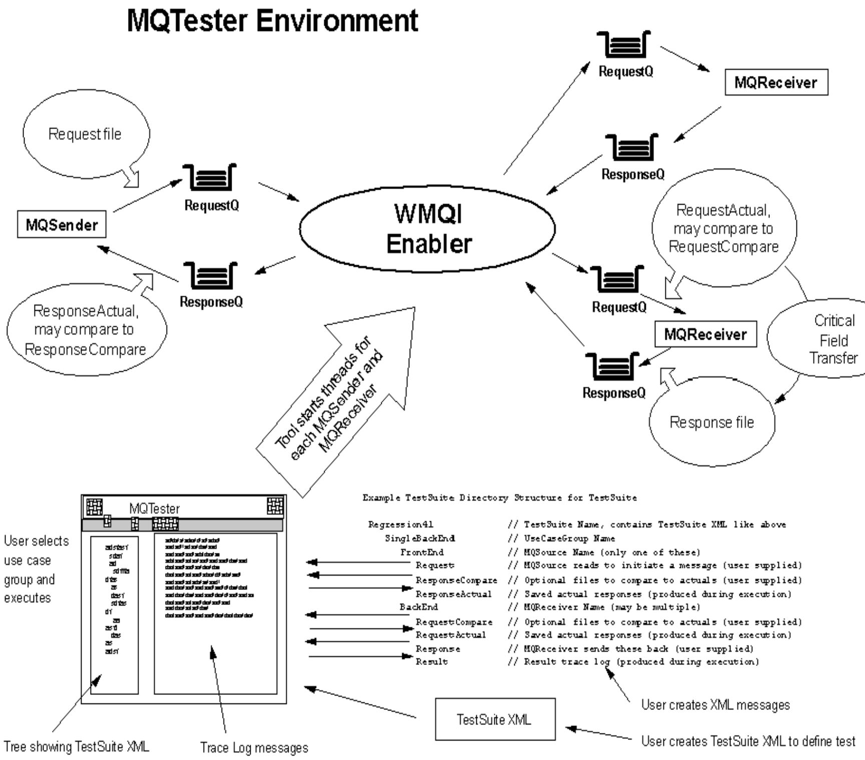
Test Suite	MQ Manager	Queues
Hub	MQSIQM	HUB_IN FEIN BEIN
No_Hub	MQSIQM	FEIN BEIN

Table 1: Sample TestSuite MQSeries configuration.

These are the MQSeries settings defined in the MQSender and MQReceiver definitions. You can change them if you like.

MQTester overview

The execution of the MQTester requires a pre-existing MQSeries environment and files that must be created by the user. The environment below shows how the MQTester tool is used to test the WMQI Enabler product.


MQTester Environment
Request file
MQSender
RequestQ
ResponseQ
ResponseActual, may compare to ResponseCompare
WMQI Enabler
RequestQ
MQReceiver
ResponseQ
RequestActual, may compare to RequestCompare
RequestQ
MQReceiver
Critical Field Transfer
ResponseQ
Response file
Tool starts threads for each MQSender and MQReceiver
MQTester
User selects use case group and executes
Tree showing TestSuite XML
Trace Log messages
TestSuite XML
User creates XML messages
User creates TestSuite XML to define test

```
Example TestSuite Directory Structure for TestSuite

 Regression41            // TestSuite Name, contains TestSuite XML like above
    SingleBackEnd        // UseCaseGroup Name
       FrontEnd          // MQSource Name (only one of these)
          Request        // MQSource reads to initiate a message (user supplied)
          ResponseCompare // Optional files to compare to actuals (user supplied)
          ResponseActual // Saved actual responses (produced during execution)
       BackEnd           // MQReceiver Name (may be multiple)
          RequestCompare // Optional files to compare to actuals (user supplied)
          RequestActual  // Saved actual responses (produced during execution)
          Response       // MQReceiver sends these back (user supplied)
          Result         // Result trace log (produced during execution)
```

Figure 3: MQTester tool overview.

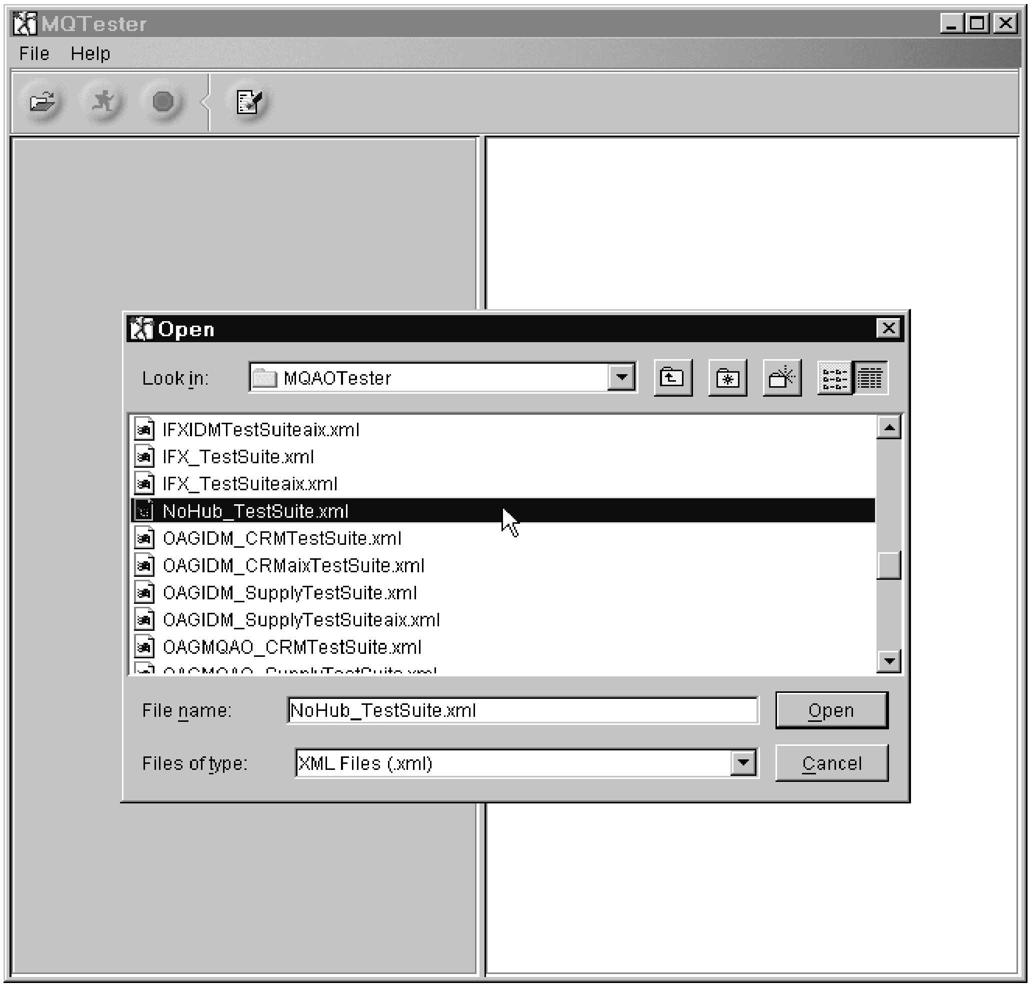
As shown in the figure above, MQTester starts MQSender and MQReceiver threads as defined by a user in the MQTester TestSuite XML File. This process starts after a user selects which Use Case Groups they want to execute.

Using MQTester

1. Start MQTester using the **MQTester.bat** file.

   An icon file, **MQTester.ico**, is provided if you want to make a shortcut to MQTester.bat and assign an icon to the shortcut.

   The application starts and appears as shown in Figure 4:



Look in: MQAOTester

File name: NoHub_TestSuite.xml

Files of type: XML Files (.xml)

Figure 4: Starting MQTester.

MQTester is shipped with seven pre-defined TestSuites:

a. Hub: a simple example of using the WMQI Enabler product with an AddPerson request and the required system messages for initialization.

b. NoHub: a simple example that connects a front-end directly to a back-end and sends an AddPerson request.

c. OAG_CRM: CRM OAG messages and their associated system messages.

d. OAG_SupplySide: Supply Management OAG messages and their associated system messages.

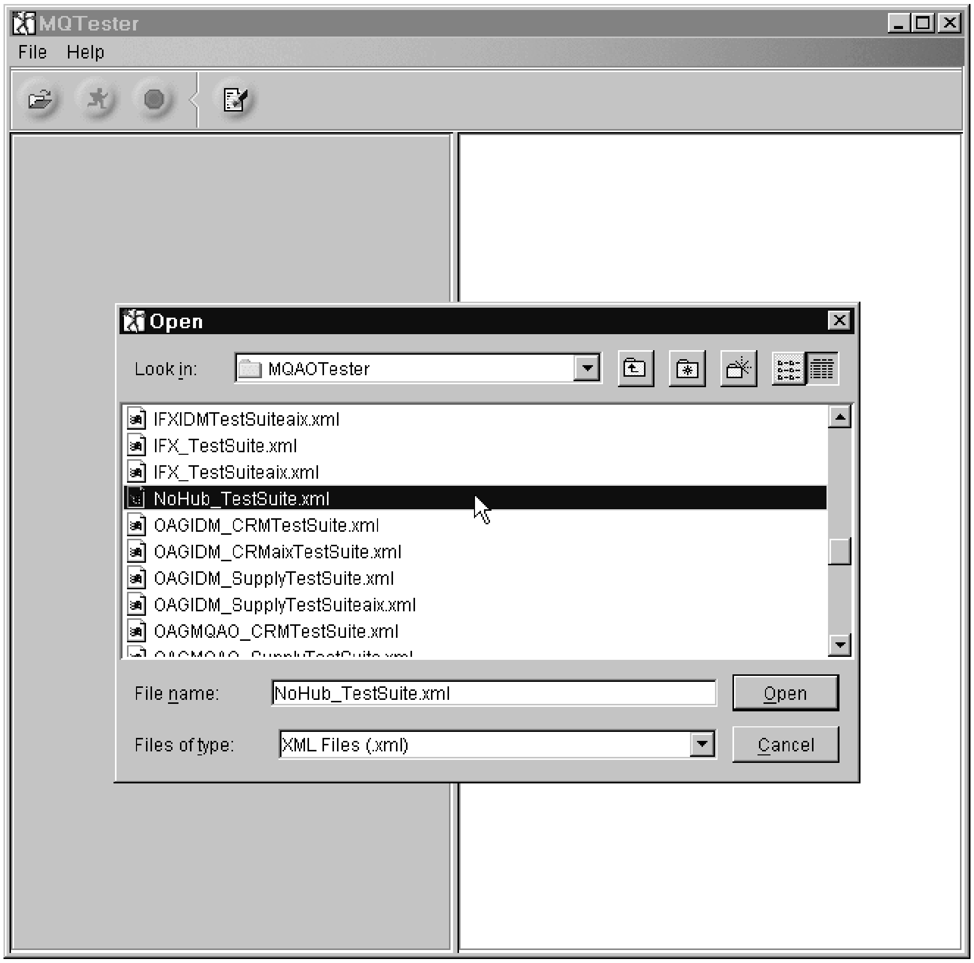
e. OAGIDM-CRM: CRM OAG messages built using IBM's OAG data model.

f. OAGIDM-SupplySide: Supply Management OAG messages built using IBM's OAG data model.

g. Two Backend: is an example of output to two back-end systems.

2. Select the TestSuite XML file, **No_Hub_sampleTS.xml** and click **Open**.

   MQTester displays the TestSuite as a tree collapsed to the UseCaseGroup level on the left as shown in the figure below:



Look in: MQAOTester

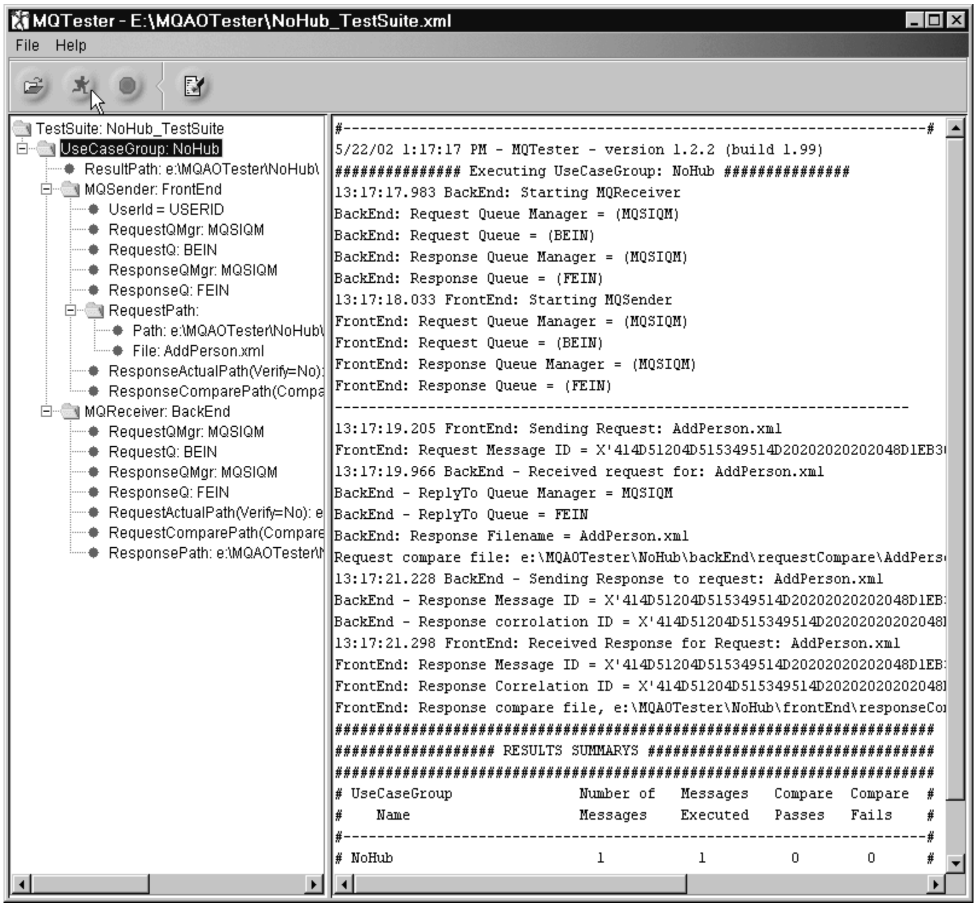
File name: NoHub_TestSuite.xml

Files of type: XML Files (.xml)

Figure 5: Loading a TestSuite in MQTester.

Expand, collapse, and select any items in the tree, however, the only selections that count are **UseCaseGroup** selections. Expanding the tree will show you the configurations of the **UseCaseGroups**.

An example of an expanded tree is shown below:



```
TestSuite: NoHub_TestSuite
  UseCaseGroup: NoHub
    ResultPath: e:\MQAOTester\NoHub\
    MQSender: FrontEnd
      UserId = USERID
      RequestQMgr: MQSIQM
      RequestQ: BEIN
      ResponseQMgr: MQSIQM
      ResponseQ: FEIN
      RequestPath:
        Path: e:\MQAOTester\NoHub\
        File: AddPerson.xml
      ResponseActualPath(Verify=No):
      ResponseComparePath(Compa
    MQReceiver: BackEnd
      RequestQMgr: MQSIQM
      RequestQ: BEIN
      ResponseQMgr: MQSIQM
      ResponseQ: FEIN
      RequestActualPath(Verify=No): e
      RequestComparePath(Compare
      ResponsePath: e:\MQAOTester\N
```

```
#---------------------------------------------------------------------#
5/22/02 1:17:17 PM - MQTester - version 1.2.2 (build 1.99)
############### Executing UseCaseGroup: NoHub ###############
13:17:17.983 BackEnd: Starting MQReceiver
BackEnd: Request Queue Manager = (MQSIQM)
BackEnd: Request Queue = (BEIN)
BackEnd: Response Queue Manager = (MQSIQM)
BackEnd: Response Queue = (FEIN)
13:17:18.033 FrontEnd: Starting MQSender
FrontEnd: Request Queue Manager = (MQSIQM)
FrontEnd: Request Queue = (BEIN)
FrontEnd: Response Queue Manager = (MQSIQM)
FrontEnd: Response Queue = (FEIN)
--------------------------------------------------------------------
13:17:19.205 FrontEnd: Sending Request: AddPerson.xml
FrontEnd: Request Message ID = X'414D51204D515349514D20202020202048D1EB3
13:17:19.966 BackEnd - Received request for: AddPerson.xml
BackEnd - ReplyTo Queue Manager = MQSIQM
BackEnd - ReplyTo Queue = FEIN
BackEnd: Response Filename = AddPerson.xml
Request compare file: e:\MQAOTester\NoHub\backEnd\requestCompare\AddPers
13:17:21.228 BackEnd - Sending Response to request: AddPerson.xml
BackEnd - Response Message ID = X'414D51204D515349514D20202020202048D1EB
BackEnd - Response corrolation ID = X'414D51204D515349514D20202020202048
13:17:21.298 FrontEnd: Received Response for Request: AddPerson.xml
FrontEnd: Response Message ID = X'414D51204D515349514D20202020202048D1EB
FrontEnd: Response Correlation ID = X'414D51204D515349514D20202020202048
FrontEnd: Response compare file, e:\MQAOTester\NoHub\frontEnd\responseCo
#######################################################################
################### RESULTS SUMMARYS ###################################
#######################################################################
# UseCaseGroup               Number of   Messages   Compare  Compare  #
#    Name                    Messages    Executed   Passes   Fails    #
#---------------------------------------------------------------------#
# NoHub                        1           1          0        0      #
```

Figure 6: Configuring UseCaseGroups in MQTester.

3. Select the desired use case groups (in this case there is only one).
4. Then execute them using the **go** button (green man running).

   Test results are displayed in the text area on the right and written to the **ResultsPath** directory.

   The results text area can be cleared by pressing the **delete** button (eraser head on paper).

   To load and run additional MQTester TestSuite files, select the **open** icon from the tool bar or the File menu.

   A Result file is written to the **\No_Hub** directory named **No_Hub.log**, which has the same messages as displayed to the screen.

Configuration possibilities

The MQTester TestSuite XML files can be configured in several ways with the following restrictions:

Multiple MQSenders/MQReceivers

The senders and receivers must not share target path definitions or input queue definitions.

MQSender Only

The back-end that is used must put the MQMD header messageId field from the request into the MQMD header correlationId of the response.

This configuration supports development of back-end applications. The back-end must place the incoming MQMD messageId into the MQMD correlId in its response message in order to allow the MQSender to correctly identify the response. The incoming message will contain a comment that communicates the filename used to generate the request. How or if that filename is used depends on the back-end system.

MQReceiver Only

For an MQReceiver to correlate the incoming request with the desired response, the request XML must contain a comment to communicate the filename that should be used to compare and respond to the request. The comment must be in the exact form as the example below for file **AddPerson.xml**:

```
        <!--filename=AddPerson.xml-->
```

This configuration supports the development of front-end applications. The front-end must place a filename comment into its message in order to supply the MQReceiver with the information it needs to compare and respond correctly to the front-end’s request message. Then MQReceiver will place the request’s MQMD header messageId into the MQMD correlId of its response.

NOTE *comments may not come before the XML declaration, which must be the first item in the message.*

Multiple MQTester Processes

Multiple MQTester processes can be used if not relying on the same target paths or input queues. Also, results will contain only results for MQSenders and MQReceivers in that process.

Multiple Machine MQTester Processes

Same as above.

How to set up an adapter test environment

- Decide if your adapter will operate as a sender or receiver.
- Identify the need for a new queue.
- Create queue(s) as needed.
- Configure MQTester to operate in the mode(s) that your adapter requires.
- If your adapter test environment utilizes MQTester, ensure your adapter conforms to the restrictions noted in the *Configuration possibilities* section.
- Generate adapter messages.
- Generate MQTester messages.
- Configure WMQI Enabler as needed to support the adapter/MQTester test messages.
- Perform end-to-end testing.

How to develop an MQTester TestSuite

MQTester is a very generic and flexible tool. Prior to using MQTester, the user must prepare a configuration file, and the associated test message files. The primary file that drives MQTester is the configuration file known as the MQTester TestSuite file. This TestSuite file contains XML, which conforms to the **TestSuite.dtd**. Before going through the elements of this file, it is necessary to understand some terms used in the MQTester environment that relate to user-developed files.

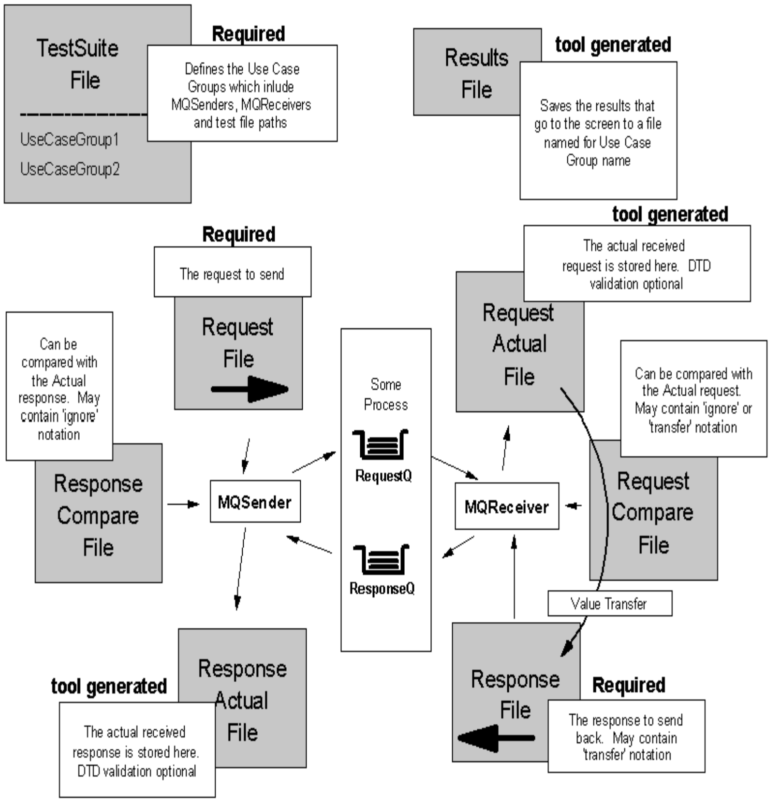

TestSuite File
UseCaseGroup1
UseCaseGroup2
Required
Defines the Use Case Groups which inlude MQSenders, MQReceivers and test file paths
Results File
tool generated
Saves the results that go to the screen to a file named for Use Case Group name
tool generated
The actual received request is stored here. DTD validation optional
Required
The request to send
Request File
Request Actual File
Can be compared with the Actual response. May contain 'ignore' notation
Some Process
RequestQ
ResponseQ
Can be compared with the Actual request. May contain 'ignore' or 'transfer' notation
Response Compare File
MQSender
MQReceiver
Request Compare File
Value Transfer
Response Actual File
tool generated
The actual received response is stored here. DTD validation optional
Response File
Required
The response to send back. May contain 'transfer' notation

Figure 7: Developing a TestSuite for MQTester.

TestSuite file

The TestSuite file is the test configuration file for the MQTester tool. When the tool is started, a TestSuite file must be opened. After testing Use Case Groups in one TestSuite file, other TestSuite files can be opened. A TestSuite file is an XML file that conforms to the **TestSuite.dtd** and may be given any valid filename.

Refer to the sample TestSuite file **Hub_SampleTS.xml** below.

```
<?xml version='1.0'?>
<!DOCTYPE TestSuite SYSTEM "TestSuite.dtd" >
<!-- Created by IBM's XMLGenerator -->
<TestSuite TestSuiteName="Hub_sampleTS">
<UseCaseGroup UseCaseGroupName="Hub"    SpecialCharacters="##">
    <ResultPath>D:\tester\Hub\</ResultPath>
    <MQSender SenderName="FrontEnd">
            <UserId>USERID</UserId>
      <AuthenticationId>USERID</AuthenticationId>
      <RequestQMgr>MQSIQM</RequestQMgr>
      <RequestQ>HUB_IN</RequestQ>
      <ResponseQMgr>MQSIQM</ResponseQMgr>
      <ResponseQ>FEIN</ResponseQ>
      <RequestPath>
<Path>D:\tester\Hub\frontEnd\request</Path>
        <File>AddPerson.xml</File>
      </RequestPath>
      <ResponseActualPath Verify="No">
        D:\tester\Hub\frontEnd\responseActual
      </ResponseActualPath>
      <ResponseComparePath Compare="Yes">
        D:\tester\Hub\frontEnd\responseCompare
      </ResponseComparePath>
    </MQSender>
    <MQReceiver ReceiverName="BackEnd">
      <RequestQMgr>MQSIQM</RequestQMgr>
      <RequestQ>BEIN</RequestQ>
      <ResponseQMgr>MQSIQM</ResponseQMgr>
      <ResponseQ>HUB_IN</ResponseQ>
      <RequestActualPath Verify="No">
        D:\tester\Hub\backEnd\requestActual
      </RequestActualPath>
      <RequestComparePath Compare="Yes">
        D:\tester\Hub\backEnd\requestCompare
      </RequestComparePath>
      <ResponsePath>
        D:\tester\Hub\backEnd\response
      </ResponsePath>
    </MQReceiver>
  </UseCaseGroup>
</TestSuite>
```

TestSuite tag

The structure of the TestSuite file includes some number of <*UseCaseGroup(s)*>, which contain a specification for some number of <*MQSender(s)*> and <*MQReceiver(s)*>, which in turn contain specifications for MQSeries queues and file paths.

Attributes

TestSuiteName (required)

A user defined name given to the *TestSuite*.

Child Elements

UseCaseGroup (required, one or more)

UseCaseGroup tag

This refers to a grouping of use case test messages and a specification of the MQSender and MQReceiver environment which those messages require.

The UseCaseGroup code sample above, *Hub*, executes the AddParty use case in the <*File*> tag. There could have been many <*File*> tags in this UseCaseGroup and they would be executed in the order defined.

Multiple UseCaseGroups can occur within a TestSuite file. After a TestSuite file is loaded into the MQTester tool, only the UseCaseGroups that the user selects will be executed.

The tool starts the MQSenders/Receivers as threads. These threads are stopped at the end of that UseCaseGroup.

Attributes

UseCaseGroupName (required)

A user defined name given to the *UseCaseGroup*. This name will also be the name of the **.log** results file.

SpecialCharacter (optional)

Specifies the characters you want to use for compare and transfer notation. If not specified, the default, ##, is used.

Child Elements

ResultPath (optional)

Specifies a path, without filename, where the result log will be placed. The result log is equivalent to what is displayed during execution.

MQSender (optional, 0 or more)

Describes a message originator needed for this *UseCaseGroup*.

MQReceiver (optional, 0 or more)

Describes a message receiver needed for this *UseCaseGroup*.

MQSender tag

An MQSender initiates a use case message cycle. It gets a request message from the <*RequestPath*> using the <*Path*>, in the order specified by the sequence of <*File*> tags. This path is the only MQTester path specification allowing the user specification of a file name. All other directories specified in this MQSender and its associated MQReceivers must use the same file name given here to allow association. MQSenders will wait for responses to requests that they have sent. MQSender will only get a response off its response queue that has a correlation Id that matches the current request message Id.

Attributes

SenderName (required)

A user defined name given to the *MQSender*.

Child Elements

AuthenticationId (optional)

Specifies a value to be placed in the authenticationId attribute of the message tag of messages originating from this *MQSender*. This tag is required if you want to use the MQTester sessionId propagation feature.

LocalQMgr (optional)

The name of the MQSeries Queue Manager that the MQTester will use as a local interface.

MQMDReplyToQ (optional)

The value that will be placed in the MQSeries MQMD header *replyToQueue* field for the outgoing request message. If not specified, the value in the *ResponseQ* tag will be used.

MQMDReplyToQMgr (optional)

The value that will be placed in the MQSeries MQMD header *replyToQueueManager* field for the outgoing request message. If not specified, the value in the *ResponseQMgr* tag will be used.

RequestPath (required)

Specifies two child elements:

Path (required): the path, without filename, that contains the requests for this MQSender.

File (required, one or more): the filenames, with extensions, in the order of execution.

Datagram Attribute: The File tag has an optional attribute, Datagram, that defaults to "No". Using Datagram = "Yes" will cause the MQSender to send an MQSeries datagram and not wait for a response.

DelayBefore Attribute: The File tag has an optional attribute, DelayBefore, that allow specification of a delay in milliseconds to wait before sending the message. The default is no delay.

DelayAfter Attribute: The File tag has an optional attribute, DelayAfter, that allow specification of a delay in milliseconds to wait after sending the message before sending the next message or terminating the MQSender if this was the last message. The default is no delay.

RequestQ (required)

The name of the MQSeries Queue that this *MQSenders* outgoing requests will be placed on.

RequestQMgr (required)

The name of the MQSeries Queue Manager that owns the queue that this *MQSenders* outgoing requests will be placed on.

ResponseActualPath (optional)

Specified the path where the MQTester will place actual responses that come back to this *MQSender*. Thiselement has an optional attribute, Verify="Yes"/"No". Thedefault is "No" which means a DTD verification will not be done. If Verify is "Yes", the user must supply the required DTD files in the MQTester directory.

ResponseComparePath (optional)

Specified the path where the MQTester will find a user specified XML file to compare with the actual response. This element has an optional attribute, Compare="Yes"/"No". The default is "Yes" which causes a compare against the actual to be done.

ResponseQ (required)

The name of the MQSeries Queue that this *MQSender* will expect to receive incoming responses.

ResponseQMgr (required)

The name of the MQSeries Queue Manager that owns the queue that this *MQSender* will expect to receive incoming responses.

UserId (optional)

The userId that will be placed in the MQSeries MQMD header identity context field, userId, of the outgoing request to identify the originator of the message. If not specified, the userId running the MQTester session will be used.

MQReceiver tag

MQReceiver waits on a request queue, gets a request, and sends back a response to the response queue. MQReceivers read every message that comes to their request queue regardless of correlationId. The MQReceiver correlates the request with compare files and responses by expecting the XML to have a filename comment that was added by the MQSender. If this comment is missing, the MQReceiver will not be able to compare or send a response. In this case, the MQReceiver will save the received request in the <*RequestActualPath*> directory with a filename of Unknown(n).xml. An optional compare capability is provided. In addition, MQReceiver has the ability to move specified fields in the received request to the response.

Attributes

ReceiverName (required)

A user defined name given to the *MQReceiver.*

Child Elements

LocalQMgr (optional)

The name of the MQSeries Queue Manager that the MQTester will use as a local interface.

MQMDReplyToQ (optional)

The value that will be placed in the MQSeries MQMD header *replyToQueue* field for the outgoing response message. If not specified, the value in the *RequestQ* tag will be used.

MQMDReplyToQMgr (optional)

The value that will be placed in the MQSeries MQMD header *replyToQueueManager* field for the outgoing response message. If not specified, the value in the *RequestQMgr* tag will be used.

RequestActualPath (optional)

Specified the path where the MQTester will place actual requests that come to this *MQReceiver*. This element has an optional attribute, Verify="Yes"/"No". The default is "No" which means a DTD verification will not be done. If Verify is "Yes", the user must supply the required DTD files in the MQTester directory.

RequestComparePath (optional)

Specified the path where the MQTester will find a user specified XML file to compare with the actual request. This element has an optional attribute, Compare="Yes"/"No". The default is "Yes" which causes a compare against the actual to be done. Since this file is also used for transferring values, set Compare to "No" if you want to transfer but not compare.

RequestQ (required)

The name of the MQSeries Queue that this MQReceivers incoming requests will be placed on.

RequestQMgr (required)

The name of the *MQSeries* Queue Manager that owns the queue that this *MQReceivers* incoming requests will be placed on.

ResponsePath (required)

Specifies the path, without filename, that contains the responses this MQReceiver will return.

ResponseQ (required)

The name of the MQSeries Queue that this *MQReceiver* will place outgoing responses. If the request message MQMD header contains a replyTo Queue, that response destination is used instead of this one.

ResponseQMgr (required)

The name of the MQSeries Queue Manager that owns the queue that this *MQReceiver* will place outgoing responses. If the request message MQMD header contains a replyTo Queue Manager, that response destination is used instead of this one.

RequestResponseMap (optional)

A new element tag, <RequestResponseMap>, is available for MQReceivers to allow the response filename to differ from the request file name. The MQSender specification of the request file is unchanged.

The use of this element is optional and all exiting TestSuites will still work. If a particular request file does not have a response file mapping, it will be assumed that the response file is named the same as the request file. This element is required for messages such as OAG that have different request/response names

Child Elements:

Map (1 or more)

RequestFile Attribute: specifies the name of the file being sent as a request.

ResponseFile Attribute: specifies the name of the file that the MQReceiver should return as a response to the specified request.

Compare 'Ignore' notation

To enable MQReceiver request comparing or MQSender response comparing, provide the associated <*xxxActualPath*>, the <*xxxComparePath*>, and "known good files". These files must have the same name as the associated request file that began the message cycle. Equality is determined by comparing several things:

Structure	The compared XML documents must have the same element/child element structure.
Attributes	Corresponding elements must have the same number of attributes.
Attribute Values	Corresponding attribute values must match.
Element Text	Corresponding element text values must match (leading and trailing white space is trimmed from the text before comparing).

All other aspects of the document, such as white space, comments, doctype declaration, processing instructions, and name spaces, are ignored.

If there are values in the file that are dynamic and cannot be predetermined, have the compare ignore them. To ignore a value, first characters of the element text or attribute value must be the SpecialCharacters as specified on the UseCaseGroup element, or the default "##" can be used. To ignore, only place these characters at the beginning so as not to confuse them with the “transfer” notation which has the special characters at the beginning and end.

The following examples are valid “ignore” notation when the default special characters are used:

<elementName>##</elementName>

<elementName>##textValue</elementName>

<elementName>########</elementName>

AttributeName="##"

AttributeName="##attributeValue"

AttributeName="######"

Transfer notation

One of the features specific to an MQReceiver is the ability to transfer element text or attribute values from the **RequestActual** to the **Response** file. Notation to indicate this is found in the **RequestCompare** and **Response** files. Values come from the RequestActual. The **RequestCompare** file contains the 'transfer' notation to indicate the location of the value in the RequestActual. For this reason, in order to do transfers, the RequestCompare must mimic the RequestActual in the following ways:

- Navigation from the root to the selected element must be identical in terms of parent/child names. All document structure need not match.
- Repeating elements must be the same in number to find the corresponding element.
- Unlike comparing, only the desired attribute has to exist.

In some cases, it may be to transfer values but not cause a compare. To do this, specify the Compare="No" attribute on the *<RequestComparePath>* tag. Source values in the RequestActual can only be element text or attribute values. When placing that value in the Response, however, it can go anywhere in the file. To indicate a value to transfer, the element text or attribute value must be bracketed by the SpecialCharacters as specified on the UseCaseGroup element, or the default "##". The value can be any unique value (unique among other transfer notations within that RequestCompare file). A target Response file can have multiple occurrences of a single transfer notation. Note that any transfer is also an ignore if a compare is done.

The following examples are valid 'transfer' notation when the default special characters are used:

<elementName>##whatever##</elementName>

AttributeName="##whatever##"

SessionId propagation

If you are issuing a WMQI Enabler logon message and want the returned sessionId to be placed in subsequent messages, you will want to use MQTester sessionId propagation. In order to do this you must place the AuthenticationId tag in your MQSender specification.

This will have the following effect:

- When a WMQI Enabler logon message is issued, it will have that AuthenticationId placed in the logon request.
- The Hub will verify the authenticationId and issue a sessionId in the logon response.
- MQTester will relate this pair together for the rest of your MQTester session or until another logon for the same authenticationId replaces it.
- Subsequent messages issued after the logon will have both the authenticationId and the corresponding sessionId placed into the Message tag attributes.
- If you are also configuring an MQReceiver to receive these subsequent messages, use transfer notation to move the authenticationId and sessionId from the received request to the response as described above.
- If your MQSender is doing compares, you may want to use ignore notation in the **ResponseCompare** file to avoid failing on the sessionId compare.

Using MQTester remotely

Often it is convenient to run the MQTester tool from a machine other than the one with the WMQI Enabler MQ Manager. There are two ways to do this:

- Using the MQ server/client feature.
- Connecting two MQ managers via channels.

Connecting using MQ server/client

The MQTester can connect to a remote MQ manager using the MQ Client feature. This does not require the installation of a complete MQ Series server, only the client software. Below are the instructions to configure MQ client on the WMQI Enabler and MQTester machines.

WMQI Enabler machine (MQManager MQSIQM)

1. Using the MQSeries Explorer, create a Server Connection Channel for MQ Manager, MQSIQM.
2. Note the port that MQSIQM uses for a Listener. (The port can be noted using the MQSeries Services window and examining Listener properties)

3. Using the MQSeries Explorer, start the Server Connection Channel created above

MQTester machine (MQ Client)

1. Install IBM MQSeries Client software
2. Edit tester\lib\MQTester.properties to include the following properties:
   HOSTNAME = <the hostname of the WMQI Enabler machine>
   CHANNEL = <the Server Connection Channel created above>
   PORT = <the Listener port for MQSIQM>

NOTE: If you want to use a local MQ Manager instead of the MQ Client, set the values above to NONE or leave them blank

Connecting two MQ Managers via channels

The environment will resemble the diagram shown in Figure 8:

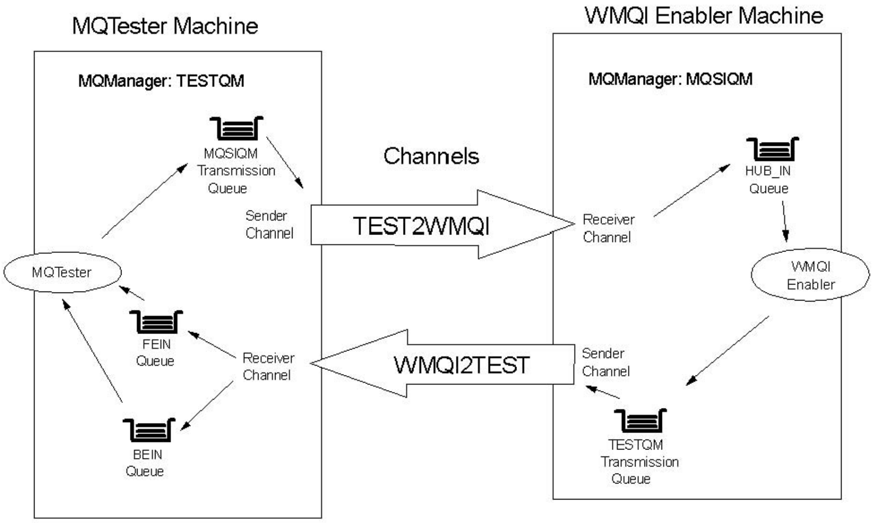


Figure 8: Running MQTester from a remote system

This example assumes MQTester is running under MQManager, **TESTQM**, and WMQI Enabler is running on a different machine under MQManager **MQSIQM**. If the MQTester machine also has an MQ Manager named **MQSIQM**, make sure that it is stopped to avoid a resolution conflict. Using the MQSeries Explorer, create the following objects on the two machines:

MQTester machine (MQManager TESTQM)

1. Create a local queue named **MQSIQM**

   Usage = Transmission
2. Create local queues **FEIN, BEIN**
3. Create sender channel **TEST2MQSFSE**

   Connection Name = IP address of WMQI Enabler Machine

   Transmission Queue = **MQSIQM**

   NOTE: If your MQManager does not use the default port 1414, specify the IP address with a port in the form: **<ip>(<port#>)**
4. Create receiver channel **MQSFSE2TEST**

WMQI Enabler machine (MQManager MQSIQM)

1. Create a local queue named **TESTQM**

   Usage = Transmission
2. Local queue HUB_IN should already exist
3. Create sender channel **MQSFSE2TEST**

   Connection Name = IP address of MQTester Machine

   Transmission Queue = **TESTQM**

   NOTE: If your MQManager does not use the default port 1414, specify the IP address with a port in the form: **<ip>(<port#>)**
4. Create receiver channel **TEST2MQSFSE**

Start the channels

1. Start the sender channel **TEST2MQSFSE** on MQManager **TESTQM**
2. Start the sender channel **MQSFSE2TEST** on MQManager **MQSIQM**

MQTester configuration

When connecting remotely, you must use the **LocalQMgr** tag to indicate the Queue manager is directly accessable by MQTester. Also, as indicated below, an MQSenders ResponseQMgr and ResponseQ and an MQRecivers RequestQMgr and RequestQ must be local.

```
<MQSender SenderName="FrontEnd">
  <UserId>USERID</UserId>
  <LocalQMgr>TESTQM</LocalQMgr>
  <RequestQMgr>MQSIQM</RequestQMgr>
  <RequestQ>HUB_IN</RequestQ>
  <!-ResponseQMgr/Q must be local>
  <ResponseQMgr>TESTQM</ResponseQMgr>
  <ResponseQ>FEIN</ResponseQ>
..............
<MQReceiver ReceiverName="BackEnd">
  <LocalQMgr>TESTQM</LocalQMgr>
  <!-RequestQMgr/Q must be local>
  <RequestQMgr>TESTQM</RequestQMgr>
  <RequestQ>BEIN</RequestQ>
  <ResponseQMgr>MQSIQM</ResponseQMgr>
  <ResponseQ>HUB_IN</ResponseQ>
```

To achieve proper routing, two hub-only messages must be issued to alter the WMQI Enabler Symbolic Destination Resolution (SDR) table.

MQTester UpdateSDREntry commands

WMQI Enabler maintains a Symbolic Destination Resolution (SDR) table to know what MQManager and Queue a system monitors. The UpdateSDREntry commands below will set the SDR table:

```
For FrontEnd:
<COMMAND cmdType="UpdateSDREntry">
   <SystemSymbolic>FrontEnd</SystemSymbolic>
     <Queue>FEIN</Queue>
   <QueueManager>TESTQM</QueueManager>
</COMMAND>
For BackEnd:
<COMMAND cmdType="UpdateSDREntry">
   <SystemSymbolic>BackEnd</SystemSymbolic>
     <Queue>BEIN</Queue>
   <QueueManager>TESTQM</QueueManager>
</COMMAND>
```

Chapter 5
WMQI Enabler - Configurator

WMQI Enabler uses the services of multiple DB2 database tables. Much of the operational characteristics are configurable. Presently, direct database manipulation and HUBONLY messages are used to setup and manage an WMQI Enabler instance. This chapter describes a tool that allows an end user to manage an WMQI Enabler instance.

Overview

The figure below shows the WMQI Enabler - Configurator communicating with an WMQI Enabler installation. The WMQI Enabler - Configurator builds and sends an WMQI Enabler Hubonly XML request message to WMQI Enabler and waits for an WMQI Enabler Hubonly XML response message.

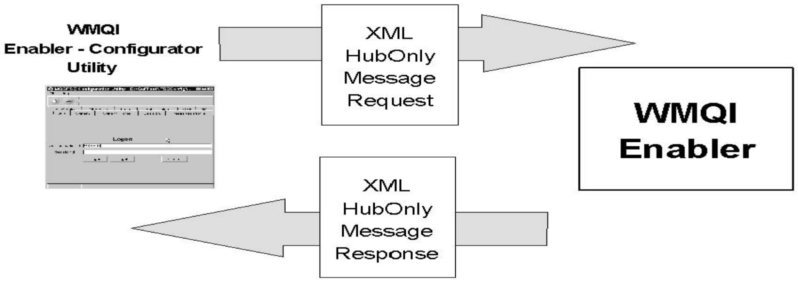


Figure 9: Configuration utility overview.

WMQI Enabler - Configurator supports the following WMQI Enabler HubOnly messages:

- Logon
- Logoff
- SystemRestart
- SystemShutdown
- GetSystemProfile
- UpdateSystemProfile
- KillSession
- KillProcess
- GetMessageProfile
- UpdateMessageProfile
- GetSDREntry
- UpdateSDREntry
- GetNLSEntry
- UpdateNLSEntry
- GetINSEntry
- UpdateINSEntry
- SetSubscription
- CRF

Additional features

In addition to sending and receiving the WMQI Enabler HubOnly messages, the WMQI Enabler - Configurator includes the following features:

- Logon Session ID Propagation: When a user sends the WMQI Enabler Logon command and successfully receives a response with the assigned session ID, that session ID is placed in the WMQI Enabler header of subsequent messages.
- Saved Preferences: The choices a user makes for preferences such as MQSeries and user ID values are saved in a configuration file. The user may choose from multiple, previously saved configuration files.
- XML Message View: The user can select the XML tab to see the last request that was sent and the last response that was received in XML format.

Installation

Prerequisite software

- Microsoft Windows NT v4.0 Service Pack 5 or greater.
- MQSeries v5.2 + SupportPack MA88.
- WebSphere MQ Integrator Enabler v1.2.2 .
- IBM Java Development Kit (JDK) v1.2.2 or greater.

Installation

Installation of the WMQI Enabler - Configurator is typically handled by the WMQI Enabler installation process. After installation you should expect to see the following files in the **/wmqiecfg** directory:

- **default.config**: the default configuration file.
- **wmqiecfg.bat**: a command file for starting the WMQI Enabler - Configurator.
- **wmqiecfg.ico**: an icon that can be associated with the .bat file shortcut.

- **wmqiecfg.jar**: the WMQI Enabler - Configurator Java jar.
- **session.$config**: a system file to save last used configuration.

Configuration

Creating an NT shortcut (optional)

If it is preferred, an NT shortcut can be created to the WMQI Enabler - Configurator as indicated below:

1. Highlight the **wmqiecfg.bat** file
2. **Right** click the highlighted file from the previous step.
3. Choose **Create Shortcut from** the pop-up menu.

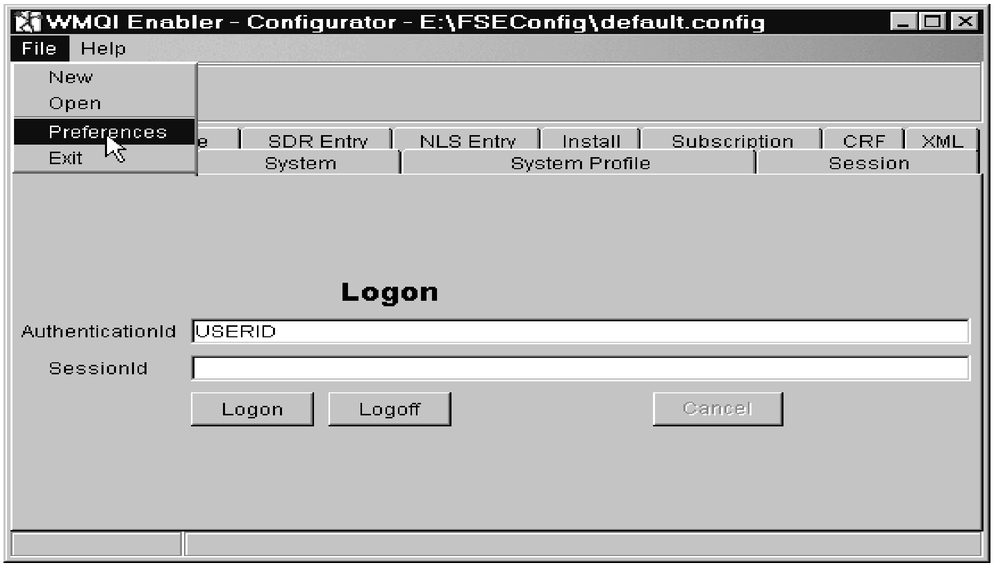
Next you may rename the shortcut if desired and drag to the desktop or to another location. To change the icon of the shortcut, flow the steps below:

1. Highlight the shortcut file.
2. **Right** click on the highlighted file from the previous step.
3. Choose **Properties** from the pop-up menu.
4. Click on the **Shortcut** tab
5. Click on the **Change Icon** button
6. Click on the **Browse** button to locate the file **wmqiecfg.ico**
7. When finished click **OK** twice

WMQI Enabler - Configurator configuration

To configure the WMQI Enabler - Configurator, you must first start it.

1. Run **wmqiecfg.bat**
   (or use the shortcut created in the previous step)
2. From the **File** menu select **Preferences**

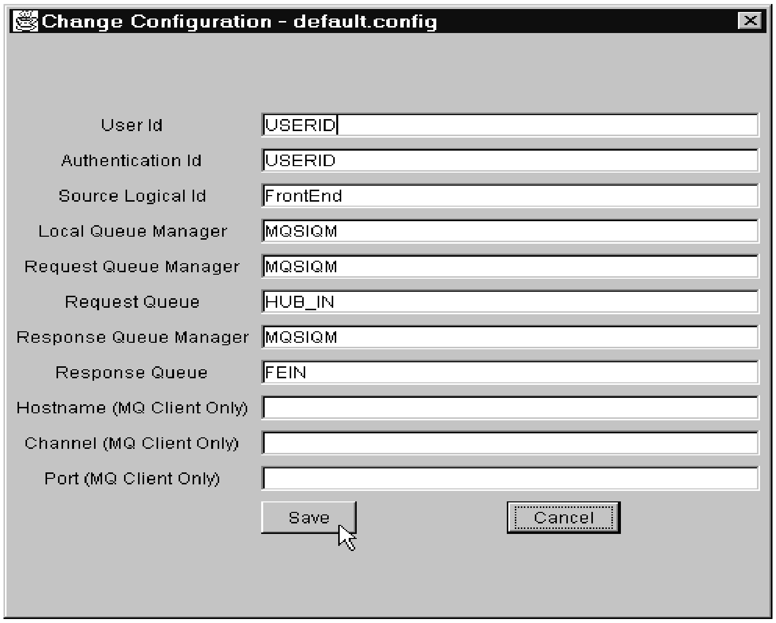


AuthenticationId: USERID

SessionId:

Figure 10: Selecting Preferences in the WMQI Enabler - Configurator

This will allow you to change any of the default fields below:



User Id	USERID
Authentication Id	USERID
Source Logical Id	FrontEnd
Local Queue Manager	MQSIQM
Request Queue Manager	MQSIQM
Request Queue	HUB_IN
Response Queue Manager	MQSIQM
Response Queue	FEIN
Hostname (MQ Client Only)	
Channel (MQ Client Only)	
Port (MQ Client Only)	

Figure 11: Change Configuration screen

Configuration fields

User Id: The User Id that will be used in the MQSeries MQMD header. If blank, the current Windows NT user Id will be used.

Authentication Id: The User Id that will be placed in the WMQI Enabler header.

Source Logical Id: The logical name of the sending system in the WMQI Enabler header.

Local Queue Manager: The MQSeries queue manager accessible by the WMQI Enabler - Configurator.

Request Queue Manager: The MQSeries queue manager of the WMQI Enabler installation.

Request Queue: The request input queue for WMQI Enabler.

Response Queue Manager: The MQSeries queue manager of the queue where the WMQI Enabler - Configurator will read responses.

Response Queue: The MQSeries queue where the WMQI Enabler - Configurator will read responses.

Hostname (MQ Client Only): If using MQSeries Client, put the hostname of the machine that is acting as your MQ Series server.

Channel (MQ Client Only): If using MQSeries Client, put the name of the server connection channel on the host machine.

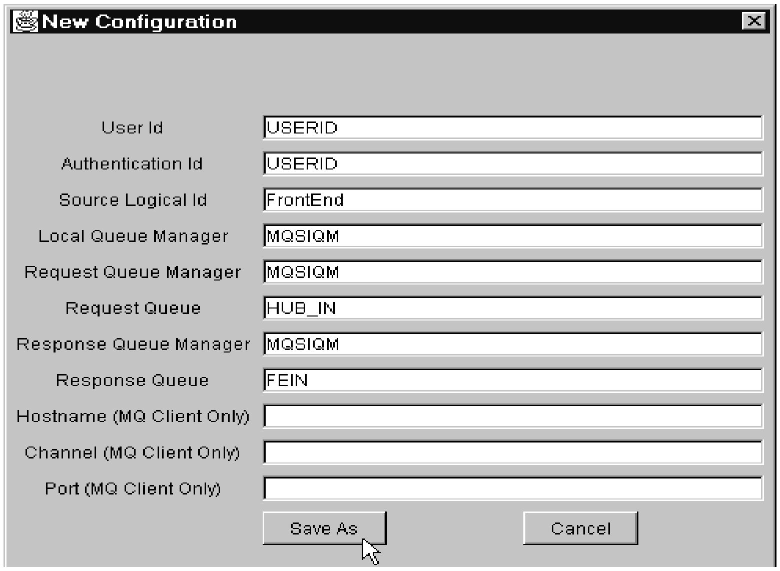
Port (MQ Client Only): If using MQSeries Client, put the port number of the hosting MQSeries manager listener. For example: 1414

To create a new configuration file to use instead of the **default.config**,

1. From the **File** menu, select **New**
   (or click the **New Configuration File** button on the main toolbar)

   You will see the fields defaulted to your last configuration's values as below.

2. Edit the fields as desired, click **Save As**,and name the new configuration file



User Id	USERID
Authentication Id	USERID
Source Logical Id	FrontEnd
Local Queue Manager	MQSIQM
Request Queue Manager	MQSIQM
Request Queue	HUB_IN
Response Queue Manager	MQSIQM
Response Queue	FEIN
Hostname (MQ Client Only)	
Channel (MQ Client Only)	
Port (MQ Client Only)	

Figure 12: New configuration screen.

To choose which configuration is in effect.

1. From the **File** menu, select **Open**
   (or click the **Open Configuration File** button on the main toolbar)

Notice that the path to the current configuration file is displayed in the application title bar.

Sending WMQI Enabler messages

The main view of the MQFSFE Configuration Utility is a series of tabs that group the supported commands into logical areas. The headings below give the details on using each of these command tabs.

Each of the command area tabs share certain function:

Cancel Capability: when a request is sent, the WMQI Enabler - Configurator periodically polls the response queue until a response for that request (based on correlation ID) is received. Pressing the Cancel button will stop that polling.

Status Bar Information: the status bar at the bottom of the application window will give a status about the state of the transaction including error messages.

XML Tab: All messages that are sent and received will log the XML on the XML tab page. Only the last transaction is viewable. This is primarily for problem determination or verification of values not reported to the individual command screens.

Get Update Pattern: the messages that have a get/update pair share the same page. A get can be executed to see the current values, followed by an update after changes have been made to those values.

Using the WMQI Enabler - Configurator

The WMQI Enabler - Configurator user interface consists of one multi-tabbed folder. Each tabbed page is for sending the message or messages that relate to that tab.

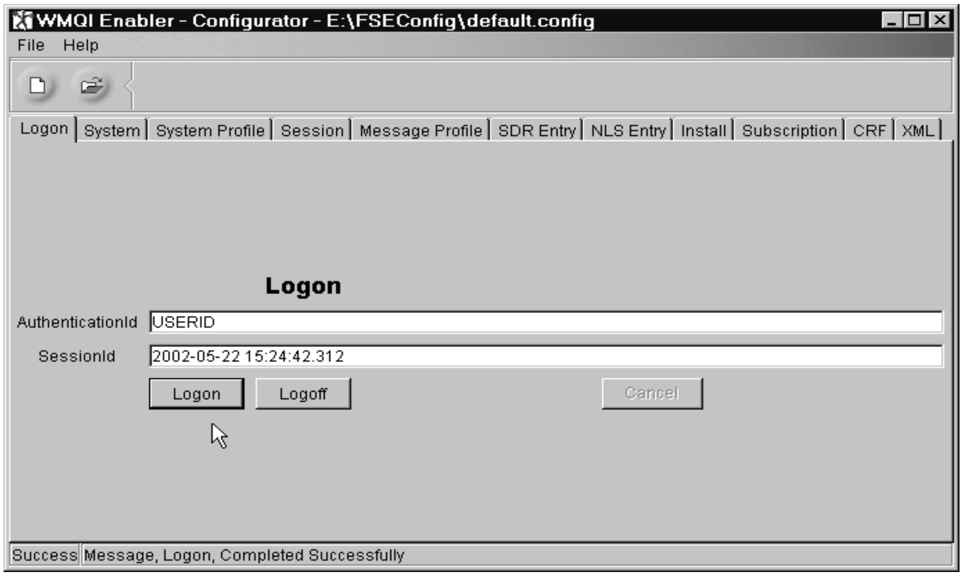
Messages are generally sent by first providing some identifying value, or key, for the requested information. Clicking the **Get** button would send the message to WMQI Enabler. If necessary, the **Cancel** button will interrupt the wait for the response. When the response is received, the received values are displayed.

In the case of Get/Update pair messages, changes can be made to the displayed values and pressing the **Update** button will send the Update message.

The status line at the bottom of the application screen will provide status during your message transactions. If more detail is needed, the XML tab shows the last message sent and received. The sections below show each tab page and give information specific to that page.

The Logon page is displayed first. If your WMQI Enabler installation has been configured to require session management for HubOnly messages, you will need to Logon before issuing other messages. The returned session Id will be placed in the header of subsequent messages that you issue in that session. Remember to logoff if you logon.

Logon tab



AuthenticationId: USERID

SessionId: 2002-05-22 15:24:42.312

Success Message, Logon, Completed Successfully

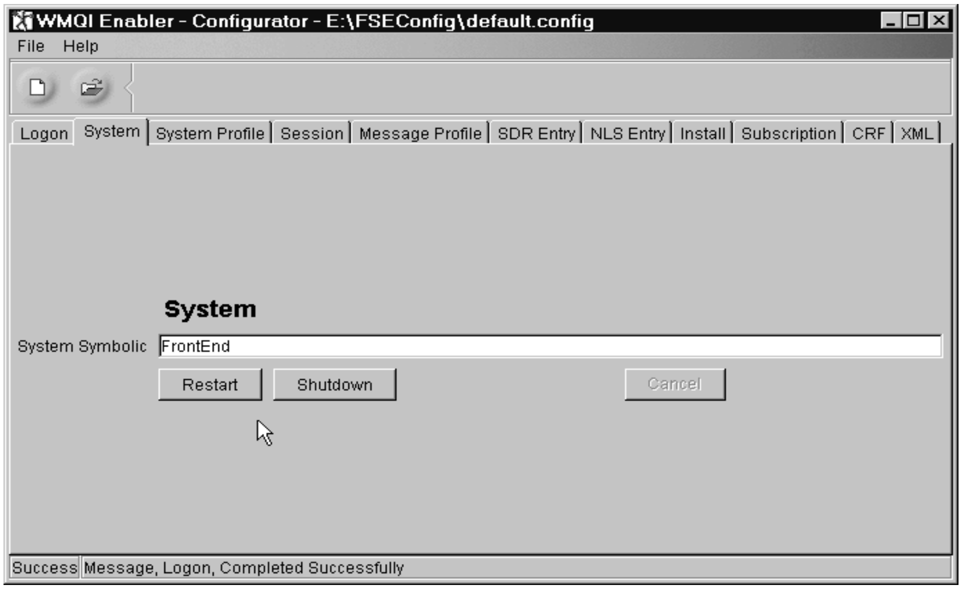
Figure 13: Logon tab.

Use Logon if your WMQI Enabler installation is configured to require session management on HubOnly messages. The returned session ID will be placed in the WMQI Enabler header of subsequent messages. Remember to Logoff when you are finished.

Logon: Enter an AuthenticationId and SessionId and click the **Logon** button to initiate a logon.

Logoff: Enter an AuthenticationId and SessionId and click the **Logoff** button to initiate a logoff.

System tab



System Symbolic: FrontEnd

Success Message, Logon, Completed Successfully

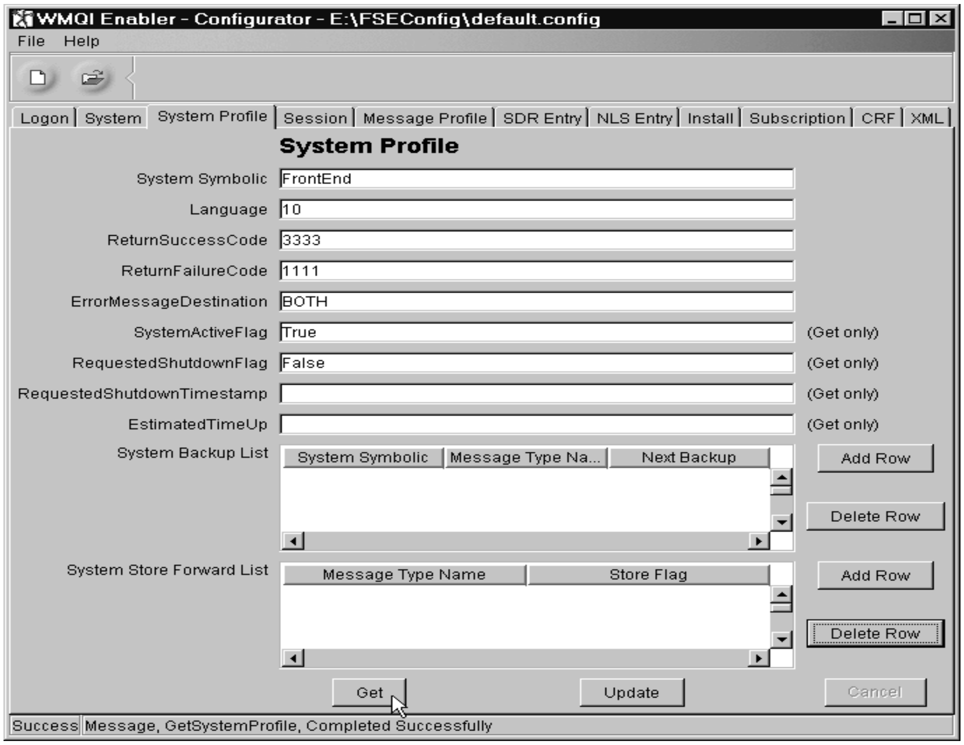
Figure 14: System tab.

Starts the "System Symbolic" system and considers that system ready to send and/or receive message traffic. It also initiates a shutdown of the "System Symbolic" system.

SystemRestart: Enter a System Symbolic and click the **Restart** button to request a system restart.

SystemShutdown: Enter a System Symbolic and click the **Shutdown** button to request a system shutdown.

System profile Tab



System Profile

System Symbolic FrontEnd
Language 10
ReturnSuccessCode 3333
ReturnFailureCode 1111
ErrorMessageDestination BOTH
SystemActiveFlag True (Get only)
RequestedShutdownFlag False (Get only)
RequestedShutdownTimestamp (Get only)
EstimatedTimeUp (Get only)

Success Message, GetSystemProfile, Completed Successfully

Figure 15: System profile tab.

Allows reading and updating of the tables used in system profiles.

GetSystemProfile: Enter a System Symbolic and click the **Get** button to request the Profile for that System Symbolic.

UpdateSystemProfile: Enter a System Symbolic and the parameters defining that system's profile. Click the **Update** button to change the system's profile.

The fields in the system profile tab are explained below:

System Symbolic: The symbolic name for this system must be entered for **Get** or **Update**.

Language: The language code number for this system.
For example, US English = **10**.

System BackUp List: List of systems that can also do the same function as this one.

> **System Symbolic:** The symbolic name for this system. Should be same as at the top of this Tab.

Message Type Name: The message type that would use this backup system if necessary.

Next Backup: The symbolic name for the next backup system in the chain.

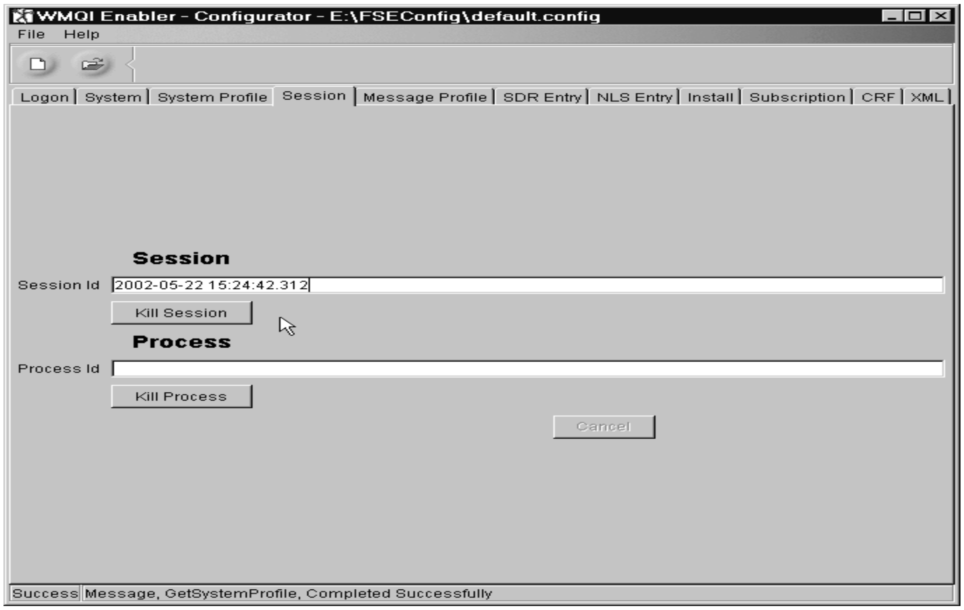
System Store Forward List: Message types destined for this system that should be stored and forwarded later if this system is temporarily unavailable.

Message Type Name: The message type to be forwarded.

Store Flag: Set to **True** if this message type is to be stored and forwarded. Set to **False** otherwise.

The Add Row/Delete Row: For these last two sections, add and delete rows from the text area, respectively.

Session tab



WMQI Enabler - Configurator - E:\FSEConfig\default.config

Session Id: 2002-05-22 15:24:42.312

Process Id:

Success | Message, GetSystemProfile, Completed Successfully

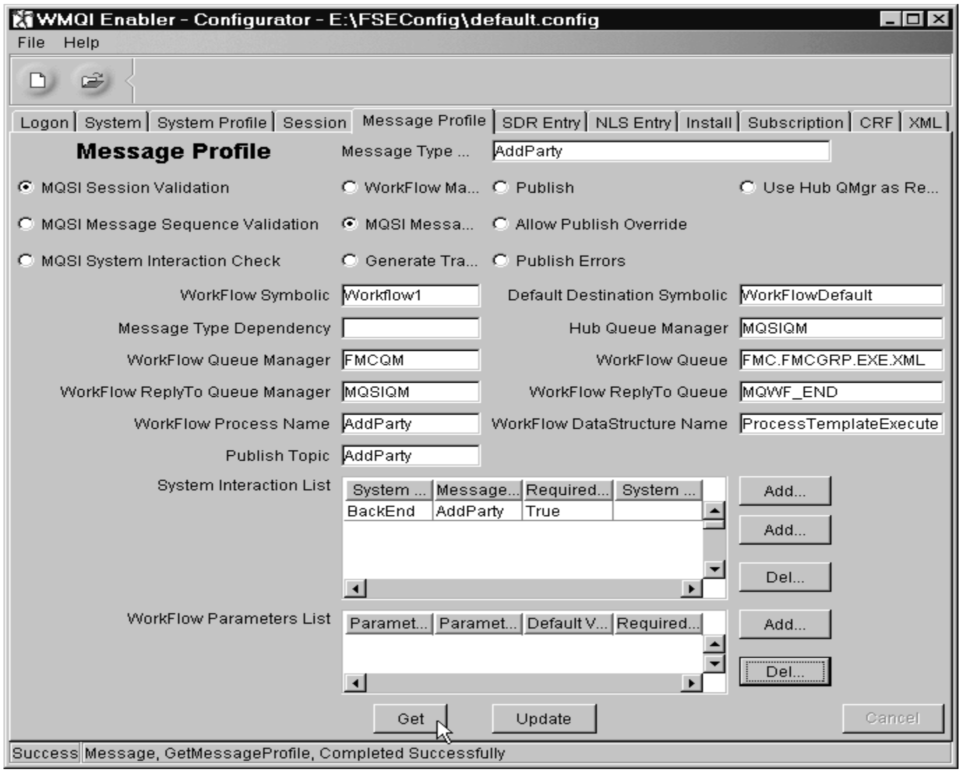
Figure 16: Session tab.

Allows a user to manually terminate a session or process.

KillSession: After entering the Session Id, click the **Kill Session** button to terminate the session.

KillProcess: After entering the Process Id, click the **Kill Process** button to terminate the process.

Message profile Tab



Message Type ... AddParty

● MQSI Session Validation
○ MQSI Message Sequence Validation
○ MQSI System Interaction Check
○ WorkFlow Ma...
● MQSI Messa...
○ Generate Tra...
○ Publish
○ Allow Publish Override
○ Publish Errors
○ Use Hub QMgr as Re...

WorkFlow Symbolic: Workflow1
Message Type Dependency:
WorkFlow Queue Manager: FMCQM
WorkFlow ReplyTo Queue Manager: MQSIQM
WorkFlow Process Name: AddParty
Publish Topic: AddParty
Default Destination Symbolic: WorkFlowDefault
Hub Queue Manager: MQSIQM
WorkFlow Queue: FMC.FMCGRP.EXE.XML
WorkFlow ReplyTo Queue: MQWF_END
WorkFlow DataStructure Name: ProcessTemplateExecute

System Interaction List

System ...	Message...	Required...	System ...
BackEnd	AddParty	True	

WorkFlow Parameters List

Paramet...	Paramet...	Default V...	Required...

Success | Message, GetMessageProfile, Completed Successfully

Figure 17: Message profile tab.

The Message Profile is a configurable table of flags and other values that can be toggled to change the way messages of a specific type are processed through WMQI Enabler. This Tab allows the user to read or update message profiles in the WMQI Enabler Database.

GetMessageProfile: Enter a Message Type Name and click the **Get** button to read all Non-Null fields from the entry in the MESSAGE_PROFILE_TABLE.

UpdateMessageProfile: Enter a Message Type Name and all fields necessary for this Message Type. Click the **Update** button to update that entry in the MESSAGE_PROFILE_TABLE.

The fields in the message profile tab are explained below:

Message Type Name: Required Field. This is determining value for what type of message is flowing through the hub. The Message Type Name determines which message profile is pulled from the MESSAGE_PROFILE_TABLE.

MQSI Session Validation: Select this radio button if you want this message type to utilize session validation.

MQSI Message Sequence Validation: Select this radio button if you want this message type to utilize message sequence validation.

MQSI System Interaction Check: Select this radio button if you want this message type to utilize system interaction checking. This will allow WMQI Enabler to determine whether required systems are available to receive this message.

WorkFlow Management: Select this radio button if you want this message type to utilize a WorkFlow Process template.

MQSI Message Enabled: Select this radio button if you want this message type to be enabled. This will allow this type of message to pass through WMQI Enabler.

Generate Trace: Select this radio button if you want tracing to be turned on when this type of message flows through WMQI Enabler.

Publish: Select this radio button if you want this message type to be published to subscribers.

Allow Publish Override: Select this radio button if you want this message type to published in the case that a field in the message is set to 'publish = **true**' even though the above field radio button is **off**.

Publish Errors: Select this radio button if you want this message type to publish any errors which may occur during processing through WMQI Enabler.

WorkFlow Symbolic: Specifies the SDR symbolic to be used when determining which queue and queue manager to put messages which are destined for MQSeries Workflow.

Default Destination Symbolic: Specifies a default destination symbolic to be used in SDR in case the message does not have a destinationLogicalId.

Message Type Dependency: In the case of using MQSI Message Sequence Validation, this tells WMQI Enabler what message type must occur immediately before the current message type.

Hub Queue Manager: Specifies a default queue manager for WMQI Enabler.

WorkFlow Queue Manager: Specifies a default queue manager for MQSeries Workflow.

WorkFlow Queue: Specifies a default queue for MQ Series Workflow.

WorkFlow Reply To Queue Manager: Specifies a default queue manager for MQSeries Workflow to which to reply.

WorkFlow Reply To Queue: Specifies a default queue for MQSeries Workflow to which to reply.

WorkFlow Process Name: Specifies a default Workflow process template to use for this message type.

WorkFlow Data Structure Name: Specifies a default Workflow data structure to use when building messages destined for MQSeries Workflow.

Publish Topic: Specifies a topic under which this message type will be published if publishing is used.

System Interaction List: Each item is a system that this message will use at some point in processing.

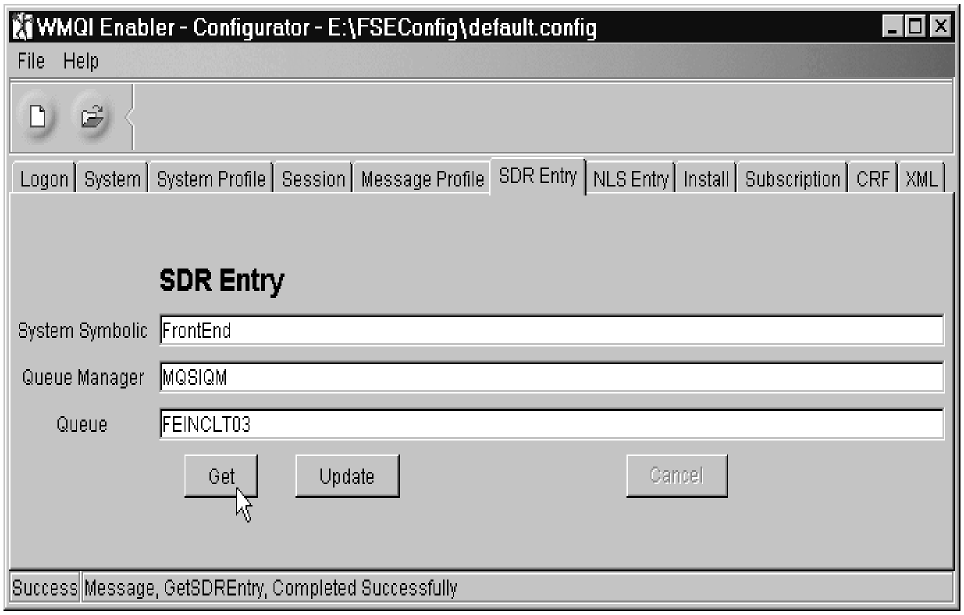
> **System Symbolic:** The symbolic value of the system.
>
> **Message Type Name:** Should coincide with the Message Type Name at the top of the Tab
>
> **Required Interaction Flag: True** if this system is absolutely required for this message type to be processed. **False** otherwise.
>
> **System Backup:** The symbolic value of the next backup system in line if this system is down.

WorkFlow Parameters List: Listing of any extra fields that should be added to the WorkFlow container section for messages utilizing MQSeries Workflow.

> **Parameter Name:** The name of the field as it appears in XML.
>
> **Parameter Path:** The path that should be used to pull this value from the message and put it into the container section.
> Example: **Message.COMMAND.cmdStatus**
>
> **Default Value:** Specifies a value to put in this field if it cannot be found in the path specified.
>
> **Required Parameter Flag: True** specifies that the parameter is required. Code should respond with an error if no value can be found in the path or in the default value field.

The Add Row/Delete Row: For these last two sections add and delete rows from the text area, respectively.

SDR entry tab



System Symbolic: FrontEnd

Queue Manager: MQSIQM

Queue: FEINCLT03

Success Message, GetSDREntry, Completed Successfully

Figure 18: SDR entry tab.

Use this tab to define Queue and Queue Manager by System Symbolic. This table will be used for routing messages destined for a particular system.

GetSDREntry: Enter a valid System Symbolic and click the **Get** button to see what Queue and Queue Manager receive messages for this system.

UpdateSDREntry: Enter a valid System Symbolic, Queue and Queue Manager. Click the **Update** button to see change the entry in the SDR_TABLE.

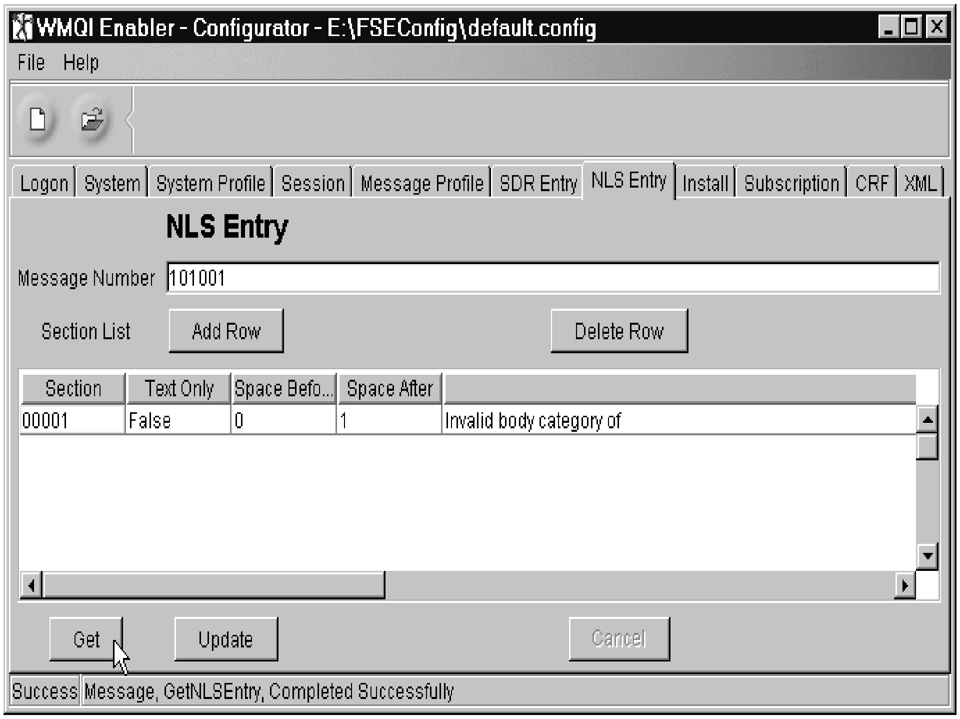
The fields in the SDR entry tab are explained below:

System Symbolic: The symbolic name for the system.

Queue Manager: The name of the queue manager used by this system.

Queue: The queue from which this system reads.

NLS entry Tab



Message Number 101001

Section	Text Only	Space Befo...	Space After	
00001	False	0	1	Invalid body category of

Success Message, GetNLSEntry, Completed Successfully

Figure 19: NLS entry tab.

This tab is used to read and update the Natural Language Support function of WMQI Enabler.

GetNLSEntry: Enter a valid Message Number and click the **Get** button to see the corresponding table entry.

UpdateNLSEntry: Enter a valid Message Number and fill in the required Section List fields. Click **Update** to make changes to the NLS_ERROR_MESSAGE_TABLE.

The fields in the NLS entry tab are explained below:

Section List: Defines a NLS error message by section.

Section: The section number. Should start at 1 and go up incrementally by 1.

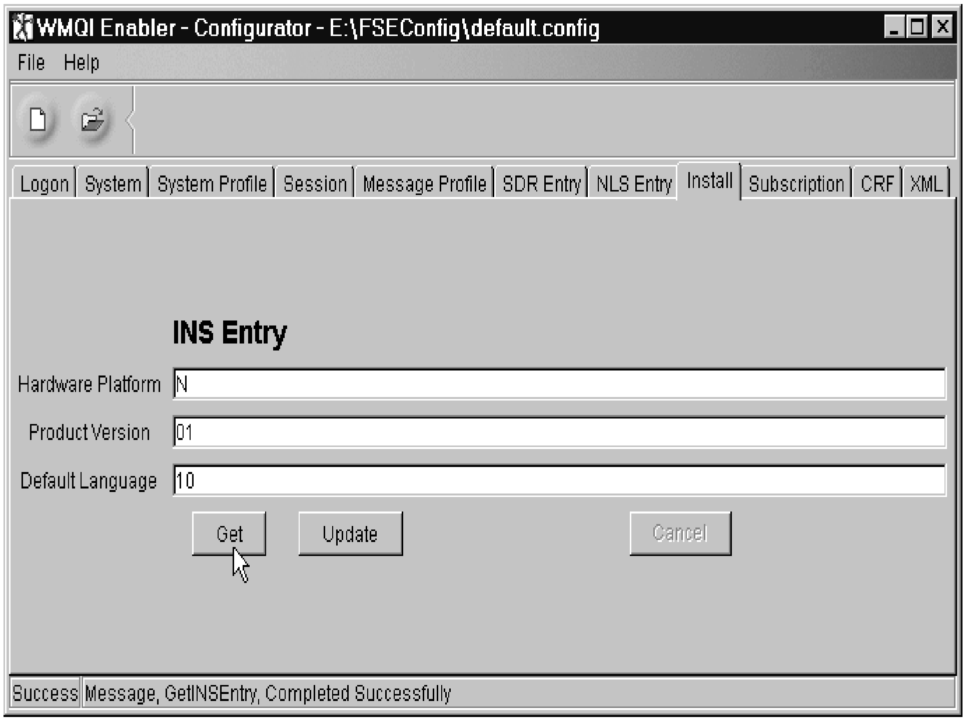
Text Only: True if message is text only. **False** otherwise.

Space before: True if a space is required before this section of the message.

Space after: True if a space is required after this section of the message.

The Add Row/Delete Row: For these last two sections add and delete rows from the text area, respectively.

Install Tab



Hardware Platform: N

Product Version: 01

Default Language: 10

Success Message, GetINSEntry, Completed Successfully

Figure 20: Install tab.

Allows the user to read or update entries in the INSTALL_DATA_TABLE.
GetINSEntry: Click the **Get** button to see the data for your hub installation.

UpdateINSEntry: Enter data in each field and click **Update** to add/change data for your hub installation.

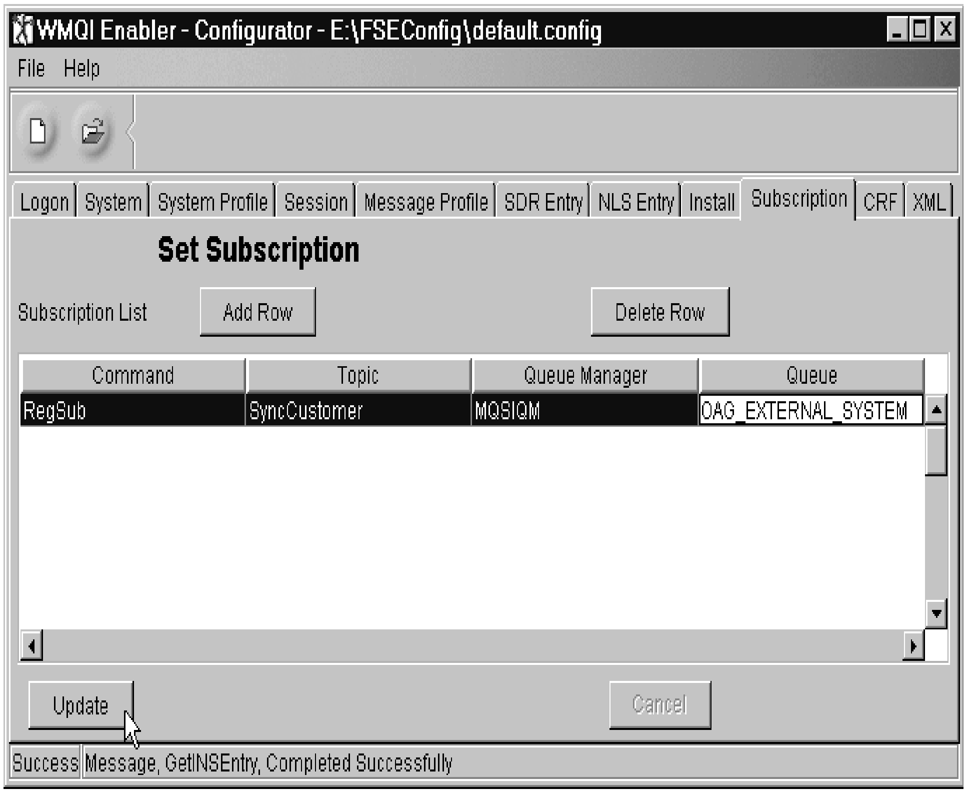
The fields in the Install entry tab are explained below:

Hardware Platform: Platform of WMQI Enabler is installation. Example, 'NT'

Product Version: WMQI Enabler product version. For Example: **01**

Default Language: Default language code for the system.
For example: US English = **10**.

Subscription tab



Command	Topic	Queue Manager	Queue
RegSub	SyncCustomer	MQSIQM	OAG_EXTERNAL_SYSTEM

Success Message, GetINSEntry, Completed Successfully

Figure 21: Subscription tab.

Allows the user to Register and drop subscriptions. All fields are required for registering and dropping subscriptions. The command for registering is 'RegSub'. The command for dropping (de registering) is 'DeregSub'.

SetSubscription: Enter values in each field and click **Update** to register or drop a subscription. The subscription will be registered/dropped per topic for just one queue and queue manager.

The fields in the subscription tab are explained below:

Command: 'RegSub' to register a subscription. 'DeregSub' to drop (deregister).

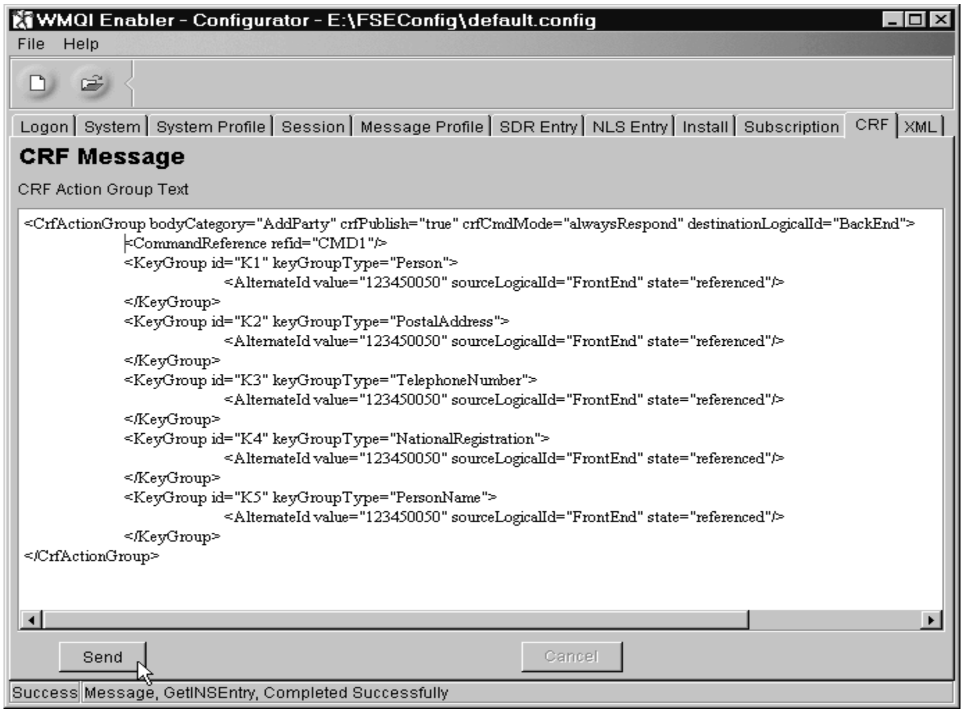
Topic: Topic about which subscriber wishes to receive publications.
For example: 'AddParty'

Queue Manager: Queue Manager to which publications should be published.

Queue: Queue to which publications should be published.

The Add Row/Delete Row: For these last two sections add and delete rows from the text area, respectively.

CRF tab



```
<CrfActionGroup bodyCategory="AddParty" crfPublish="true" crfCmdMode="alwaysRespond" destinationLogicalId="BackEnd">
        <CommandReference refid="CMD1"/>
        <KeyGroup id="K1" keyGroupType="Person">
                <AlternateId value="123450050" sourceLogicalId="FrontEnd" state="referenced"/>
        </KeyGroup>
        <KeyGroup id="K2" keyGroupType="PostalAddress">
                <AlternateId value="123450050" sourceLogicalId="FrontEnd" state="referenced"/>
        </KeyGroup>
        <KeyGroup id="K3" keyGroupType="TelephoneNumber">
                <AlternateId value="123450050" sourceLogicalId="FrontEnd" state="referenced"/>
        </KeyGroup>
        <KeyGroup id="K4" keyGroupType="NationalRegistration">
                <AlternateId value="123450050" sourceLogicalId="FrontEnd" state="referenced"/>
        </KeyGroup>
        <KeyGroup id="K5" keyGroupType="PersonName">
                <AlternateId value="123450050" sourceLogicalId="FrontEnd" state="referenced"/>
        </KeyGroup>
</CrfActionGroup>
```

```
Success Message, GetINSEntry, Completed Successfully
```

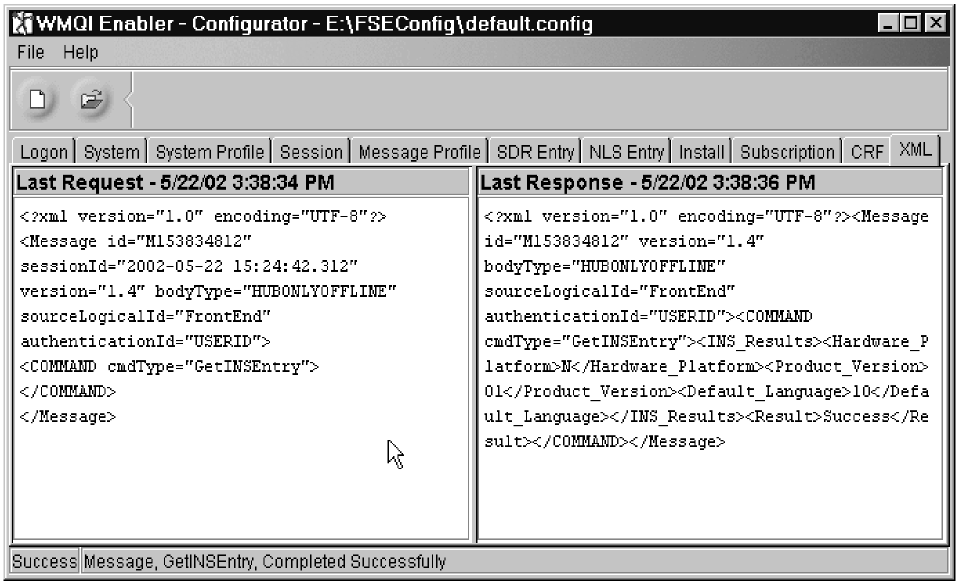
Figure 22: CRF tab.

This Tab allows the user to perform CRF actions.

CRF: The actual code that would appear in an WMQI Enabler header CrfActionGroup should be put in the text field. Click the **Send** button to send the message.

The CRF message page is different in that it is a “free for input” page where you would put the actual XML that made up the CRFActionGroup. The input field is initialized with the XML structure of the CRFActionGroup as a pattern.

XML Tab



Last Request - 5/22/02 3:38:34 PM

```
<?xml version="1.0" encoding="UTF-8"?>
<Message id="M153834812"
sessionId="2002-05-22 15:24:42.312"
version="1.4" bodyType="HUBONLYOFFLINE"
sourceLogicalId="FrontEnd"
authenticationId="USERID">
<COMMAND cmdType="GetINSEntry">
</COMMAND>
</Message>
```

Last Response - 5/22/02 3:38:36 PM

```
<?xml version="1.0" encoding="UTF-8"?><Message
id="M153834812" version="1.4"
bodyType="HUBONLYOFFLINE"
sourceLogicalId="FrontEnd"
authenticationId="USERID"><COMMAND
cmdType="GetINSEntry"><INS_Results><Hardware_P
latform>N</Hardware_Platform><Product_Version>
01</Product_Version><Default_Language>10</Defa
ult_Language></INS_Results><Result>Success</Re
sult></COMMAND></Message>
```

Success | Message, GetINSEntry, Completed Successfully

Figure 23: XML tab.

This page shows the actual Request/Response XML generated by the actions performed on the other Tabs.

Last Request: Displays the last Request message.

Last Response: Displays the last Response message.

Appendix
Notices

This information was developed for products and services offered in the U.S.A. and Europe. IBM may not offer the products, services, or features discussed in this document 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 document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories
Hursley Park
WINCHESTER, Hampshire
SO21 2JN
United Kingdom

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 Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks or services of IBM Corporation in the United States or other countries or both:

IBM®

MQSeries®

DB2®

IAA®

Insurance Application Architecture®

Rational Rose® is a trademark of Rational Software Corporation in the United States or other countries or both.

OAG is a trademark of the Open Architecture Group in the United States or other countries or both.

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

Permission statement

Copyright © 2001 Interactive Financial eXchange Forum. All Rights Reserved.

Redistribution and use of this material for both commercial and noncommercial purposes are permitted subject to the below-stated conditions:

1. This Permission Statement shall be reproduced in its entirety in each copy of the material;
2. This material is provided AS IS without warranty of any kind, including but not limited to, any warranty of noninfringement or any warranty (express or implied) of merchantability or fitness for a particular purpose; and
3. The material may be modified provided
    a. Prior written notice of each modification is provided to the Interactive Financial eXchange Forum at the address listed below,

Interactive Financial Exchange Forum, Inc.
333 John Carlyle Street
Suite 600
Alexandria, VA 22314
U.S.A.

b. Any redistribution of modified materials shall be accompanied by a notice that modifications have been made and a clear description of the modifications, and

c. The party making the modifications assumes all responsibility for the consequences of the modifications.

Glossary

This glossary defines terms and abbreviations used in this book. If you do not find the term you are looking for, see the *Index* or the ***IBM Dictionary of Computing***, New York: McGraw-Hill, 1994.

A

API: Application Programming Interface

(1) A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program.

(2) In VTAM, the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

C

Class

(1) A UML class. A description of an object. (2) refers to object oriented programming, a description of a set of similar objects.

D

DB2

An IBM relational database management system that is available as a licensed program on several operating systems. Programmers and users of DB2 can create, access, modify, and delete data in relational tables using a variety of interfaces.

E

EID: Enterprise Integration Domain

EIDBe: Enterprise Integration Domain Back-end

I

IAA: Insurance Application Architecture

IBM's business model for the insurance and financial services industry.

IAA-XML: Insurance Application Architecture-eXtensible Markup Language.

The Common Language used across integrated applications in MQSeries.

L

Log
A record of a sequence of operational activities on a computer.

M

MQSeries
Pertaining to a family of IBM licensed programs that provide message queuing services.

O

Object
Instance of a class.

P

Party
Any person or organization that the insurance company has, or had, or may have a business interest in.

Q

Queue
An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages: they point to other queues, or can be used as models for dynamic queues.

S

SQL: Structured Query Language
A programming language that is used to define and manipulate data in a relational database. It is often embedded in general purpose programming languages.

U

UML: Unified Modeling Language

W

WMQI Enabler: WebSphere MQ Integrator Enabler
A complete scalable messaging and information integration add-on to the MQSeries family of products. Especially designed for the needs of the financial services industry, WMQI Enabler can integrate front-end systems with back-end systems using a hub/spoke architecture using XML as the common vocabulary across systems.

X

XML

eXtensible Markup Language. XML is a markup language for message definition, and is an open and public domain standard. XML is a subset of SGML designed for easy implementation in commercial and web environments.

Index