WebSphere MQ Everyplace for Multiplatforms – MQe_Explorer

User Guide: Version 2.0

SupportPac Category 3 – March 2003

Barry Aldred
IBM Corporation
Hursley Park
Winchester
UK
SO21 2JN

barry_aldred@uk.ibm.com

Ninth Edition, March 2003

This edition applies to version 2.0 of *WebSphere MQ Everyplace for Multiplatforms – MQe_Explorer* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of contents

Figures

Summary of amendments

Date	**Changes**
6 November 2000	Version 1.0 (initial release)
1 May 2001	Version 1.20 This version supports the creation of queue managers without the use of initialization files. It provides for the remote set-up of other queue managers for administration and configuration. Password security is handled. Full support is provided for both online and offline management of MQe objects. MQe_Explorer can now be invoked with parameters or can be called from applications. There have been substantial functional and usability enhancements over the previous version. A tool is included for basic performance characterization.
14 May 2001	Version 1.21 Replacement of embedded MQe v1.20 classes with v1.21 classes.
10 August 2001	Version 1.23 Addition of support for the creation and management of the MQe bridge to MQSeries queue managers (MQe gateway capability). Replacement of embedded MQe v1.21 classes with v1.23 classes. Support for the mapping of the registry into a chosen storage adapter. Miscellaneous minor changes.
1 October 2001	Version 1.25 Addition of console support. Display of system information. Improved display of trace information. Additional support for the creation of new queue managers and the modification of existing local queue managers. Miscellaneous fixes & minor changes. Embedded MQe v1.23 classes replaced with v1.25 classes.
18 December 2001	Version 1.26 Additional properties supported for local queue managers. Services folder replaced by Bridges folder. Additional connection type and related changes. Accessibility improvements. Addition of a script for gateway configuration. Support for a description property for all bridge-related objects. Support for the message store property of queues. Addition of a ping capability. SupportPac classification upgraded to Category 3. Withdrawal of the MQe_ExplorerX.exe option. Miscellaneous fixes and minor changes.

14 June 2002	Version 1.27 Additional support for the WTLS mini-certificate server and for WTLS mini-certificates. Additional properties available when creating a queue manager. Support for *MQeMQMsgObject* class test messages. File transfer capability added using either *MQeMQMsgObject* or *MQeFileTransferMsg* class messages; the latter providing segmented file transfer support. Full support for destinations in store and forward queues. Programming examples added for MQe_Explorer applications. Extensions to the performance tool to allow measurements to MQSeries destinations. Miscellaneous fixes and minor changes, including exploitation of the bridge-enabled queue manager property introduced in MQe v1.27.
1 December 2002	Version 1.27a Inclusion of support for client connections to an MQSeries queue manager, removing the need to install the MA88 SupportPac for local gateway function. Miscellaneous fixes and minor changes.
3 March 2003	Version 2.0 Product renaming and support for MQe v2 features. Withdrawal of support for miscellaneous MQe v1 features. Additions to allow control of HTTP proxies.

Preface

The WebSphere MQ Everyplace MQe_Explorer is a management tool for WebSphere MQ Everyplace (MQe). It allows MQe queue managers and their associated objects, such as queues, connections and bridges, to be locally or remotely configured. Similarly, messages on queues can be inspected and test messages sent to validate the operation of the network. MQe_Explorer also provides a simple way of creating local queue managers, which can then be further configured to meet the needs of applications. It also offers a launch and debug environment for MQe applications.

A tool for gathering basic performance information is included.

*** Upgrade notice ***

If upgrading from a previous version of MQe_Explorer, please note:

- This version pre-reqs WebSphere MQ Everyplace version 2.003 or later – you must install that level (or later) on any machine that is to run MQe_Explorer. Any remote queue manager to be managed must also be upgraded to version 2.003 or later. Any MQe v1.x queue managers can be managed by MQe_Explorer v1.27a.

- If you intend to use the local gateway function (i.e. the bridge to WebSphere MQ within the same JVM as MQe_Explorer) there is no requirement to install the *WebSphere MQ Classes for Java*. Such a local bridge however can only operate in client mode; be aware that current releases of the *WebSphere MQ Classes for Java* no longer support the Microsoft JVM.

- Support for the following MQe and MQe_Explorer v1.x features is withdrawn in this release:
    - Peer channels.
    - Peer channel listeners and local connection definitions.
    - Distinction between clients, peers, servers and gateways.
    - Ini file sections excepting *[Registry]*, *[QueueManager]*, *[Preload]* and *[Alias]*.
    - Automatic setup of a remote queue manager for MQe_Explorer administration (alternate provision is available).

- Support for the following MQe v2.0 features is added in this release:
    - Remote administration of comms. listeners.
    - Miscellaneous additional object properties.
    - Version 2 registries and classes.
    - Integration of bridge capabilities into the queue manager.

- Information display has been modified, including:
    - Remote messages are only visible when inspecting the relevant *local queue*; previously such messages were also visible under *proxy queues* (previously known as remote queues – see below).
    - Comms. listeners are displayed in a folder under the relevant queue manager.

- Control over proxy configuration when using HTTP protocol-based communications.

- The configuration file **.cfg* is replaced by an options file **.opt*. Existing configuration files will not be migrated but a new default option file will be automatically created when required.

- There are changes to the programming interface for MQe_Explorer.

- Certain properties and types have been renamed for consistency and clarity across the MQe tooling; the most notable being:
    - *MQe local* queues ⇒ *application* queues
    - *MQe async. remote* queues ⇒ *async. proxy* queues
    - *MQe sync. remote* queues ⇒ *sync. proxy* queues
    - *MQe store and forward* queues (w/o target qMgr.) ⇒ *store* queues
    - *MQe store and forward* queues (with target qMgr.) ⇒ *forward* queues

Bibliography

- *WebSphere MQ Everyplace: Introduction,* IBM Corporation, SC34-6277
- *WebSphere MQ Everyplace Application Programming Guide,* IBM Corporation, SC34-6278
- *WebSphere MQ Everyplace Java Programming Reference,* IBM Corporation, SC34-6279
- *WebSphere MQ Everyplace Configuration Guide,* IBM Corporation, SC34-6283
- *WebSphere MQ SupportPac MA88: WebSphere MQ Classes for Java.*

Related material

- *WebSphere MQ Everyplace SupportPac EA01: WebSphere MQ Everyplace - XML conversion utility*
- *WebSphere MQ Everyplace SupportPac EAP1: WebSphere MQ Everyplace - Device code for Palm OS*
- *WebSphere MQ Everyplace SupportPac ED01: WebSphere MQ Everyplace - Get Started*
- *WebSphere MQ Everyplace SupportPac ED02: Using WebSphere MQ Everyplace with WebSphere Everyplace Server.*
- *WebSphere MQ Everyplace SupportPac EP01: WebSphere MQ Everyplace – Performance report*
- *WebSphere MQ Everyplace SupportPac ES03: WebSphere MQ Everyplace – WTLS Mini-Certificate Server*
- *WebSphere MQ Integrator SupportPac ID03: WebSphere MQ Integrator – Working with WebSphere MQ Everyplace*

Web references

The following URLs provide useful resources for both WebSphere MQ Everyplace and MQe_Explorer:

Download sites

IBM WebSphere (WebSphere MQ SupportPacs):

http://www.ibm.com/software/ts/mqseries/txppacs/

IBM Boulder (WebSphere MQ Everyplace downloads):

http://www6.software.ibm.com/dl/mqsem/mqsem-p

IBM Visual Age Micro Edition (Java stacks & related technologies):

http://www.embedded.oti.com

Microsoft Corp. (JVM downloads):

http://www.microsoft.com/java/download.htm

Newsgroups

IBM Software Group (WebSphere MQ Everyplace newsgroup):

news://news.software.ibm.com/ibm.software.websphere.mqeverplace

1 Introduction

MQe_Explorer is a management tool for WebSphere MQ Everyplace (MQe). It runs on Windows XP and other Windows platforms and can be used to manage one or more target MQe queue managers. Although the tool itself executes only on Windows platforms, the managed targets can be on any platform supported by MQe. MQe_Explorer has the Windows look and feel – mimicking the appearance of the *Windows Explorer* – and its use should be intuitive for those familiar with Windows applications. It is similar in some respects to the WebSphere MQ version 5 Explorer used to manage distributed WebSphere MQ, though it differs substantially in many details.

MQe_Explorer allows MQe queue managers and their associated objects, such as queues, connections and bridges, to be locally or remotely configured. Similarly, messages on queues can be inspected and test messages sent to validate the operation of the network. It provides a simple way of creating queue managers, which can then be further configured to meet the needs of applications. MQe_Explorer can be used to launch multiple MQe applications against a local queue manager in such a way that all applications share the same JVM. It can also remotely modify the configuration of existing queue managers such that they become amenable to MQe_Explorer inspection and change.

MQe_Explorer provides one of the fastest and easiest ways of getting started with WebSphere MQ Everyplace. It provides a powerful visualization of an MQe network and is a useful educational tool both for understanding the relationships between MQe objects and the functionality of MQe.

1.1 Support of WebSphere MQ

MQe_Explorer includes full support for the configuration and management of gateway queue managers, i.e. those MQe queue managers that can bridge to WebSphere MQ queue managers and queues. Those gateway queue managers created by MQe_Explorer *and* running in the same JVM, are restricted to interfacing to WebSphere MQ queue managers through the WebSphere MQ client interface (not the bindings interface). This capability is included with MQe_Explorer and does not require installation of additional software.

MQe_Explorer does not support the configuration of the WebSphere MQ queue managers themselves – they should be configured using the various WebSphere MQ management tools and protocols.

2 Installation

2.1 Environments

MQe_Explorer executes as an application running in a Java virtual machine. It can manage (and even create) its own local MQe queue manager and manage any remote MQe queue managers for which its local queue manager has MQe network connectivity.

Local execution and management environment

WebSphere MQ Everyplace MQe_Explorer version 2.0 requires a Microsoft Windows environment[1] with the following characteristics:

- *Microsoft Java Virtual Machine* (at a level of version 5.00.3155 or later[2]).
- *Microsoft Foundation Classes* (MFC) – these are always present on the supported Windows operating systems.

Windows 200 systems meet these requirements as does Windows NT 4.0 and Windows 98, with the correct level of Microsoft JVM installed. New installations of Windows XP do not include the JVM by default and therefore it must be downloaded from the Microsoft web site (see related footnote below). Upgrades to Windows XP preserve an existing JVM. If a choice of operating system is available, Windows XP or 2000 is preferred.

MQe_Explorer version 2.0 also requires a local installation of:

- *WebSphere MQ Everyplace version 2.003 or later.*

Remote management environments

Any environments (i.e. both Windows and non-Windows systems) running WebSphere MQ Everyplace version 2.003 or later may be remotely administered by MQe_Explorer version 2.0.

2.2 Contents

In addition to this User Guide, the MQe_Explorer package includes the files:

- *MQe_ExplorerV2.exe*
  The MQe_Explorer executable.

- *MQe_ExplorerV2.jar*
  A .jar file containing MQe_Explorer classes/resources and WebSphere MQ client classes.

[1] On Windows 95, 98 and ME there are minor functional deficiencies. Tool tips are not provided – affecting the provision of hover help for toolbar buttons, queue & message object class feedback on node selection in the object tree, etc.
[2] To determine the level of the Microsoft JVM type the command JVIEW at a DOS prompt. If the command is not recognized then your machine does not have a Microsoft JVM installed and any other JVM is not acceptable. JVM SDK downloads to install or upgrade are available from http://www.microsoft.com/java/download.htm.

Install the *MQe_ExplorerV2.exe* executable to run the MQe_Explorer from a program icon, the Windows *Run* menu, or from a DOS command line. Additionally (or alternatively), install *MQe_ExplorerV2.jar* to be able to invoke MQe_Explorer from a Java application or to gain access to the classes in the package *com.ibm.mqe.mqe_explorer*.

2.3 Installation

The following instruction is important:

Capitalization is significant to both MQe and MQe_Explorer. At all times, both during installation and subsequent use, ensure that any text input matches the capitalization given in this document – this instruction applies to all names and strings (i.e. class names, alias names, directory names, file paths and names etc). Descriptive text is the only exception.

Before installation check that you have a Microsoft JVM installed on your machine and that it is at the correct level – see the details given in *Local execution and management environment* on page 2[3].

Preparation

The instructions below assume that the standard MQe v2.0 installation defaults have been adopted[4]. Thus, as a brief check on the MQe installation, confirm that a directory *C:\Program Files\MQe\Java\com\ibm\mqe* containing various *.class* files and other sub-directories (such as *\adapters*, *\administration* etc) exist. If these directories and files are missing you do not have a complete MQe installation and must re-install.

It is important to ensure that the *classpath*[5] variable has been set in accordance with the MQe installation instructions. On Windows systems there is a choice available between either setting the *classpath system variable,* or setting the *classpath user variable*. The system variable (illustrated below) will apply to all users of the machine; the user variable applies only to the selected user. Note that any value present for the user variable will over-ride the system variable value for that user.

On Windows 2000 & XP:

From the *Windows Start* button, go to *Settings*, then *Control Panel* and click on the *System* icon to bring up the *System Properties* panel. Click the *Advanced* tab followed by the *Environment Variables…* button. In the resulting *System Variables* section check the *classpath* definition. It should appear as:

```
classpath          C:\Program Files\MQe\Java
```

Alternatively, the string "C:\Program Files\MQe\Java" should be present in the value.

If *classpath* is not present, click *New…*, then add "classpath" in the *Variable Name* input field and "C:\Program Files\MQe\Java" in the *Variable Value* field; click *OK*, then exit the previous panel via the *OK* button.

If *classpath* is present with the wrong value, then select *classpath*, click *Edit…*, and append the string ";C:\Program Files\MQe\Java" to the existing *Variable Value*. Click *OK*, and then exit the previous panel via the *OK* button.

---

[3] If on starting MQe_Explorer you get the error message "Unable to start the application -- the Java Virtual Machine cannot be loaded. Class not registered", then the JVM is not at the correct level.
[4] Non-default MQe installations are acceptable if the appropriate name substitutions are made for those given in this document.
[5] Once MQe_Explorer is running, the value of the *classpath* variable is shown in *System Information*, accessed from the *Help→About MQe_Explorer* menu item.

On Windows NT:

From the *Windows Start* button, go to *Settings*, then *Control Panel* and click on the *System* icon to bring up the *System Properties* panel. Click the *Environment* tab. In the *System Variables* section check the *classpath* definition. It should appear as:

```
classpath          C:\Program Files\MQe\Java
```

Alternatively, the string "C:\Program Files\MQe\Java" should be present in the value.

If *classpath* is not present, click an item in the *System Variables* section, then add "classpath" in the *Variable* input field and "C:\Program Files\MQe\Java" in the *Value* field; click *SET*, then exit via the *OK* button.

If *classpath* is present with the wrong value, then select *classpath*, and append the string ";C:\Program Files\MQe\Java" to the existing string in the *Value* field. Click *SET*, then exit via the *OK* button.

On Windows 98:

From the *Windows Start* button go to *Run…* In the program name input field type "sysedit" and click *OK*. The *System Configuration Editor* appears; click on the C:\AUTOEXEC.BAT window to bring it to the front. In this file the line:

```
set classpath=C:\Program Files\MQe\Java
```

should be present, or alternatively the string "C:\Program Files\MQe\Java" should be present in the *classpath* value. If *classpath* is missing add the line shown above; if *classpath* is present with the wrong value, append the string ";C:\Program Files\MQe\Java" to the existing value. Go to *File*, click *Save*, then *Exit*.

If the *classpath* system variable is not correctly set MQe_Explorer (and any other MQe applications) will not be able to locate the MQe classes and will not execute.

The sample Windows NT authenticator (supplied with MQe) is used later in the sample scripts in this guide to demonstrate queue security. Ensure that you have made all the changes necessary for this authenticator to execute as per the MQe v2.0 installation instructions.

MQe_Explorer executable - new installation

The MQe_Explorer executable allows you to run the MQe management tool from the Windows desktop. If you have not previously installed a version of MQe_Explorer then:

1. Create a directory *C:\Program Files\MQe\Java\MQe_Explorer* and move the file MQe_ExplorerV2.exe into it.

2. Create a desktop shortcut to MQe_Explorer by locating the *MQe_ExplorerV2.exe* file in the *Windows Explorer*, right click it and select *Create Shortcut*. Drag the resulting shortcut to the desktop. Rename if desired.

MQe_Explorer executable - upgrade

If you have previously installed a version of MQe_Explorer executable, then:

1. In the directory *C:\Program Files\MQe\Java\MQe_Explorer* replace the existing *MQe_ExplorerV2.exe* with the new file.

2. If you have created existing v1.x queue managers, ensure that they have been migrated to v2.x.

3. Delete any files with the extension *.cfg*; they are no longer required.

Installing the MQe_Explorer classes

The MQe_Explorer classes allow you to launch and run the MQe management tool from within a Java application. To install the classes store the *MQe_ExplorerV2.jar* file in your preferred location in the file system and then add the file into the *classpath* (using the full file specification, i.e. including its path)

The classes provide access to the same function as the *MQe_ExplorerV2.ex*e, including the WebSphere MQ client for use by the bridge, the performance tool and the example rule class.

The WebSphere MQ Classes for Java

The *WebSphere MQ Classes for Java* provide a Java interface to WebSphere MQ queue managers, either locally (in bindings mode) or remotely (through a WebSphere MQ client channel). When an MQe queue manager is configured as a gateway queue manager, it normally uses the *WebSphere MQ Classes for Java* to exchange information with WebSphere MQ queue managers. Current levels of these classes no longer support the Microsoft JVM, however to overcome this restriction, MQe_Explorer 2.0 includes the necessary function for local MQe gateway queue managers to be created and to communicate with WebSphere MQ queue managers over client channels. Consequently there is no requirement to install the *WebSphere MQ Classes for Java* and no associated *classpath* changes need be made.

Customization

MQe_Explorer can be customized through the *Tools/Options* menus. These panels give control over display layout, list box contents, default values, HTTP proxy options and performance-related parameters. The installation defaults are tuned to small MQe networks; networks with many queue managers, with slow links or with custom classes being deployed, may benefit from option changes. See the appropriate chapter and section headings for more details.

3 Getting started

3.1 Configuring a first queue manager

Start MQe_Explorer by clicking on the desktop shortcut .

If this is a new installation (or an upgrade and you have deleted the old configuration file) then a message will appear indicating that no saved options have been found and that defaults will be used.

Click *OK* in the message box.

The main MQe_Explorer window then appears:

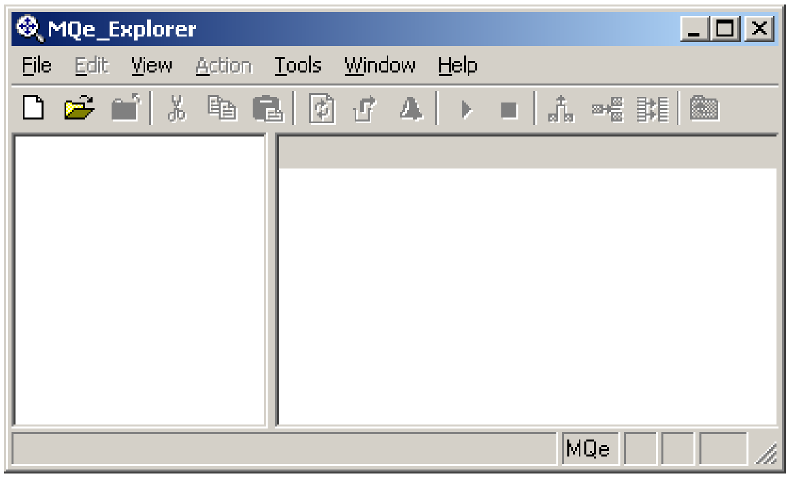


Figure 3-1: New MQe_Explorer window

Click the file *New* toolbar icon 🗋 to create a new queue manager.

The window for queue manager creation appears:

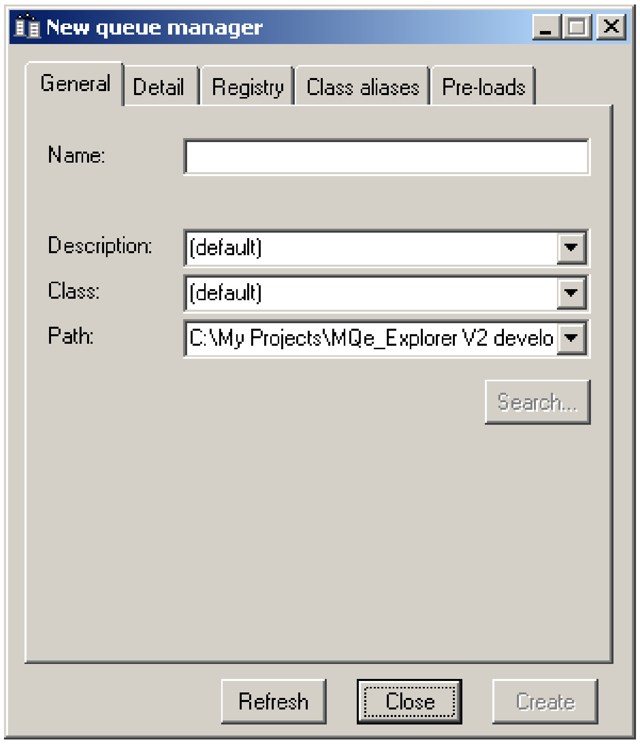


Figure 3-2: New queue manager creation – General tab

Type in the string "FirstQM" as the name of the new queue manager and click the *Create* button. A message box appears displaying the name of an initialization file – this name is important because, if in the future you want to restart the queue manager, you will need to open this file with MQe_Explorer.

Click OK to accept the message (making a note of the full file path for future use).

The queue manager is created and the main MQe_Explorer window changes appearance:

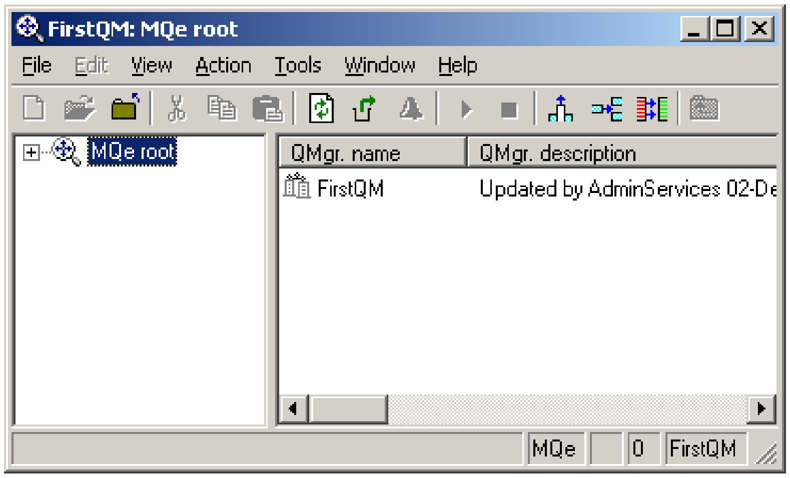


Figure 3-3: Main MQe_Explorer window

The new queue manager is now running; configured (by default) as a client, i.e. unable to accept incoming connection requests from other queue managers. MQe_Explorer is running in the same JVM and as an application of that queue manager – other applications may now be loaded into that same JVM if required. If MQe_Explorer is subsequently shut down the fully configured MQe queue manager will still exist on the file system and may be later restarted. A structure of sub-directories has been created to store the configuration and these are located along with the initialization file in the file system.

If you are anxious to get started and are familiar with Windows, you may choose to skip the rest of this chapter and run the first sample script *Concepts and objects* on page 120.

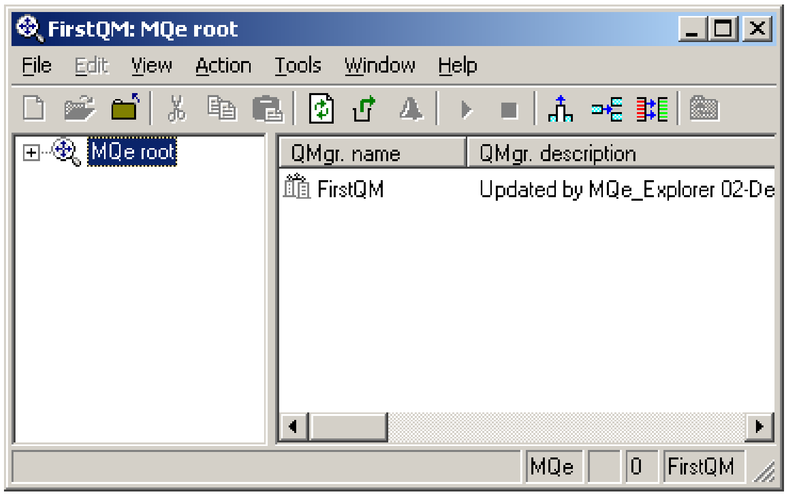
3.2 Main window layout

The principal elements of the main window are a menu bar at the top, a toolbar below, two large panes (a tree view pane on the left, a list view pane on the right) and a status bar at the bottom. Individual menu options are described in a later chapter; the various toolbar buttons offer shortcuts to commonly used menu items.

The tree view pane presents a tree view of the MQe network.

Expand the ⊞ sign next to the *MQe root* icon.

The next level of the tree is visible:



QMgr. name	QMgr. description
FirstQM	Updated by MQe_Explorer 02-De

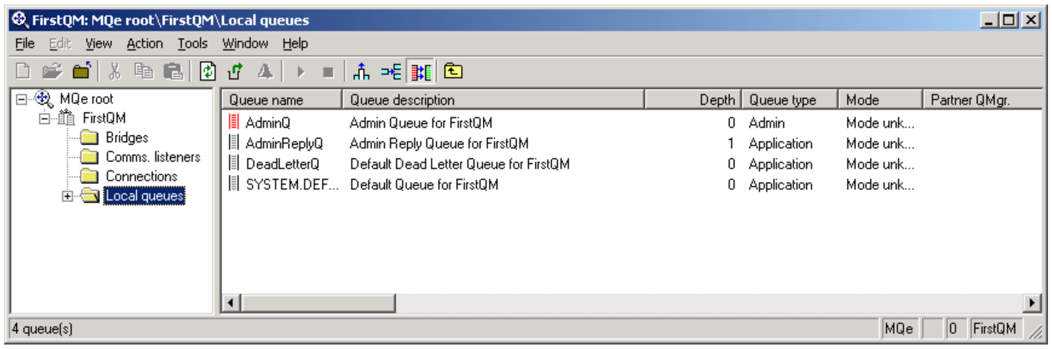
Figure 3-4: Root expanded in main MQe_Explorer window

Expanding the ⊞ symbol next to the queue manager icon reveals the next level. The queue manager icon has four forms: when a queue manager has responded with its details it has the appearance (unselected) (selected); however if no response has been received then it is shown as (unselected) (selected).

Other nodes can be expanded, just as were the *MQe root* and the *FirstQM* queue manager. For example, closed folders represented by this icon, can be selected and opened.

Select *FirstQM*. Click the ⊞ icon that appears next to *FirstQM* (note: a double click of a tree node has the same effect as a select followed by a click).

Click the folder *Local queues*. Expand the main window by pulling the right hand edge. Adjust the spacing between the tree view and list view panes by dragging the separator.


FirstQM: MQe root\FirstQM\Local queues

Queue name	Queue description	Depth	Queue type	Mode	Partner QMgr.
AdminQ	Admin Queue for FirstQM	0	Admin	Mode unk...	
AdminReplyQ	Admin Reply Queue for FirstQM	1	Application	Mode unk...	
DeadLetterQ	Default Dead Letter Queue for FirstQM	0	Application	Mode unk...	
SYSTEM.DEF...	Default Queue for FirstQM	0	Application	Mode unk...	

4 queue(s) MQe 0 FirstQM

Figure 3-5: Queues folder expanded in main MQe_Explorer window

Besides folders and queue managers, many other tree node icons may be expanded, including: connections, queues, messages, fields-type fields and many bridge-related icons. The window title changes to reflect the currently selected element in the tree. At the same time the list view pane shows the details of the child objects corresponding to the selected object in the tree. In the example above, the title indicates that the *FirstQM* queue manager is hosting MQe_Explorer, and that the *FirstQM* local queues folder is selected. The list view shows the four queues owned by *FirstQM*. Clicking the ⊟ symbol contracts the tree display.

The list view comprises rows and columns. Rows may be selected; in some views only a single row is selectable – in others multiple rows can be selected. Columns can be re-ordered by dragging the column header to a new position (that is, over other headers); columns can be temporarily re-sized by dragging the column boundaries in the header. Permanent control of column order, the columns to be displayed and their default widths, are all controlled through a *Tools→Options* menu panel – see page 69. Clicking a column header sorts the data by that column; re-clicking re-sorts in the reverse sequence. Double clicking a row in the list view pane is identical to selecting the equivalent object in the tree view pane.

The overall MQe_Explorer window may be resized and the divider between the tree and list view panes can be dragged to change the relative pane widths.

The status bar contains five panels; these are, from left to right:

- *Status and information message area*.
- *Environment* (normal operation is shown as MQe; *supervisor mode* is shown as MQe+).
- *Number of objects selected in the list view pane* (when more than one selected).
- *Outstanding work items* (see *Operation* on page 116).
- *Name of the local queue manager hosting MQe_Explorer*.

The status bar may be double clicked; this displays a panel with more information on object load status.

3.3 Navigation and control

MQe_Explorer supports the standard Windows conventions for selection, shortcuts, context menus, tabbing through fields, keyboard operations, clipboard, and dragging and dropping.

Multiple object selection is supported (where appropriate) using shift key and control key combinations with the mouse select button. *Control A* selects everything in the list pane (when multiple selections are enabled). Keyboard shortcuts are displayed on the menu items where applicable. Right clicking on an object will reveal context menus for that object – and is supported for all list view and tree view objects. *Tab* will move to the next object in a panel or window – for ease of data entry. *Cut*, *copy* and *paste* functions to/from the clipboard is supported for message objects. Likewise *drag and drop* of message(s) from the list view pane to the tree view pane is permitted; be aware that only certain objects in the tree are valid drop targets. *Shift* held down whilst dragging indicates a move operation; *Control* held down, a copy operation. The complete set of supported keyboard operations is listed in *Accessibility features* on page 173.

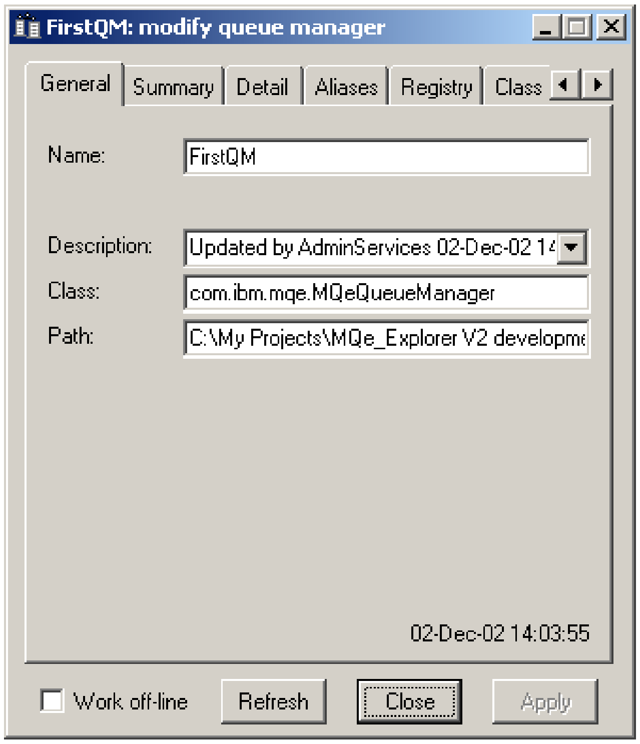
Hover help is available on many of the displayed objects. For example, when a queue is selected in the tree view, hover help shows its class name. Similarly, when a message object (or a fields object) is selected in the tree, hover help shows the class name of the message object.

3.4 Setting properties

Properties are changed through the use of property pages.

Right click on *FirstQM* in the tree. In the context menu that appears, click the *Properties* sub-menu item.

The following window appears, displaying the property pages for the *FirstQM* queue manager:



Name: FirstQM

Description: Updated by AdminServices 02-Dec-02 14

Class: com.ibm.mqe.MQeQueueManager

Path: C:\My Projects\MQe_Explorer V2 developme

02-Dec-02 14:03:55

Figure 3-6: Property pages for the *FirstQM* queue manager

The tabs each hold logical grouping of properties. Property values may be changed by editing the currently displayed value (where permitted); followed by clicking the *Apply* button to action the updates. The *Refresh* button causes the properties to be re-queried and displayed; any drop-down lists are re-loaded. *Close* removes the window. Property windows (and indeed most other MQe_Explorer windows) do not need to be closed immediately after use; they can either be left on the screen or minimized. MQe_Explorer is not modal in its operations and so, for example, the main window can be used even when other windows are present. The *Window* menu item provides for control of open windows.

3.5 Creating and deleting objects

In a similar manner, objects are created through property pages. To create an object select the parent object, in either the tree view pane or the list view pane, and then use the *New* context menu item or the *File→New* menu item. Most objects can be created in this way but certain restrictions exist:

> For queue managers, only the local queue manager object can be created and only when a local queue manager is not currently loaded.
>
> Extra fields in messages on queues cannot be created.

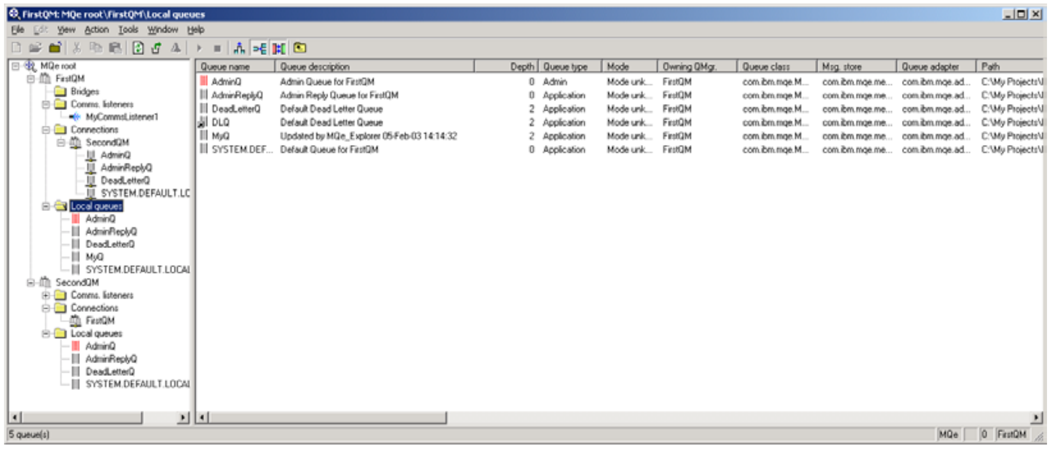
Most other MQe objects can be freely created, including queues, connections, comms. listeners, messages, MQ bridge objects, MQ queue manager proxies, MQ client connections and MQ listeners. In the majority of cases the parent object must be first selected – for example for a queue, the parent is the *Local queues* folder. However there are many other cases supported to aid usability, e.g. test messages can be sent to target queues by selecting almost any of the elements in the tree view pane. Likewise a *Test message* property page, however instantiated, is capable of sending a message to any queue in the network.

Objects are deleted by selecting them and pressing the *Delete key* (or by using the *Edit→Delete* option). One or more messages can be selected and deleted in a single operation; messages can also be cut, copied and pasted as well as dragged and dropped

When working offline, i.e. when connectivity to the remote queue manager is not available, administration of queue managers, queues, comms. listeners and connections is still possible. Since the parent object or the object to be managed cannot always be displayed, MQe_Explorer enables the relevant operation on a displayable ancestor object – thus to delete a queue, the offline delete queue operation is enabled on the *Local queues* folder.

3.6 *A simple network*

The *Concepts and objects* script on page 120 creates a simple network. The figure below shows the MQe_Explorer main window view of this network shortly after creation:



FirstQM: MQe root\FirstQM\Local queues

Queue name	Queue description	Depth	Queue type	Mode	Owning QMgr.	Queue class	Msg. store	Queue adapter	Path
AdminQ	Admin Queue for FirstQM	0	Admin	Mode unk...	FirstQM	com.ibm.mqe.M...	com.ibm.mqe.me...	com.ibm.mqe.ad...	C:\My Projects\
AdminReplyQ	Admin Reply Queue for FirstQM	0	Application	Mode unk...	FirstQM	com.ibm.mqe.M...	com.ibm.mqe.me...	com.ibm.mqe.ad...	C:\My Projects\
DeadLetterQ	Default Dead Letter Queue	2	Application	Mode unk...	FirstQM	com.ibm.mqe.M...	com.ibm.mqe.me...	com.ibm.mqe.ad...	C:\My Projects\
DLQ	Default Dead Letter Queue	2	Application	Mode unk...	FirstQM	com.ibm.mqe.M...	com.ibm.mqe.me...	com.ibm.mqe.ad...	C:\My Projects\
MyQ	Updated by MQe_Explorer 05-Feb-03 14:14:32	2	Application	Mode unk...	FirstQM	com.ibm.mqe.M...	com.ibm.mqe.me...	com.ibm.mqe.ad...	C:\My Projects\
SYSTEM.DEF...	Default Queue for FirstQM	0	Application	Mode unk...	FirstQM	com.ibm.mqe.M...	com.ibm.mqe.me...	com.ibm.mqe.ad...	C:\My Projects\

5 queue(s)

Figure 3-7: Main MQe_Explorer window of a simple network

The tree view is now more complicated than that shown previously – the *MQe ro*ot node has two child queue manager objects, representing the two queue managers in the network. Each of these has a number of local queues; each queue manager also has a connection defined to the other, a comms. listener to receive incoming connection requests, and each has proxy queues representing remote application queues on the other.

The node selected in the tree is identified in the title bar as *MQe root\FirstQM\Local queues*, that is the collection of five local queues belonging to *FirstQM*. Ten properties of these queues, Q*ueue name*, *Queue description, Depth*, etc. are visible in the list view pane – with more properties present, but not visible, to the right (as apparent from the position of the horizontal scroll bar in that pane). The toolbar option to show queue alias names as additional rows is enabled; consequently the five queues are displayed as six rows, with the additional row being due to a queue also having an alias name of “DLQ”. The *outstanding work items* information in the status bar is shown as “0” (which means that all the incoming and outgoing requests for information have been processed). The hosting queue manager for MQe_Explorer is *FirstQM*; this information appears both as a prefix in the title bar and in the panel at the bottom right of the status bar.

3.7 Tree view layout

The tree view shows the hierarchy of MQe objects – starting with the MQe root that represents the entire network. The basic structure is:

- MQe root
    - Queue managers…
        - Bridges folder
            - Bridges…
                - MQ proxies…
                    - Client connections…
                        - Listeners…
        - Comms. listeners folder
            - Comms. listeners…
        - Connections folder
            - Connections…
                - Sync. proxy queues…
                - Async. proxy queues…
                - Forward queues…
                - Home server queues…
                - Bridge queues…
                    - Messages…
                        - Fields…(Fields…)
        - Local queues folder
            - Application queues…
                - Messages…
                    - Fields…(Fields…)
            - Store queues…
            - Admin queues…

Some important points to note are:

1. Some queues appear under individual *Connections*, others under the *Local queues* folder.
2. Messages held on an MQe queue manager can only be viewed by inspecting the application queues on that queue manager; messages on a WebSphere MQ queue manager are visible from the associated bridge queue.
3. Messages in transit on store, forward and async.s proxy queues cannot be inspected; neither can messages awaiting processing on admin queues.

3.8 Remote administration

As well as supporting the creation and management of a local queue manager, MQe_Explorer is also able to manage remote queue managers. To do this it requires MQe connectivity with those remote queue managers; and likewise they must be able to communicate with the local queue manager hosting MQe_Explorer. This involves the presence of both connection definitions and queue definitions on both queue managers. Management itself can be either take place online, which is the common case, and where the results of an admin operation are immediately executed and available; or offline, where the execution of the admin operation occurs at some later time. In this book online administration is assumed to be the norm; the offline aspects are deferred until the section *Offline administration* on page 98. The details of the queue manager configuration required for MQe_Explorer administration are described in *Managing remote queue managers* on page 113.

3.9 Gateway administration

MQe_Explorer supports the creation and administration of gateway queue managers that have the ability to link an MQe network with a WebSphere MQ network. The facilities are described under the relevant menu topics. A script, *Gateway configuration and use* on page 149, illustrates setting up such a gateway from both MQe and WebSphere MQ perspectives.

It is now recommended that you run the first script *Concepts and objects* on page 120. The information in the following chapters is more detailed and may be more useful after the first script has given an appreciation of MQe_Explorer operation. Subsequent scripts are concerned with setting up different messaging topologies and exploring more advanced MQe function.

4 Main window menus

4.1 File

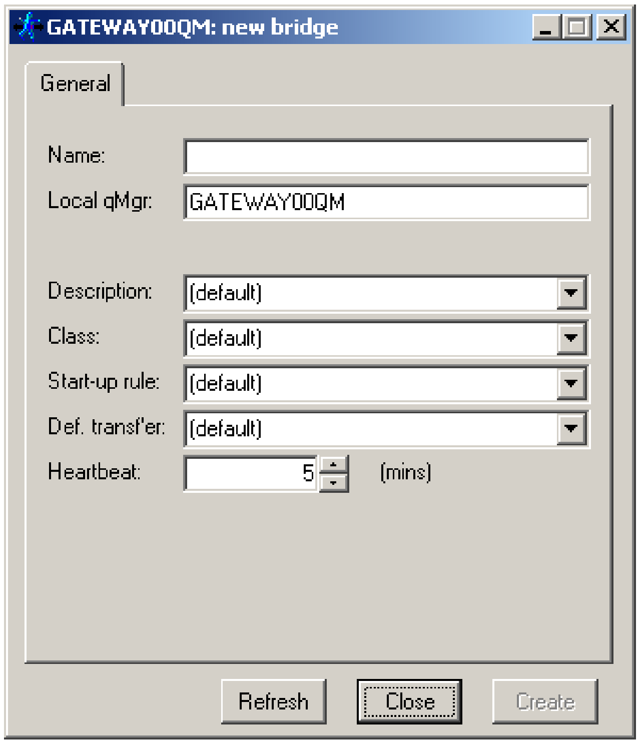
New

This command has eight sub-commands – the available one is determined by the nature of the object selected in either the tree view pane or the list view pane:

- *Bridge (*select a *Bridges folder).*
- *MQ client connection* (select an *MQ proxy).*
- *Comms. listener* (select a *Comms. Listeners folder).*
- *Connection* (select a *Connection folder*).
- *Listener (*select an *MQ client connection).*
- *Message* (select a *Queue manager*, a *Local queues folder*, a suitable local or remote queue, or a *Connection*).
- *MQ proxy (*select an *MQ bridge.)*
- *Queue* (select either a *Local queues folder* or a *Connection*).
- *Queue manager* (requires that a local queue manager has not been loaded – no prior selection required).

Bridge

File→New→Bridge enables the *New bridge* definition dialog:



Name:
Local qMgr: GATEWAY00QM
Description: (default)
Class: (default)
Start-up rule: (default)
Def. transf'er: (default)
Heartbeat: 5 (mins)

Figure 4-1: New bridge dialog

The dialog presents a single tab with the bridge characteristics. The tab is shared with the *Bridge properties* dialog, used to change an existing bridge.

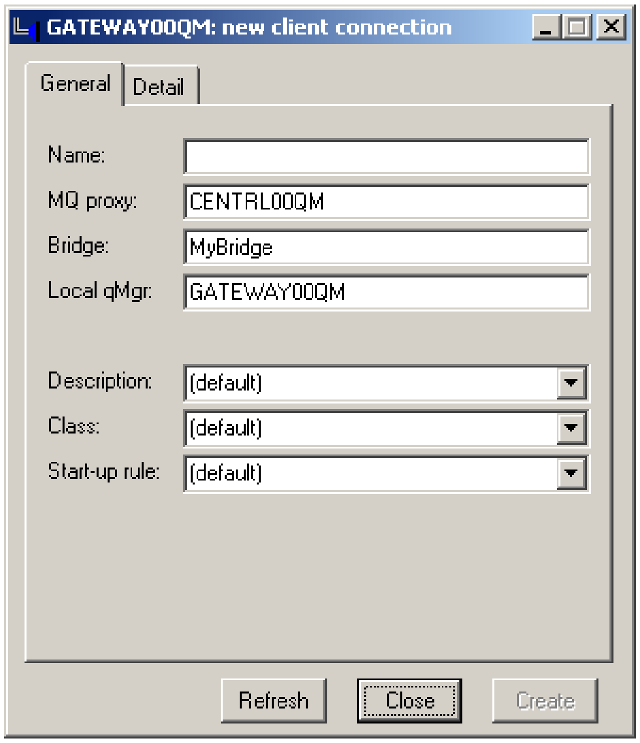
The following input fields may be present:

- *General* tab:
    - *Name* – the name of the new bridge.
    - *Local queue manager* – the name of the queue manager owing this definition.
    - *Description* – any UNICODE string.
    - *Bridge class (or alias)* – the class that implements the bridge.
        - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeMQBridge".
    - *Startup rule class (or alias)* – the class that determines whether the bridge is started at object creation time or when the owning queue manager is opened.
        - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeStartupRule".
        - *(none)* is mapped to "" (i.e. no rule class). Objects are created in the stopped state.
    - *Default transformer class (or alias)* – this sets the transformer class used for message conversion – if not otherwise specified by either (a) the target queue (for messages being sent from MQe to WebSphere MQ), and/or (b) the MQ transmission queue listener (for messages being sent from WebSphere MQ to MQe).
        - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeBaseTransformer".
    - *Heartbeat* – defines the frequency of pulses issued to bridge resources. Using this stimulus, bridge resources perform low-priority monitoring, cleanup and timeout operations, for example, the closure of client connections to MQ that have remained idle beyond a specific timeout period. The value is in minutes within the range 1 – 60. The default is 5.
    - *Status* – indicates whether the bridge is in the stopped or started state.

The *Create* button creates the new bridge and updates the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Client Conn.

File→New→Client Conn. enables the *New client connection* definition dialog:



Name:
MQ proxy: CENTRL00QM
Bridge: MyBridge
Local qMgr: GATEWAY00QM
Description: (default)
Class: (default)
Start-up rule: (default)

Figure 4-2: New client connection dialog

The dialog presents tabs with the client connection characteristics. These tabs are shared with the *Client connection properties* dialog, used to change an existing client connection.

The following input fields may be present:

- *General* tab:
  - *Name* – the name of the new client connection.
  - *MQ proxy* – the name of the MQ proxy owning this definition.
  - *Bridge* – the name of the bridge owning this definition.
  - *Local queue manager* – the name of the queue manager owing this definition.
  - *Description* – any UNICODE string..
  - *Client connection class (or alias)* – the class that implements the client connection.
    - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeClientConnection".
  - *Startup rule class (or alias)* – the class that determines whether the client connection and its parents are started at object creation time or when the owning queue manager is opened.
    - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeStartupRule".
    - *(none)* is mapped to "" (i.e. no rule class). Objects are created in the stopped state.
  - *Status* – indicates whether the client connection is in the stopped or started state.

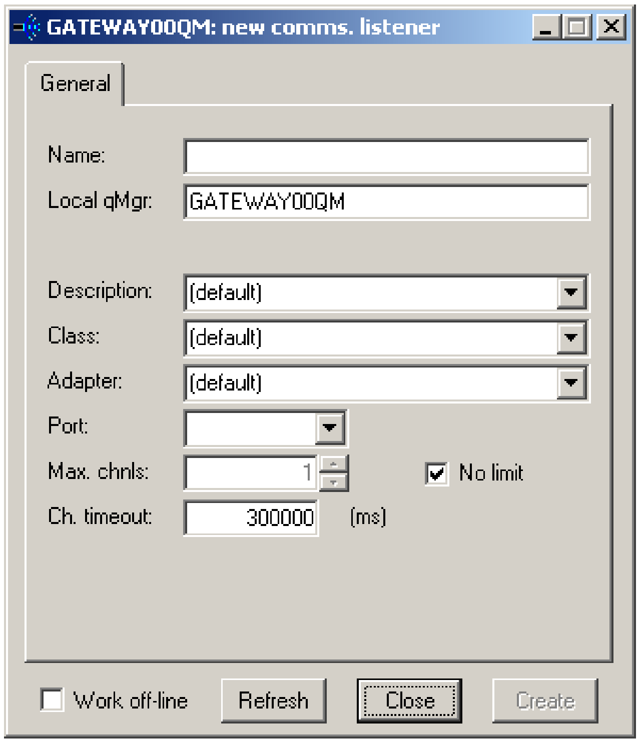
- *Detail* tab:
  - *Bridge adapter class (or alias)* – the class used to move messages from MQe to the target WebSphere MQ queue manager.
    - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeMQAdapter".
  - *User id* – the WebSphere MQ user id.
    - *(default)* is mapped to "".
  - *Password* – the WebSphere MQ password (note that the typed string is displayed in * characters).
  - *Port* – the IP port number used by the target WebSphere MQ queue manager. If bindings mode is being used, this value should be set to ""; otherwise the default is 1414.
  - *CCSID* – the WebSphere MQ CCSID property. The value must be a zero or positive integer; the default is 819.

- *Send exit* – used to match the send exit specified at the remote end of the WebSphere MQ client channel.
    - *(default)* is mapped to “".
- *Receive exit* – used to match the receive exit specified at the remote end of the WebSphere MQ client channel.
    - *(default)* is mapped to “".
- *Security exit* – used to match the security exit specified at the remote end of the WebSphere MQ client channel.
    - *(default)* is mapped to “".
- *Sync. queue* – the name of the synchronization queue on the WebSphere MQ queue manager that is used by the MQ bridge. This queue keeps track of messages moving from MQe to WebSphere MQ and, if the listener state store is also set to use WebSphere MQ; it also keeps track of messages moving from WebSphere MQ to MQe.
    - *(default)* is mapped to “MQE.SYNC.DEFAULT".
- *Purger rule class (or alias)* – the class that is used when a message on the sync. queue indicates a failure of MQe to confirm a message.
    - *(default)* is mapped to “"com.ibm.mqe.mqbridge.MQeSyncQueuePurgerRule".
- *Purger interval* – the time between successive purges of the synchronization queue. The value is in minutes and must be a zero or positive integer; the default is 60.
- *Max. idle* – the time after which an idle connection to WebSphere MQ is discarded and the resources returned to the pool. The value is in minutes within the range 0 – 720. The default is 5.

The *Create* button creates the new client connection and updates the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Comms. Listener

File→New→Comms. Listener enables the *New comms. listener* definition dialog:



Name:
Local qMgr: GATEWAY00QM
Description: (default)
Class: (default)
Adapter: (default)
Port:
Max. chnls: 1
No limit [✓]
Ch. timeout: 300000 (ms)

Figure 4-3: New communications listener dialog

The following fields are present:

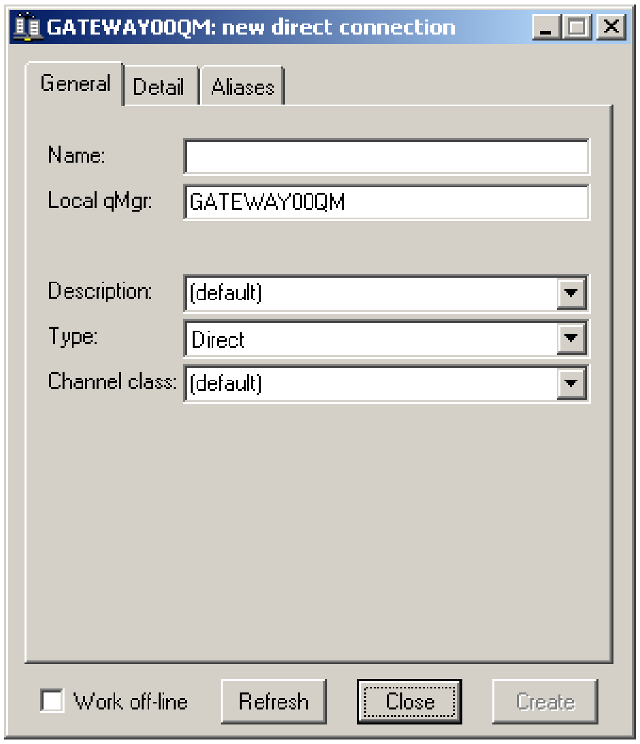
- *General* tab:
    - *Name* – the name of the new communications listener.
    - *Local queue manager* – the name of the queue manager owing this definition.
    - *Description* – any UNICODE string.
    - *Comms. listener class (or alias)* – the class that handles incoming connection requests.
        - *(default)* is mapped to “com.ibm.mqe.communications.MQeListener”.
    - *Adapter class (or alias)* – the class of the communications protocol adapter. Each protocol adapter supports a specific IP protocol, for example HTTP, TCP or UDP; multiple adapters may exist per protocol with differing performance or other characteristics.
        - *(default)* is mapped to "com.ibm.mqe.adapters.MQeTcpipHistoryAdapter".
    - *Port* – the IP port number used by the remote queue manager to service incoming connection requests.

- *Max. channels* – the maximum number of channels that will be spawned to service the reception needs of the queue manager through this listener. If *No limit* is checked then an unlimited number of channels can be simultaneously instantiated.
- *Channel timeout* – the time in milliseconds after which an idle incoming connection will timeout. This value must be zero (no timeout) or a positive integer; the default value is 300,000.
- *Current channels* – the number of channels currently associated with this comms. listener.

The *Create* button creates the new comms. listener and updates the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window. *Work offline* allows comms. listener definitions to be created on remote queue managers that are not currently available on the network.

Connection

File→New→Connection enables the *New connection* definition dialog (sometimes known as a new remote queue manager definition):


GATEWAY00QM: new direct connection

Name:
Local qMgr: GATEWAY00QM
Description: (default)
Type: Direct
Channel class: (default)

Figure 4-4: New connection dialog

Various types of connection can be created[6]:

- *Alias/MQ*

  An *alias connection* is a specialized type of connection that is used to give alias names to remote queue managers that feature as destinations

[6] Local connections are not supported by MQe_Explorer v2; however the information content (queue manager aliases) is presented by MQe v2 as a queue manager property and supported in the queue manager dialogs. Peer connection information from MQe v1 queue managers is ignored.

in store or forward queue definitions. No other parameters are required because the store or forward queue itself handles any associated delivery aspects.

An *MQ connection* identifies a remote queue manager as a WebSphere MQ queue manager, as opposed to an MQe queue manager.

- *Direct*

  A direct connection supplies the information needed for the local queue manager to create channels directly to another queue manager elsewhere in the MQe network. The connection name is required to match the name of that remote queue manager. Direct means that the channels go to the remote queue manager without passing through an intermediate queue manager.

- *Indirect*

  An indirect connection (or *via* connection) indicates that messages destined for a remote queue manager elsewhere in the MQe network are to be sent to an intermediate queue manager (for which a direct connection must exist). The connection name must match the name of the destination remote queue manager.

The dialog presents various tabs, each relating to a related set of characteristics. The tabs are shared with the *Connection properties* dialog, used to change an existing queue. The complete set of tabs is:

- *General*

  The name of the connection, local queue manager, type and channel class.

- *Detail*

  The network associated connection details and related information.

- *Aliases*

  Alternative names that should be resolved to this connection.

The following input fields may be present (note that the properties available depend upon the type of connection and the values of other properties):

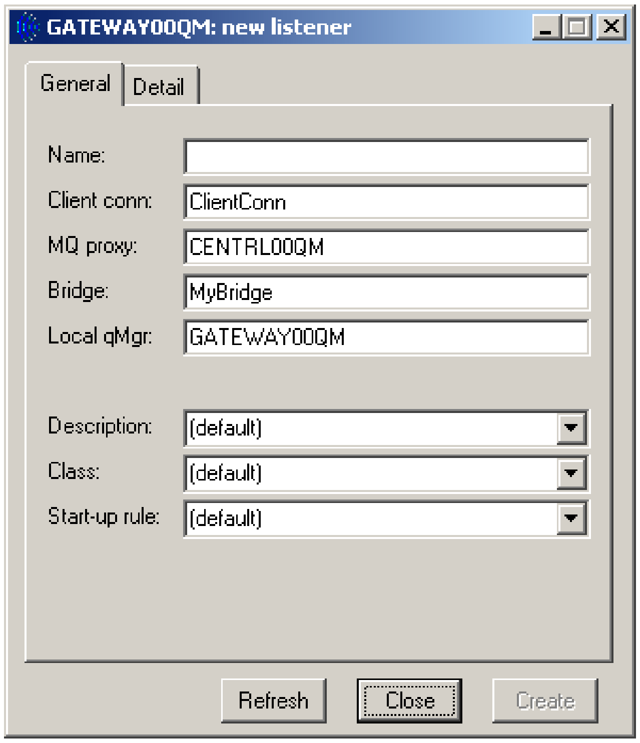
- *General* tab:
  - *Name* – the name of the new connection (and therefore the name – or an alias – of the remote queue manager to be connected).
  - *Local queue manager* – the name of the queue manager owing this definition.
  - *Description* – any UNICODE string.
  - *Type* – connection type (see above).
  - *Channel class (or alias)* – the class that handles data transfers to the remote queue manager.
    - For a remote direct connection: *(default)* is mapped to " com.ibm.mqe.communications.MQeChannel".
    - For an Alias/MQ connection: *(default)* is mapped to null.
- *Detail* tab:
  - *Address* – the numeric or string IP address of the machine hosting the remote queue manager (available only for direct connections).

    - *Via queue manager* – the name of the queue manager through which the connection should be routed (available only for indirect connections).
    - *Port* – the IP port number used by the remote queue manager to service incoming connection requests (available only for direct connections).
    - *Adapter class (or alias)* – the class of the communications protocol adapter (available only for direct connections). Each protocol adapter supports a specific IP protocol, for example HTTP, TCP or UDP; multiple adapters may exist per protocol with differing performance or other characteristics.
        - *(default)* is mapped to "com.ibm.mqe.adapters.MQeTcpipHistoryAdapter".
    - *Options* – a series of ASCII options to be passed to the communications protocol adapter (available only for direct connections). The format for specifying options is to enclose each option value within angled brackets.
        - *(default)* is mapped to "<PERSIST><HISTORY>" when the "com.ibm.adapters.MQeTcpipHistoryAdapter" adapter class or the alias "FastNetwork" is specified; otherwise *(default)* is mapped to "".
        - *(none)* is mapped to "".
    - *Ascii parameters* – an ASCII parameter string to be passed to the adapter; for example, for an HTTP connection - the name of the servlet (available only for direct connections).
        - *(default)* and *(none)* are both mapped to "".
- *Aliases tab:*
    - *Alias names* – alias names for this connection. The *Add* button adds additional alias names entered in the input field; the *Delete* button deletes selected alias names.

The *Create* button creates the new connection and updates the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window. *Work offline* allows connection definitions to be created on remote queue managers that are not currently available on the network.

Listener

File→New→Listener enables the *New listener* definition dialog:



Name:	
Client conn:	ClientConn
MQ proxy:	CENTRL00QM
Bridge:	MyBridge
Local qMgr:	GATEWAY00QM
Description:	(default)
Class:	(default)
Start-up rule:	(default)

Figure 4-5: New listener dialog

The dialog presents tabs with the listener characteristics. These tabs are shared with the *Listener properties* dialog, used to change an existing listener.

The following input fields may be present:

- *General* tab:
  - *Name* – the name of the new listener.
  - *Client connection* – the name of the client connection owning this definition.
  - *MQ Proxy* – the name of the MQ proxy owning this definition.
  - *Bridge* – the name of the bridge owning this definition.
  - *Local queue manager* – the name of the queue manager owing this definition.
  - *Description* – any UNICODE string.
  - *Listener class (or alias)* – the class that implements the client connection.
    - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeListener".

    - *Startup rule class (or alias)* – the class that determines whether the listener and its parents are started at object creation time or when the owning queue manager is opened.
        - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeStartupRule".
        - *(none)* is mapped to "" (i.e. no rule class). Objects are created in the stopped state.
    - *Status* – indicates whether the listener is in the stopped or started state.

- *Detail* tab:
    - *Dead letter queue* – the WebSphere MQ queue used to hold messages that cannot be delivered from WebSphere MQ to MQe.
        - *(default)* is mapped to "SYSTEM.DEAD.LETTER.QUEUE".
    - *State store* – specifies the permanent storage used to hold state information as messages are moved from WebSphere MQ to MQe. The storage can be defined on either MQe or WebSphere MQ.

      If MQe is to be used the value has the format of an MQe storage adapter, followed by a colon, followed by a path name to either a file or a directory. If a directory is specified then a file will be created there with the name *<listener name>-listener.sta*. If no parameter is specified a file will be created in the current working directory.

      If WebSphere MQ is to be used then the MQ adapter should be specified with no parameters (i.e. "com.ibm.mqe,mqbridge.MQeMQAdapter").
        - *(default)* is mapped to "com.ibm.mqe.adapters.MQeDiskFieldsAdapter".
    - *Undelivered rule class (or alias)* – the class determining the action taken when a message cannot be delivered from WebSphere MQ to MQe.
        - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeUndeliveredMessageRule".
    - *Transformer class (or alias)* – the class defining the transformation used for message conversion for messages from WebSphere MQ to MQe).
        - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeBaseTransformer".
        - *(null)* is mapped to a null value (i.e. use the default transformer defined in the bridge).
    - *Flows per commit* – the number of messages flowed after which the MQ sync. queue (if used) is cleaned.

The *Create* button creates the new listener and updates the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Message

File→New→Message enables the *Send test message* dialog. By default it will use a message of class *com.ibm.mqe.MQeMsgObject*; optionally it will use the *com.ibm.mqe.mqemqmessage.MQeMQMsgObject* class instead.

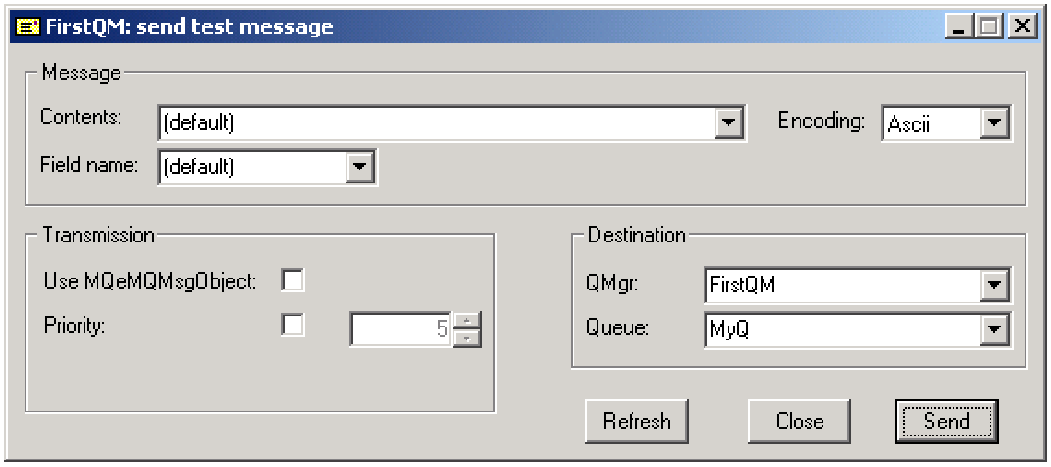


Figure 4-6: New message dialog

The following input fields are present:

- *Message* group:
  - *Message* – the text string of the message.
    - *(default)* generates a message that automatically includes the name of the sending queue manager, the date and time.
  - *Field name* – an ASCII string specifying name of the field containing the message text (only applicable when the message class is *com.ibm.mqe.MQeMsgObject*).
    - *(default)* is mapped to the string "Message".
  - *Encoding* – the encoding to be used (Ascii or Unicode).
    - *(default)* is mapped to "ASCII".
- *Destination* group:
  - *Queue manager* – the name of the destination queue manager. This may be entered or selected from the drop-down list. Any name must identify a queue manager to which addressability exists from the queue manager hosting MQe_Explorer, i.e. it must be one of the following: the name of the hosting queue manager (or an alias); a connection name (or an alias); a destination name owned by a *store* or *forward queue* (or an alias).
  - *Queue* – the name of the target queue. This may be entered or selected from the drop-down list. Any name entered must either be known to the queue manager hosting MQe_Explorer (in conjunction with the target queue manager name) as a queue name (or alias), or otherwise will be assumed to identify a synchronous proxy queue.

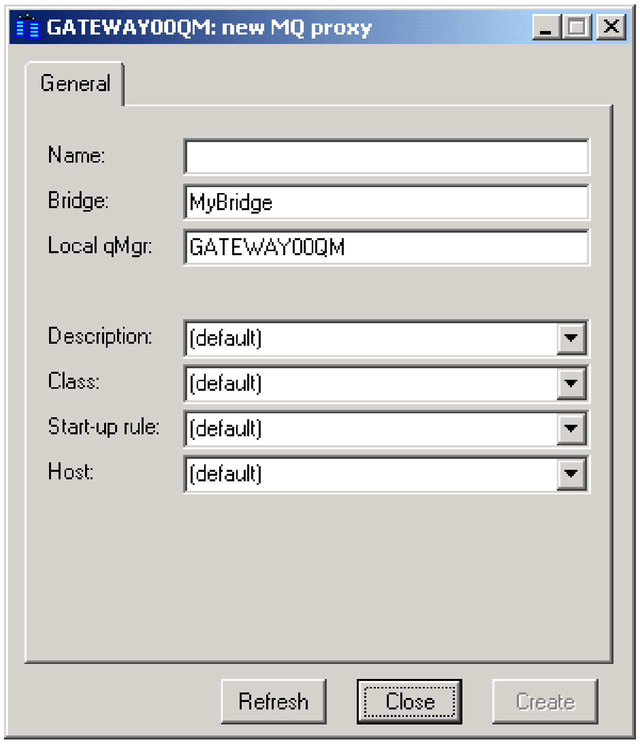
- *Transmission* group:
  - *Use MQeMQMsgObject* – the message object class used is *com.ibm.mqe.mqemqmessage.MQeMQMsgObject* (this class may be useful when sending to MQ queue managers, as it is supported by the default bridge transformer).
  - *Priority* – if the box is unchecked the message will not contain a priority field; if checked a priority field with the selected priority (0 – 9) is added to the message.

The test message will include message expiry; the expiry time is set to be the same as the online admin message expiry (see *Tools→Options→Advanced-2*).

The *Send* button sends the new message; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

MQ Proxy

File→New→MQ Proxy enables the *New MQ proxy* definition dialog that creates a proxy for an MQ queue manager:


GATEWAY00QM: new MQ proxy

Name:
Bridge: MyBridge
Local qMgr: GATEWAY00QM
Description: (default)
Class: (default)
Start-up rule: (default)
Host: (default)

Figure 4-7: New MQ proxy dialog

The dialog presents a single tab with the MQ proxy characteristics. The tab is shared with the *MQ proxy properties* dialog, used to change an existing MQ proxy.

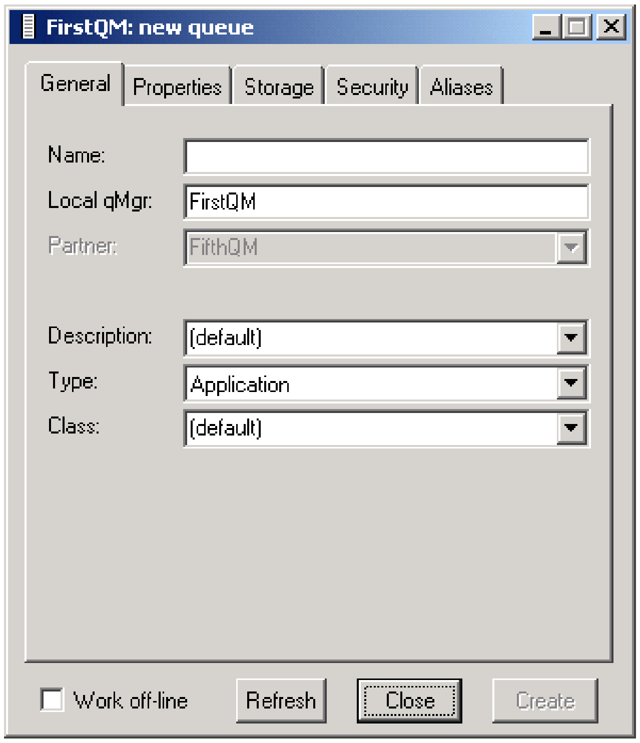
The following input fields may be present:

- *General* tab:
  - *Name* – the name of the new MQ proxy.
  - *Bridge* – the name of the bridge owning this definition.
  - *Local queue manager* – the name of the queue manager owing this definition.
  - *Description* – any UNICODE string.
  - *MQ proxy class (or alias)* – the class that implements the MQ proxy.
    - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeMQQMgrProxy".
  - *Startup rule class (or alias)* – the class that determines whether the MQ proxy and its parents are started at object creation time or when the owning queue manager is opened.
    - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeStartupRule".
    - *(none)* is mapped to "" (i.e. no rule class). Objects are created in the stopped state.
  - *Host* – the IP name or numeric address of the WebSphere MQ queue manager. If "" is specified then the interface to WebSphere MQ uses bindings mode, otherwise client mode.
    - *(default)* is mapped to "".
  - *Status* – indicates whether the MQ proxy is in the stopped or started state.

The *Create* button creates the new MQ proxy and updates the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Queue

File→New→Queue enables the *New queue* definition dialog:



Name:
Local qMgr: FirstQM
Partner: FifthQM
Description: (default)
Type: Application
Class: (default)

Figure 4-8: New queue dialog

Various types of queue can be created:

- *Admin*

  An admin queue is a queue that accepts admin messages and processes them, executing their instructions. It may, if required, send replies to these messages. All queue managers should normally have one admin queue defined; otherwise the queue manager itself cannot be configured. The admin queue is normally created when the queue manager is created (see *Tools→Options→Advanced-1*). It is a convention that the queue is named 'AdminQ'. If offline administration is also required, conventionally the admin queue is given an alias name of 'OfflineAdminQ'. The contents of admin queues are not accessible to applications.

- *Application*[7]

  An application queue stores messages with its contents accessible to applications. The number of messages stored on the queue is available as the *current depth* property.

- *Bridge*

  A bridge queue represents a remote queue on a WebSphere MQ queue manager; such queues are only defined on MQe gateways.

[7] Application queues are termed *local* queues in the MQe documentation and are normally created through the use of a *com.ibm.mqe.MQeQueueAdminMsg*.

- *Home server*

  A home server queue points to a store queue on a remote queue manager and pulls messages from that queue destined for its local queue manager, delivering them to the appropriate local application queues. The remote queue manager is known as the 'get from' queue manager. Home server queues should be regarded as system queues – messages are never addressed to home server queues and they do not hold messages in transit.

- *Async. proxy*[8]

  An asynchronous proxy queue is a proxy for an application queue on a remote queue manager. This remote queue manager is known as the *destination queue manager*. The asynchronous proxy sends messages to the remote application queue; if the messages cannot be sent immediately then they are temporarily stored locally until the proxy can send them. Applications are not able to view messages awaiting transmission on an asynchronous proxy queue; however the number of messages awaiting transmission is available as the *current depth* property.

- *Sync. proxy*[9]

  A synchronous proxy queue is a proxy for an application queue on a remote queue manager elsewhere in the MQe network. This remote queue manager is known as the *destination queue manager*. The synchronous proxy sends messages to the remote application queue; if the messages cannot be sent immediately then an exception is returned to the application. Synchronous proxy queues do not have the ability to store messages awaiting transmission; the application is responsible for messages that cannot be delivered.

- *Store*[10]

  A store queue collects messages addressed to remote queue managers. A single store queue can collect messages for one or more of these *destination queue managers*. Store queues have the collected messages pulled from them by home server queues at the destinations. Store queues should be regarded as system queues – messages are never addressed to them and their contents are not accessible to applications. The number of messages awaiting collection is available as the *current depth* property.

- *Forward*[11]

  A forward queue collects messages addressed to remote queue managers and forwards them to the next hop queue manager, known as the 'forward to' queue manager. A single forward queue can collect

---

[8] Async. proxy queues are *remote* queues with a mode property of asynchronous in the MQe documentation and are normally created through the use of a *com.ibm.mqe.MQeRemoteQueueAdminMsg.*

[9] Sync. proxy queues are *remote* queues with a mode property of synchronous in the MQe documentation and are normally created through the use of a *com.ibm.mqe.MQeRemoteQueueAdminMsg.*

[10] Store queues are *storeAndForward* queues in the MQe documentation where the queue queue manager name has been set to the name of the local queue manager; they are created through a *com.ibm.mqe.MQeStoreAndForwardQueueAdminMsg.*

[11] Foward queues are *storeAndForward* queues in the MQe documentation where the queue queue manager name has been set to the next hop queue manager name; they are created through a *com.ibm.mqe.MQeStoreAndForwardQueueAdminMsg.*

messages for one or more remote queue managers – referred to as *destination queue managers*. Forward queues should be regarded as system queues – messages are never addressed them and their contents are not accessible to applications. The number of messages awaiting transmission is available as the *current depth* property.

The dialog presents a number of different tabs, each relating to a related set of queue characteristics. The tabs are shared with the *Queue properties* dialog, used to change an existing queue. The complete set of tabs is:

- *General*

  The name of the queue, local queue manager and other properties supported by most queue types.

- *Properties*

  Additional detailed properties of a queue, their relevance depending upon queue type.

- *Storage*

  Properties related to the storage of messages; only relevant to certain queue types.

- *Bridge*

  Properties related to bridge queues.

- *Security*

  Queue security properties.

- *Certificates*

  Details of the WTLS mini-certificates stored in the registry relevant to the queue.

- *Destinations*

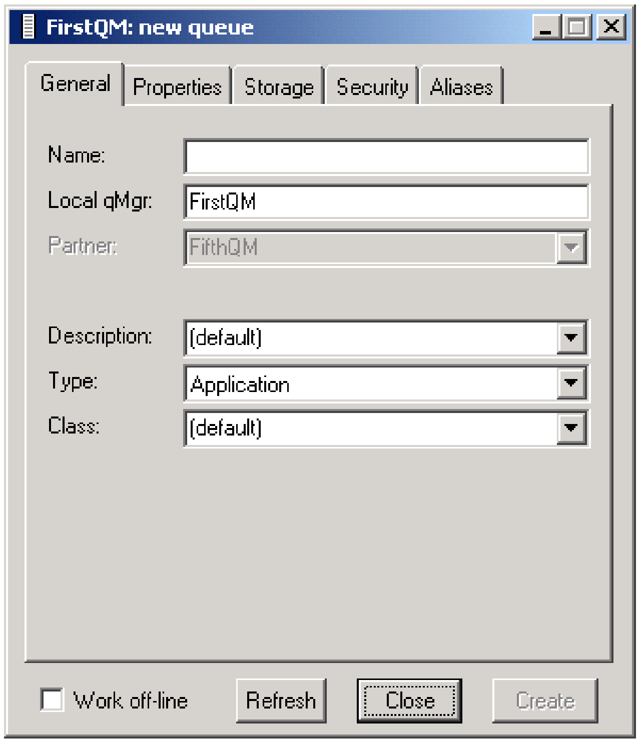
  A list of destination queue manager names (applicable only to store and forward queues).

- *Aliases*

  A list of alternative names that should be resolved to this queue.

The following input fields may be present; note that the availability of some properties depend upon the type of queue, whether the queue is new or already exists, and the values of other properties:

The *General* tab has the appearance:


FirstQM: new queue

Name:	
Local qMgr:	FirstQM
Partner:	FifthQM
Description:	(default)
Type:	Application
Class:	(default)

Figure 4-9: New queue dialog – General tab

The following input fields may be present:

- *General* tab:
    - *Name* – the name of the new queue.
    - *Local queue manager* – the name of the queue manager owing this definition.
    - *Destination, Get from, Forward to* – partner queue manager depending upon whether the queue type.
    - *Description* – any UNICODE string.
    - *Type* – the type of queue.

- *Class (or alias)* – the class that implements the queue.
  - *(default)* is mapped as follows, depending upon the type of queue:

    Async. proxy: "com.ibm.mqe.MQeRemoteQueue"

    Admin: "com.ibm.mqe.MQeAdminQueue"

    *Application: "*com.ibm.mqe.MQeQueue"

    Bridge:
    "com.ibm.mqe.mqbridge.MQeMQBridgeQueue"

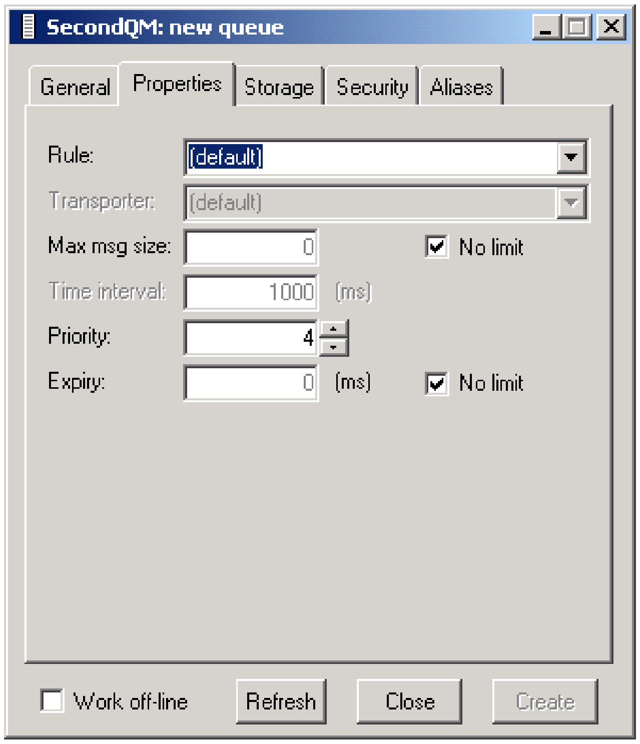
    Forward:
    "com.ibm.mqe.MQeStoreAndForwardQueue"

    Home server:
    "com.ibm.mqe.MQeHomeServerQueue"

    Store: "com.ibm.mqe.MQeStoreAndForwardQueue"

    Sync. proxy: "com.ibm.mqe.MQeRemoteQueue"
- *Created* – the date/time that this queue was created.

The *Properties* tab has the appearance:



Rule: (default)
Transporter: (default)
Max msg size: 0 [✓] No limit
Time interval: 1000 (ms)
Priority: 4
Expiry: 0 (ms) [✓] No limit

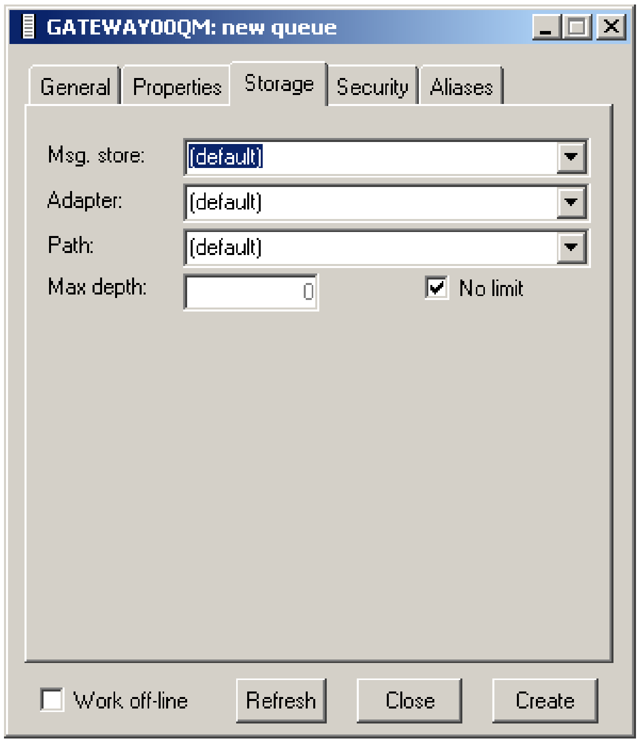
Figure 4-10: New queue dialog – Properties tab

The following input fields may be present:

- *Properties* tab:
  - *Rule class (or alias)* – the rule class to be associated with the queue.
    - *(default)* is mapped to "com.ibm.mqe.MQeQueueRule".
    - *(null)* is mapped to a null value (i.e. no rule class).

- *Transporter class (or alias)* – the transporter class to be used (handles the getting and putting of messages to/from remote queues).
    - *(default)* is mapped to "com.ibm.mqe.MQeTransporter".
- *Maximum message size* – the maximum message size property, used by the queue rule. It must be a positive integer. The *No limit* check box takes priority and allows messages of unlimited size.
- *Time interval* – the timer interval in milliseconds.
    - For an admin queue, a positive integer denotes the time between successive attempts, where an attempt has failed because resources are unavailable. Admin messages that cannot be actioned will always be retried when the queue manager is re-opened.
    - For a home server queue, a positive integer denotes the time between the getting of pending messages. If a value of zero is specified then messages will only be got when a trigger transmission event occurs (subject to the queue manager rule).
- *Priority* – the default priority associated with messages on the queue; the value must be a positive integer in the range 0 – 9.
- *Expiry* – the time in milliseconds after which messages on the queue expire, and are then removed. A value of zero indicates that messages never expire.
- *Status* – indicates whether the queue is active or inactive.

The *Storage* tab has the appearance:



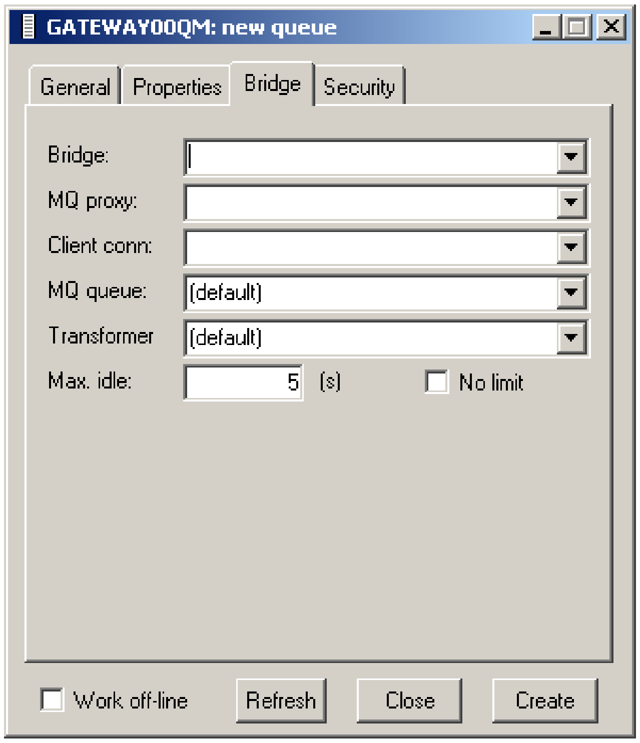
Msg. store: (default)
Adapter: (default)
Path: (default)
Max depth: 0 [✓ No limit]
[ ] Work off-line

Figure 4-11: New queue dialog – Storage tab

The following input fields may be present:

- *Storage* tab:
    - *Message store (or alias)* – the message store class that determines the way that the queue adapter stores messages. If the value *(default)* is used for the *message store*, the *adapter* and the *path*, then the default message store, adapter and path associated with the queue manager is used; otherwise *(default)* is mapped to "com.ibm.mqe.messagestore.MQeMessageStore".
    - *Adapter (or alias)* – the queue adapter class that determines the backing store associated with the queue. If the value *(default)* is used for the *message store*, the *adapter* and the *path*, then the default message store, adapter and path associated with the queue manager is used; otherwise *(default)* is mapped to "com.ibm.mqe.adapters.MQeDiskFieldsAdapter".
    - *Path* – the path locating the physical storage of the queue – passed to the queue adapter. If the value *(default)* is used then the default path associated with the queue manager is used.
    - *Maximum queue depth* – the maximum number of messages allowed on the queue, used by the queue rule. The *No limit* check box takes priority and allows an unlimited number of messages.
    - *Current depth* – for an active queue only this represents the number of messages on the queue. For a store queue this indicates the number of messages awaiting collection; for a forward or async. proxy queue, the number awaiting transmission.

The *Bridge* tab has the appearance:


GATEWAY00QM: new queue

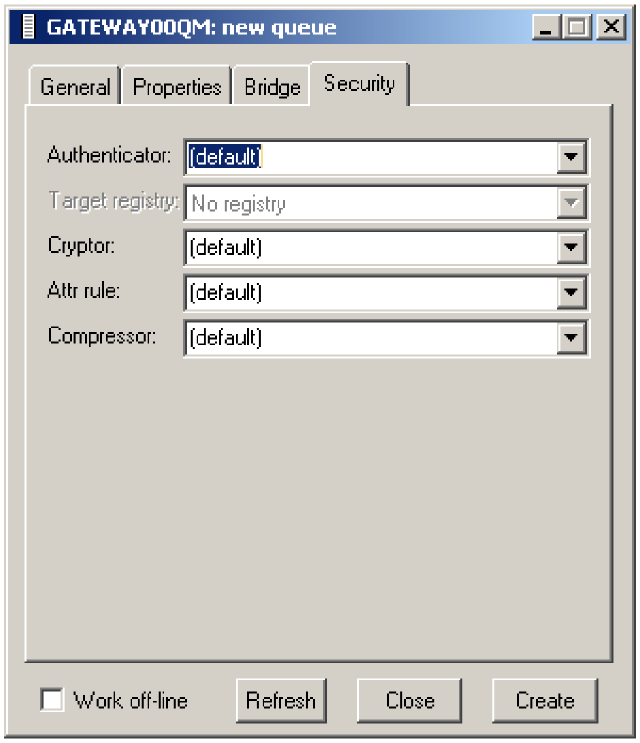
Bridge:
MQ proxy:
Client conn:
MQ queue: (default)
Transformer (default)
Max. idle: 5 (s) No limit

Figure 4-12: New queue dialog – Bridge tab

The following input fields may be present:

- *Bridge* tab:
    - *Bridge* – the bridge associated with the queue.
    - *MQ proxy* – the name of the WebSphere MQ queue manager associated with the queue.
    - *Client connection* – the name of the client connection associated with the queue.
    - *MQ queue* – the name of the WebSphere MQ queue associated with the queue.
        - *(default)* indicates that the queue name on the WebSphere MQ queue manager has the same name as this bridge queue definition.
    - *Transformer class (or alias)* – the class that transforms messages being sent from MQe to WebSphere MQ.
        - *(default)* is mapped to "com.ibm.mqe.mqbridge.MQeBaseTransformer".
        - *(null)* is mapped to a null value (use default transformer defined in the MQ bridge).
    - *Max. idle* – the maximum time (in seconds) that a bridge queue is allowed to hang on to an idle connection before it is returned to the connection pool. The value must be zero or a positive integer; the default is 5.

The *Security* tab has the appearance:



Authenticator: (default)
Target registry: No registry
Cryptor: (default)
Attr rule: (default)
Compressor: (default)

Figure 4-13: New queue dialog – Security tab

The following input fields may be present:

- *Security* tab*:*
  - *Authenticator class (or alias)* – the authenticator class associated with the security attribute assigned to the queue.
    - *(default)* is mapped to a null value.
    - *(null)* is mapped to a null value.

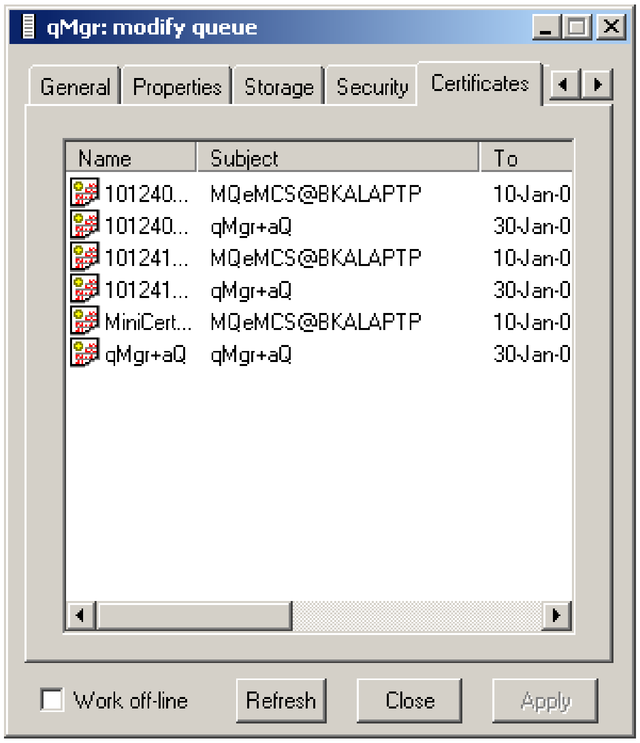
    If *com.ibm.mqe.attributes.MQeWTLSCertAuthenticator* is specified, mini-certificate based authentication will be used. The certificate used depends upon the value of the *target registry* property. If queue-based authentication is selected this will cause auto-registration with the MQe mini-certificate server. The mini-certificate server requires prior set-up such that it is allowed to issue a mini-certificate to the queue. Certificates are stored in the private registry and have a pre-determined lifetime, set when issued. They can be renewed through the *Actions→Renew Credentials* menu item[12].

    For more details see also the *Security* tab of the *Queue Manager* on page 42.
  - *Target registry* – the target registry to be used by the authenticator. If queue registries are enabled, it has one of values: “None”, “Queue”, “Queue manager”; otherwise the 'Queue' value is not valid.
    - For the *com.ibm.mqe.attributes.MQeWTLSCertAuthenticator* class, *(default)* is mapped to “Queue manager”; for all others it is mapped to "None".

    For more details see also the *Security* tab of the *Queue Manager* on page 42.
  - *Cryptor class (or alias)* – the cryptor class associated with the security attribute assigned to the queue.
    - *(default)* is mapped to a null value.
    - *(null)* is mapped to a null value.
  - *Attribute rule class (or alias)* – the rule class associated with the security attribute assigned to the queue.
    - *(default)* is mapped to " com.ibm.mqe.MQeAttributeDefaultRule ".
  - *Compressor class (or alias)* – the compressor class associated with the security attribute assigned to the queue.
    - *(default)* is mapped to a null value.
    - *(null)* is mapped to a null value.

[12] For more details of MQe security and auto-registration see the *WebSphere MQ Everyplace Configuration Guide*.

The *Certificates* tab has the following appearance:



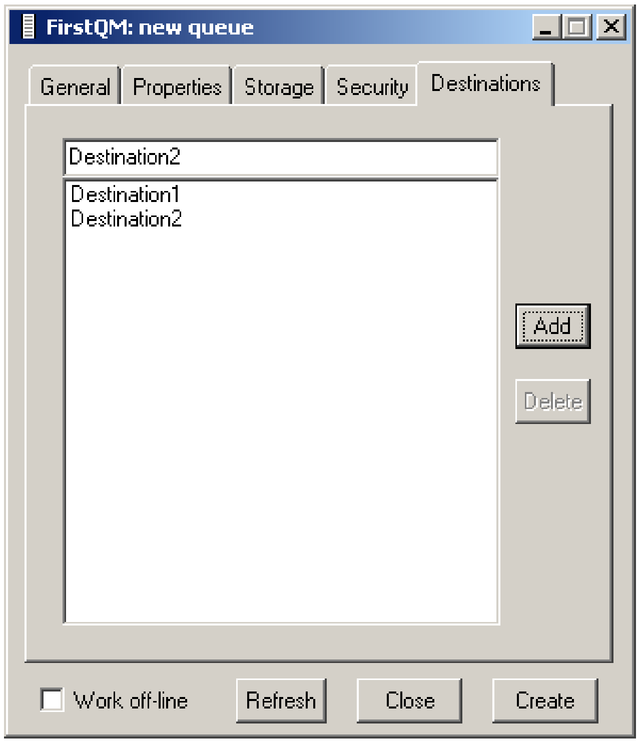
Name	Subject	To
101240...	MQeMCS@BKALAPTP	10-Jan-0
101240...	qMgr+aQ	30-Jan-0
101241...	MQeMCS@BKALAPTP	10-Jan-0
101241...	qMgr+aQ	30-Jan-0
MiniCert...	MQeMCS@BKALAPTP	10-Jan-0
qMgr+aQ	qMgr+aQ	30-Jan-0

Figure 4-14: Modify queue dialog – Certificates tab

- *Certificates* tab:
  - *Name* – the name of the certificate[13]. Names that start with a number are old certificates that have been renamed.
  - *Subject* – the name of the queue authenticatable entity that owns the certificate.
  - *To* – the date after which the certificate is invalid.
  - *From* – the date from which the certificate is valid.
  - *Issuer* – the name of the entity issuing the certificate.

[13] MQe_Explorer removes the suffix "_MiniCertificate".

The *Destinations* tab has the appearance:



Destination2

Destination1
Destination2

Figure 4-15: New queue dialog – Destinations tab

The following input fields are present:

- *Destinations* tab*:*
  - *Destination queue managers* – applicable only to *store* and *forward* queues and specifies the list of queue manager names for which this queue will collect messages. The *Add* button adds additional destinations entered in the input field; the *Delete* button deletes selected destinations.

The *Aliases* tab has the appearance:

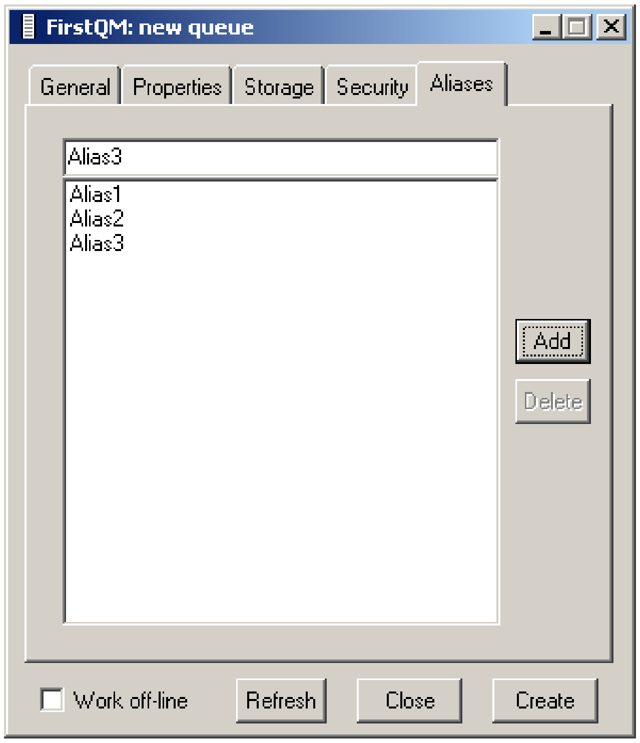


Figure 4-16: New queue dialog – Aliases tab

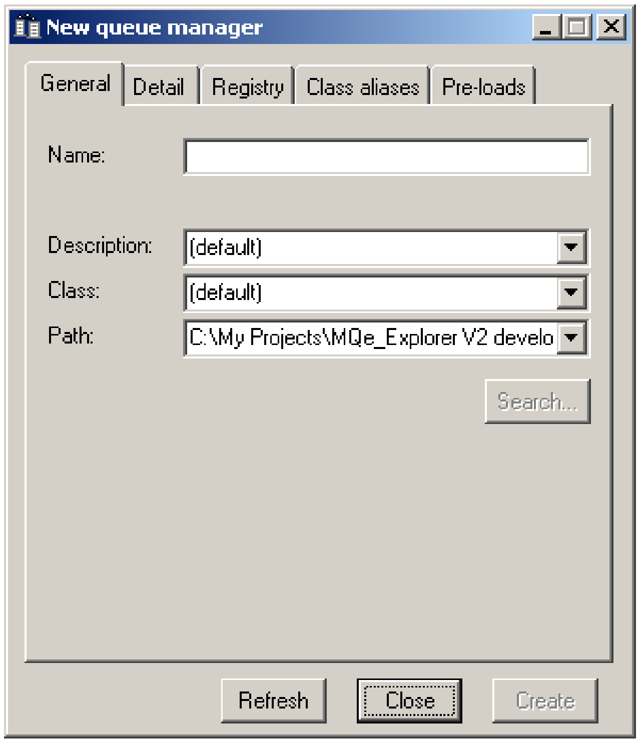
The following input fields are present:

- *Aliases* tab*:*
  - *Alias names* – alias names for this queue. The *Add* button adds additional alias names entered in the input field; the *Delete* button deletes selected alias names.

The *Create* button creates the new queue and updates the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window. *Work offline* allows queue definitions to be created on remote queue managers that are not currently available on the network.

Queue Manager

File→New→Queue Manager enables the *New queue manager* dialog that allows local queue managers to be created.



Name:

Description: (default)

Class: (default)

Path: C:\My Projects\MQe_Explorer V2 develo

Figure 4-17: New queue manager dialog

The dialog presents a number of different tabs, each relating to a related set of queue manager characteristics. The tabs are shared with the *Queue manager properties* dialog, used to change an existing queue manager. The complete set of tabs is:

- *General*

  The name of the queue manager, description, class and the path of the associated initialization file.

- *Summary*

  A high level view of the objects owned by the queue manager, e.g. queues, connections, comms. listeners, etc.

- *Detail*

  A number of detailed properties of the queue manager, e.g. channel time-out, channel attribute rule classes etc.

- *Aliases*

  Alternative names that should be resolved to this queue manager.

- *Registry*

  Details of the queue manager registry type, including the registry adapter.

- *Security*

  Security data associated with the registry.

- *Certificates*

  Details of queue manager WTLS mini-certificates stored in the registry.
- *Class aliases*

  Alternative names that can be used locally to reference Java classes.
- *Preloads*

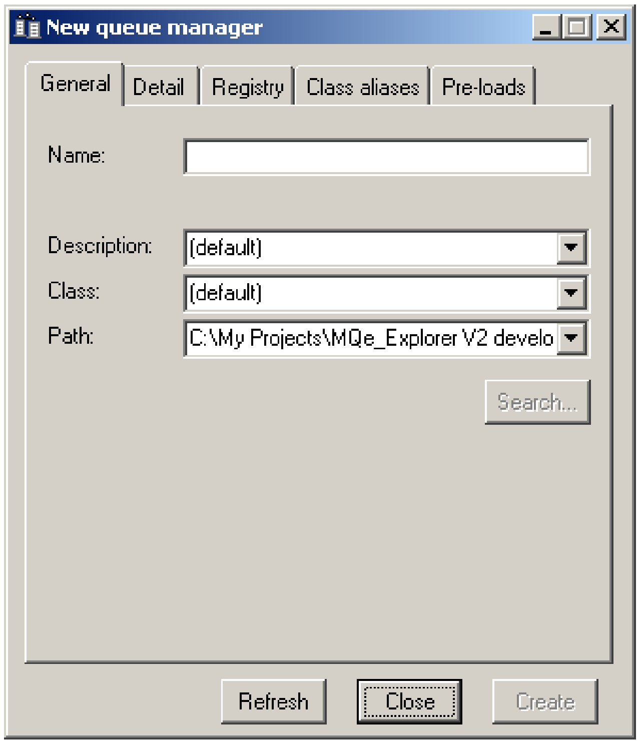
  Java classes to be pre-loaded when the queue manager is started.

The tabs that are available at any time depend upon many factors, including:

- Whether creating or modifying a queue manager
- Whether MQe_Explorer is running as a slave or a master.
- Whether the queue manager is local or remote.
- Whether online or offline administration is active.
- The nature of the registry.

Queue managers are always created as *clients*, i.e. by default they do not listen for incoming connection requests from other queue managers, but can themselves initiate such requests. Such client queue managers can be subsequently configured into *servers*, by adding communications listeners (*File→New→Comms. Listener*); furthermore they can be given *gateway* capability by configuring a bridge and its various child objects *(File→New→Bridge*)

The *General* tab has the appearance:



Name:

Description: (default)

Class: (default)

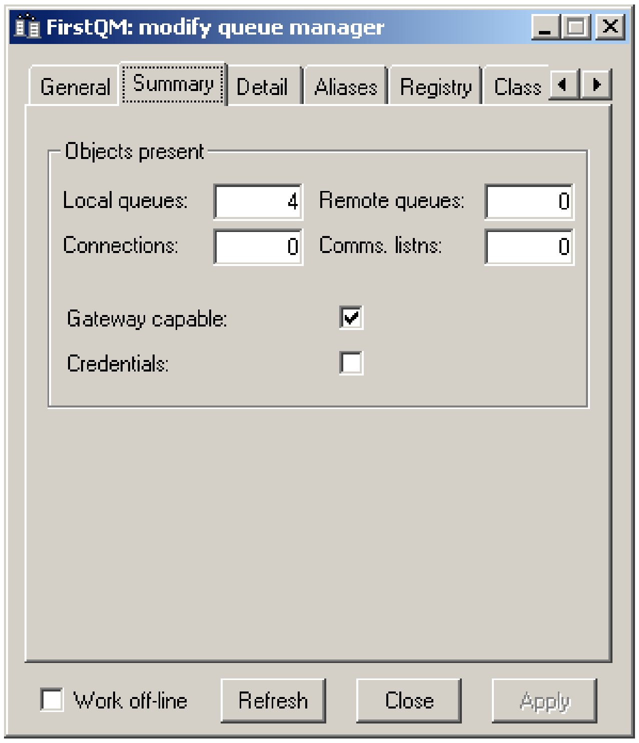
Path: C:\My Projects\MQe_Explorer V2 develo

Figure 4-18: New queue manager dialog – General tab

The following input fields may be present:

- *General* tab:
  - *Name* – the name of the queue manager.
  - *Description* – any UNICODE string.
  - *Class (or alias)* – the class of the queue manager.
    - *(default)* is mapped to “com.ibm.mqe.MQeQueueManager”.
  - *Path* – the location for the initialization file. The path can include the file name and type. If the type is omitted it will be set to *.ini*; if the file name is omitted a default will be used. The *Search* button can be used to locate a path or a file – after a queue manager name is present.

The *Summary* tab is only displayed for an existing queue manager:


FirstQM: modify queue manager

General | Summary | Detail | Aliases | Registry | Class

Objects present

Local queues: 4
Remote queues: 0
Connections: 0
Comms. listns: 0

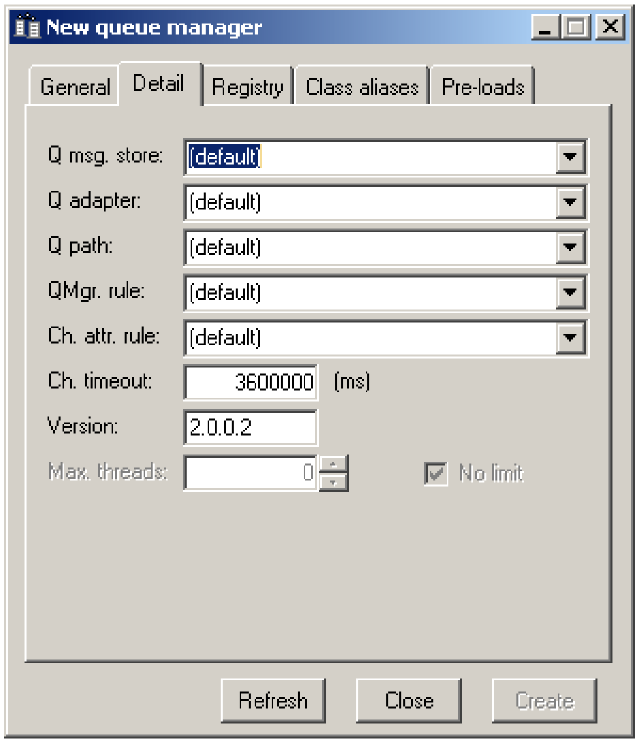
Gateway capable: ☑
Credentials: ☐

☐ Work off-line

Figure 4-19: Modify queue manager dialog – Summary tab

- *Summary* tab:
  - *Local queues* – the number of local queue defined.
  - *Remote queues* – the number of remote queue defined.
  - *Connections* – the number of connection defined.
  - *Comms. listeners* – the number of communications listeners defined.
  - *Gateway capable* – if checked, the queue manager has access to the MQe bridge classes and is capable of hosting a gateway.
  - *Credentials* – if checked, the registry contains queue manager mini-certificate credentials.

The *Detail* tab has the following appearance:



Q msg. store: (default)
Q adapter: (default)
Q path: (default)
QMgr. rule: (default)
Ch. attr. rule: (default)
Ch. timeout: 3600000 (ms)
Version: 2.0.0.2
Max. threads: 0 [✓ No limit]

Figure 4-20: New queue manager dialog – Detail tab

- *Detail* tab:
  - *Queue message store (or alias)* – the default message store class that determines the way that the queue adapter stores messages.
    - *(default)* is mapped to "com.ibm.mqe.messagestore.MQeMessageStore".
  - *Queue adapter (or alias)* – the default queue adapter class.
    - *(default)* is mapped to "com.ibm.mqe.adapters.MQeDiskFieldsAdapter".
  - *Queue path* – the default path locating the physical storage of queues – passed to the queue adapter.
    - *(default)* is mapped to <*initialization file path*>\ <*qMgr name*>\Queues\.
  - *Queue manager rule class (or alias)* – the rule class to be used with the queue manager.
    - *(default)* is mapped to "com.ibm.mqe.MQeQueueManagerRule".
    - *(null)* is mapped to a null value (i.e. no rule).
  - *Channel attribute rule class (or alias)* – the rule class to be associated with the channel attribute.
    - *(default)* is mapped to "com.ibm.mqe.MQeAttributeDefaultRule".

- *Channel timeout* – the time in milliseconds after which an idle outgoing connection will timeout. This value must be zero (no timeout) or a positive integer; the default value is 3,600,000.
- *Version* – the version of MQe hosting the queue manager (releases before 2.002 will display “Unknown”).
- *Max threads* – the maximum number of threads that will be spawned to service the transmission needs of the queue manager. Each thread is used to service transmission to one destination at any one time. If the value is zero the queue manager background thread will be used for all transmissions. If *No limit* is checked then one thread per destination will be used.

The *Aliases* tab is only displayed for an existing queue manager; it has the following appearance:

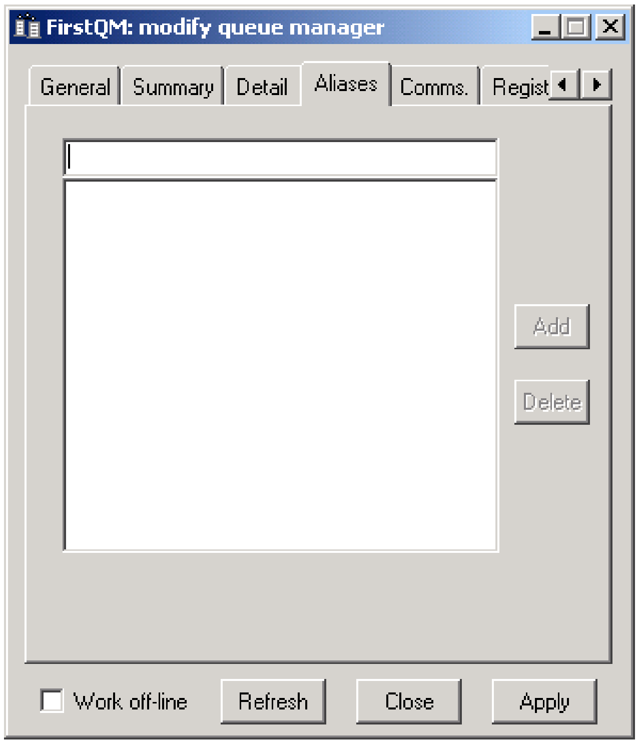
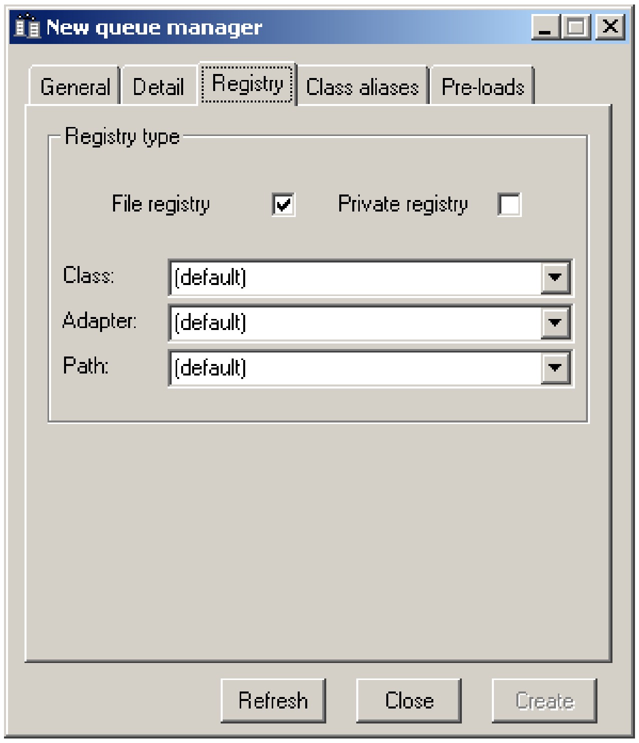


Figure 4-21: New queue manager dialog – Aliases tab

- *Aliases* tab:
  - *Alias names* – alias names for this queue manager. The *Add* button adds additional alias names entered in the input field; the *Delete* button deletes selected alias names.

The *Registry* tab has the following appearance:



Registry type

File registry ✓ Private registry ☐

Class: (default)

Adapter: (default)

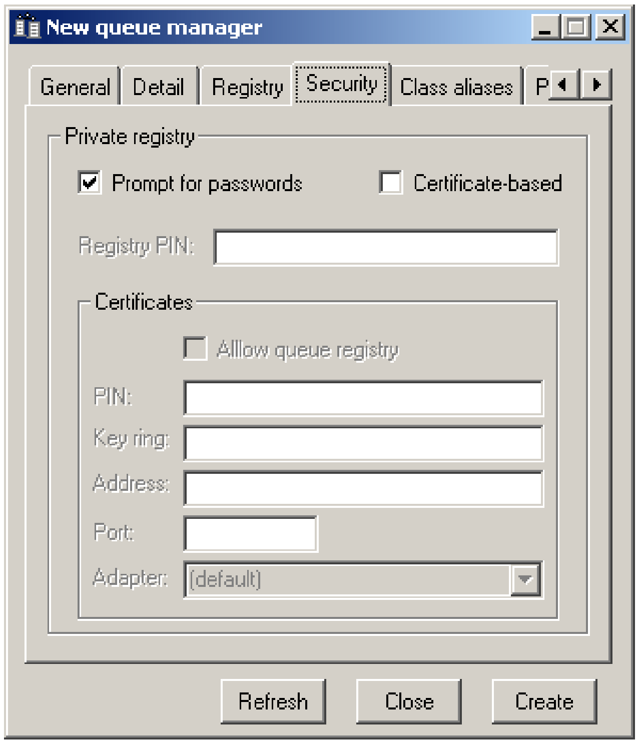
Path: (default)

Figure 4-22: New queue manager dialog – Registry tab

- *Registry* tab:
    - *File registry* – checking *File registry* creates an unprotected registry (for more details see below).
    - *Private registry* – checking *Private registry* creates a protected registry – the details are set on the *Security* tab (for more details see below).
    - *Class (or alias)* – the registry class.
        - *(default)* is mapped to "com.ibm.mqe.registry.MQeFileSession" for file registries, and to "com.ibm.mqe.registry.MQePrivateSession" for private registries.
    - *Registry adapter (or alias)* – the class of the adapter used to map the registry into storage.
        - *(default)* is mapped to "com.ibm.mqe.adapters.MQeDiskFieldsAdapter".
    - *Path* – the registry location.
        - *(default)* is mapped to <*initialization file path*>\<*qMgr name*>\Registry\.

A *file registry* is unprotected; a *private registry* has PIN-controlled access. Note that the use of a private registry is a prerequisite to using any form of mini-certificate based authentication based on the *com.ibm.mqe.attributes.MQeWTLSCertAuthenticator* class. The private registry by itself does not protect the queues (and their messages) in any way; queues and messages are protected through the use of either queue-based security (using cryptors and authenticators) or message-based security.

The *Security* tab has the following appearance:



[✓] Prompt for passwords
[ ] Certificate-based
Registry PIN:
[ ] Allow queue registry
PIN:
Key ring:
Address:
Port:
Adapter: (default)

Figure 4-23: New queue manager dialog – Security tab

- *Security* tab:
    - *Prompt for passwords* – specifies that PINs and passwords should not be stored in the initialization file but requested from the user when required.
    - *Certificate-based*[14] – specifies that mini-certificate based authentication be enabled. This will cause auto-registration with the MQe mini-certificate server when the queue manager is created (or re-started). The mini-certificate server requires prior set-up such that it is allowed to issue a mini-certificate to the named queue manager. Certificates are stored in the private registry and have a pre-determined lifetime, set when issued. They can be renewed through the *Actions→Renew Credentials* menu item.
    - *Registry PIN* – the PIN for private registry access (for more details see below).

[14] For more details of MQe security and mini-certificates, see the *WebSphere MQ Everyplace Configuration Guide*.

- *Certificates panel: Allow queue registry* – if checked, allows queues to have their own credentials; if unchecked, queues must use the credentials of the queue manager (for more details see below).
- *Certificates panel: PIN* – the certificate request PIN for the mini-certificate server (for more details see below).
- *Certificates panel: Key ring* – the password used to protect the registry's private key (for more details see below).
- *Certificates panel: address* – the IP address of the mini-certificate server.
- *Certificates panel: port* – the IP port number of the mini-certificate server.
- *Certificates panel: Adapter* – the communications adapter class (or alias) to be used to communicate with the mini-certificate server.
    - *(default)* is mapped to "com.ibm.mqe.adapters.MQeTcpipHttpAdapter".

The *Security* tab is only enabled for queue managers with a private registry. The *registry PIN* controls access to the private registry; the *key ring password* controls access to the private key within the registry. Checking the *certificates-based* security box ensures that mini-certificates will be used for authentication and is a prerequisite to any use of authentication based on the *com.ibm.mqe.attributes.MQeWTLSCertAuthenticator* class. The *certificate PIN* is used in the request to the mini-certificate server when a certificate is required. If the *allow queue registry* box is unchecked (the default setting), queues are not allowed their own credentials, but must use those of the queue manager. If *prompt for passwords* is checked (the default setting) then PINs and passwords are never stored but always requested when required. The effect of these options is as follows:

1. No use of certificates:

   If the prompt for passwords box is checked then the registry PIN must be provided every time the queue manager is restarted; otherwise the PIN will be stored (insecurely) in the *.ini* file. The *com.ibm.mqe.attributes.MQeWTLSCertAuthenticator* cannot be used for any form of authentication (but other authenticators are available).

2. Certificates in use:

   a. No prompt for passwords:

      i. Queue registry enabled: the three PINs/passwords are stored in the *.ini* file. The certificate PIN will be used for all auto-registration requests for certificates (queue manager and queue). If a different auto-registration certificate PIN is required for any reason, then the file must be edited and the queue manager restarted. If an authenticatable entity has its certificate renewed (*Action→Renew Credentials*) then the PIN associated with that renewal will be prompted for, and that PIN will replace the *.ini* file certificate PIN the *next time* the queue manager is re-started.

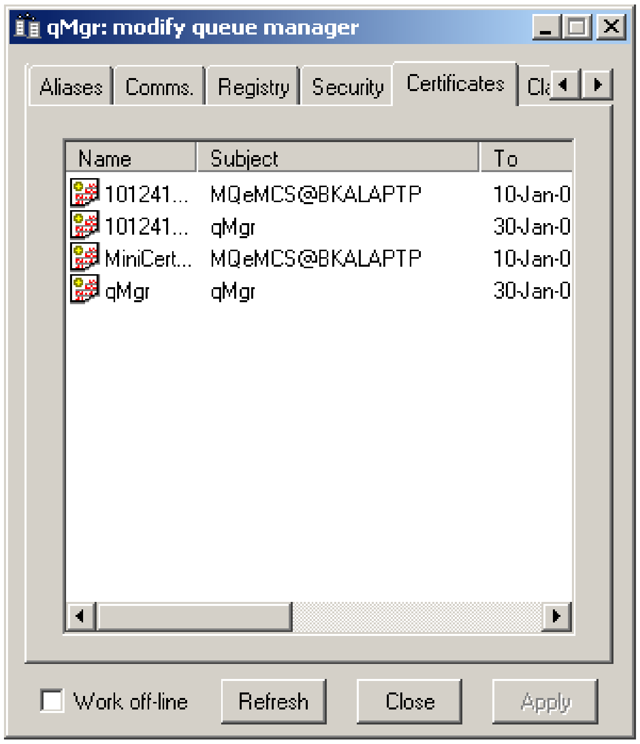
    ii. Queue registry not enabled: the three PINs/passwords are stored in the *.ini* file. The certificate PIN will be used for the auto-registration request for the queue manager certificate and will not be used again, though will remain in the *.ini* file. If the queue manager has its certificate renewed (*Action→Renew Credentials*) the PIN associated with that renewal will be prompted for, and that PIN will be placed in the *.ini* file (though will not be used again).

b. Prompt for passwords:

    i. Queue registry enabled: the three PINs/passwords will be requested each time the queue manager is started. The certificate PIN input at startup will be used for all auto-registration requests for certificates (queue manager and queue). If a different auto-registration certificate PIN is required for any reason, then the queue manager must be re-started. If an authenticatable entity has its certificate renewed (*Action→Renew Credentials*) then the PIN associated with that renewal will be prompted for, but this not affect the PIN used for auto-registration requests.

    ii. Queue registry not enabled: the first time the queue manager is started the three PINs/passwords will be requested. After a successful auto-registration of the queue manager, the certificate PIN will no longer be requested at start up. If the queue manager has its certificate renewed (*Action→Renew Credentials*) the PIN associated with that renewal will be prompted for.

The *Certificates* tab has the following appearance:



Name	Subject	To
101241...	MQeMCS@BKALAPTP	10-Jan-0
101241...	qMgr	30-Jan-0
MiniCert...	MQeMCS@BKALAPTP	10-Jan-0
qMgr	qMgr	30-Jan-0

Figure 4-24: Modify queue manager dialog – Certificates tab

- *Certificates* tab:
    - *Name* – the name of the certificate[15]. Names that start with a number are old certificates that have been renamed.
    - *Subject* – the name of the authenticatable entity that owns the certificate.
    - *To* – the date after which the certificate is invalid.
    - *From* – the date from which the certificate is valid.
    - *Issuer* – the name of the entity issuing the certificate.

Only certificates for queue manager credentials are displayed; queue-based credentials are shown in the *queue properties* panel for the relevant queue.

The *Class aliases* tab has the following appearance:

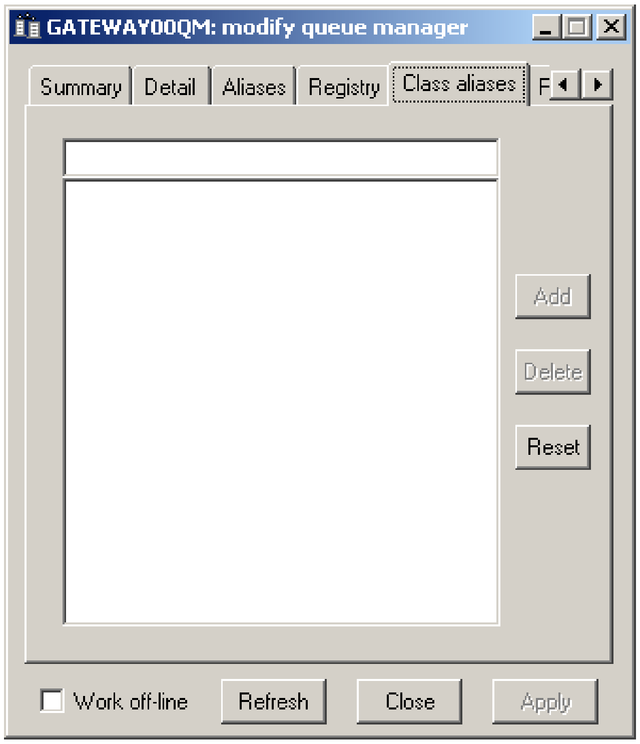


Figure 4-25: New queue manager dialog – Class aliases tab

- *Class aliases* tab:

  Alias definitions define alternative names that can be used locally to reference Java classes (usually stored in the *[Alias]* section of the initialization file). The *Add* button adds additional alias names entered in the input field; the *Delete* button deletes selected alias names; the *Reset* button resets the values to the defaults defined in the *Tools→Options→Classes* dialog.

  Alias definitions have the format:

  <Alias name>=<Class name>

[15] MQe_Explorer removes the suffix "_MiniCertificate".

Caution should be exercised before changing the mapping of a class alias name. If an MQe property takes a class as its value then, if an alias name is used, MQe will store that alias name and not the immediate resolution of that name. If the class alias is later re-defined, that re-definition will affect all properties for which that alias name has been used. This may give rise to unexpected behavior, and may even prevent a queue manager or an MQe object from access or use. For this reason, wherever possible, MQe_Explorer itself does not use alias names unless specifically requested to do so.

The *Pre-loads* tab has the following appearance:

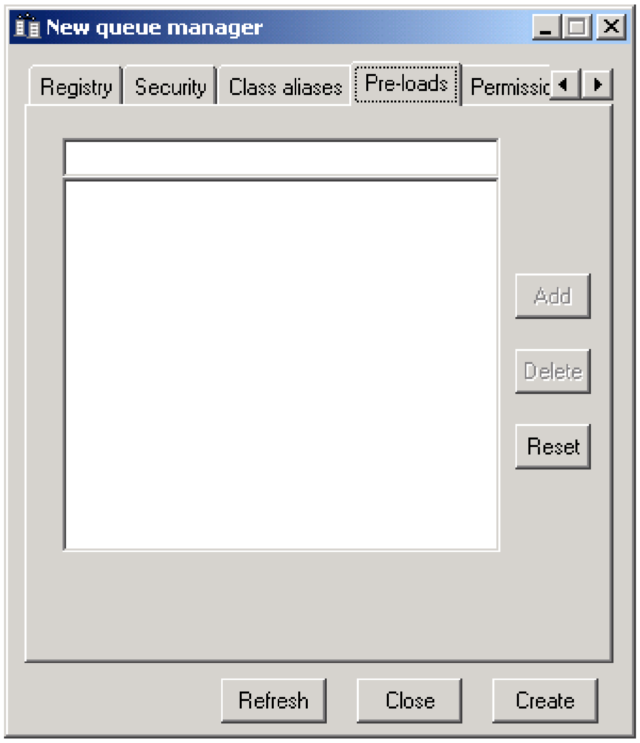


Figure 4-26: New queue manager dialog – Pre-loads tab

- *Pre-loads* tab:

  Classes (or their alias names) to be pre-loaded when the queue manager is loaded (usually stored in the *[PreLoad]* section of the initialization file). The *Add* button adds additional pre-load class names entered in the input field; the *Delete* button deletes selected names; the *Reset* button resets the values to the defaults defined by the *Tools→Options→Classes* dialog.

The *Create* button creates the new queue manager; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Open

This command is used to load a queue manager and displays the *Open file* dialog to select the associated initialization file. If the file belongs to a configured queue manager then that queue manager is started. However, if the queue manager is un-configured, MQe_Explorer will configure it, following user confirmation. The menu item *Tools→Options→Advanced-1* determines the queues that are created and the names assigned to the admin queue.

Files with extensions other than *.ini* may be opened – but are assumed to be in MQe initialization file format. *File→New* is not available after a queue manager has been loaded.

Close

The *Close* command closes the current queue manager and all associated windows. The command is not available if MQe_Explorer is running as a slave.

Properties

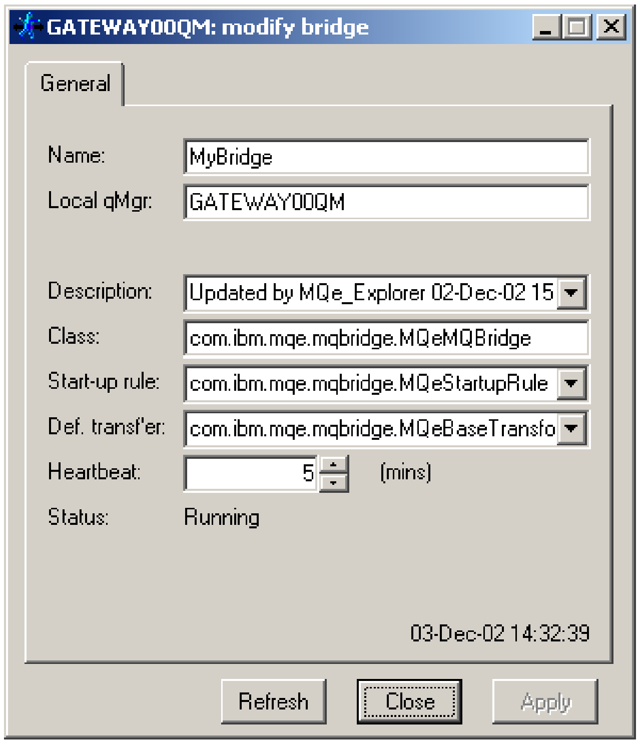
This command loads the properties pages for the selected object, i.e. one of:

- *Bridge.*
- *Client connection*.
- *Comms. listener.*
- *Connection*.
- *Listener*.
- *MQ proxy*.
- *Queue*.
- *Queue manager*.

Property pages are used to view or modify object properties. These pages are very similar to the equivalent pages used to create new objects; one difference however is that the *General* tab shows, in the bottom right hand corner, the date and time that the MQe object was queried for its details. Updated information can be obtained by clicking the *Refresh* button.

Bridge properties

A typical *Bridge property page* is:



Name: MyBridge
Local qMgr: GATEWAY00QM
Description: Updated by MQe_Explorer 02-Dec-02 15
Class: com.ibm.mqe.mqbridge.MQeMQBridge
Start-up rule: com.ibm.mqe.mqbridge.MQeStartupRule
Def. transf'er: com.ibm.mqe.mqbridge.MQeBaseTransfo
Heartbeat: 5 (mins)
Status: Running
03-Dec-02 14:32:39

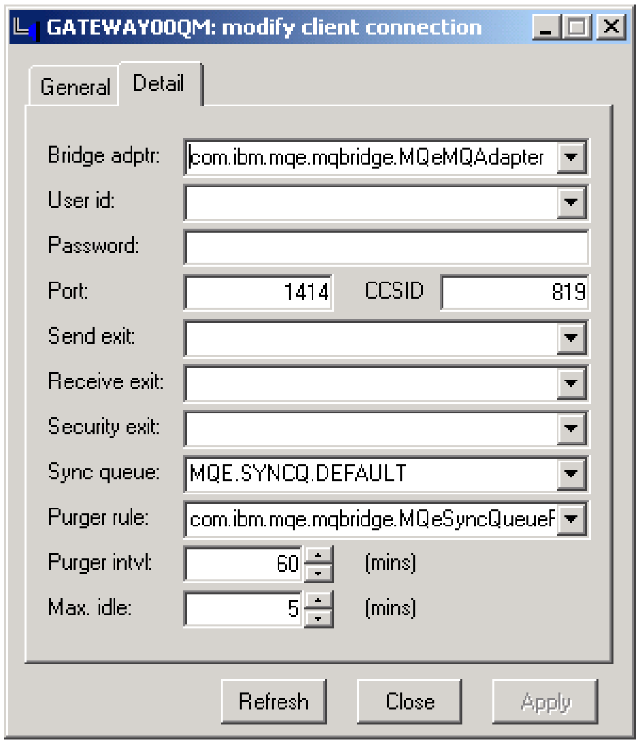
Figure 4-27: Typical bridge property page

The properties are similar to those for creating a new bridge definition in *File→New→Bridge*. Some of these cannot be changed after the object has been created, e.g. bridge name, local queue manager name and class.

The *Apply* button updates the existing properties, local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Client connection properties

A typical *Client connection property page* is:


GATEWAY00QM: modify client connection

Bridge adptr:	com.ibm.mqe.mqbridge.MQeMQAdapter
User id:	
Password:	
Port:	1414
CCSID	819
Send exit:	
Receive exit:	
Security exit:	
Sync queue:	MQE.SYNCQ.DEFAULT
Purger rule:	com.ibm.mqe.mqbridge.MQeSyncQueueF
Purger intvl:	60 (mins)
Max. idle:	5 (mins)

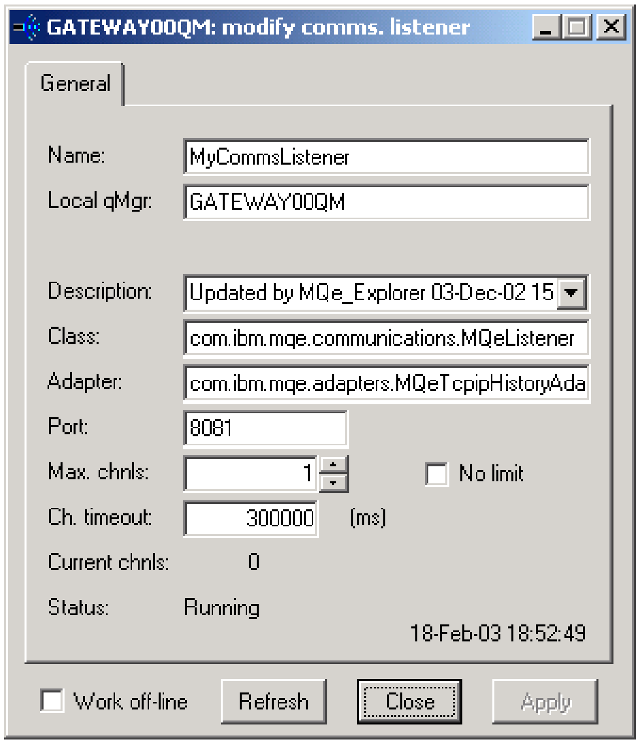
Figure 4-28: Typical client connection property page

The properties are similar to those for creating a new client connection definition in *File→New→Client Conn*. Some properties cannot be changed after the object has been created, e.g. client connection name, proxy name, bridge name, local queue manager name, class.

The *Apply* button updates the existing properties, local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Communications listener

A typical *Communications listener property page* is:


GATEWAY00QM: modify comms. listener

General

Name: MyCommsListener
Local qMgr: GATEWAY00QM
Description: Updated by MQe_Explorer 03-Dec-02 15
Class: com.ibm.mqe.communications.MQeListener
Adapter: com.ibm.mqe.adapters.MQeTcpipHistoryAda
Port: 8081
Max. chnls: 1 No limit
Ch. timeout: 300000 (ms)
Current chnls: 0
Status: Running
18-Feb-03 18:52:49

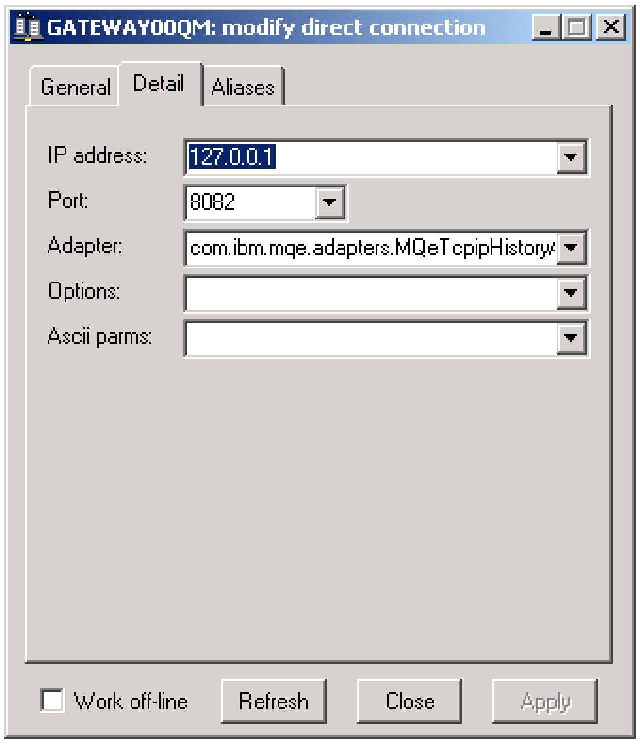
Figure 4-29: Typical communications listener property page

The properties are similar to those for creating a new comms. listener definition in *File→New→Comms. Listener*. Some properties cannot be changed after the object has been created, e.g. name, local queue manager name, adapter etc.

The *Apply* button updates the existing properties, the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window. *Work offline* allows comms. listener definitions to be modified on remote queue managers that are not currently available on the network.

Connection properties

A typical *Connection property page* is:


GATEWAY00QM: modify direct connection

General | Detail | Aliases

IP address: 127.0.0.1

Port: 8082

Adapter: com.ibm.mqe.adapters.MQeTcpipHistory

Options:

Ascii parms:

☐ Work off-line | Refresh | Close | Apply

Figure 4-30: Typical connection property page

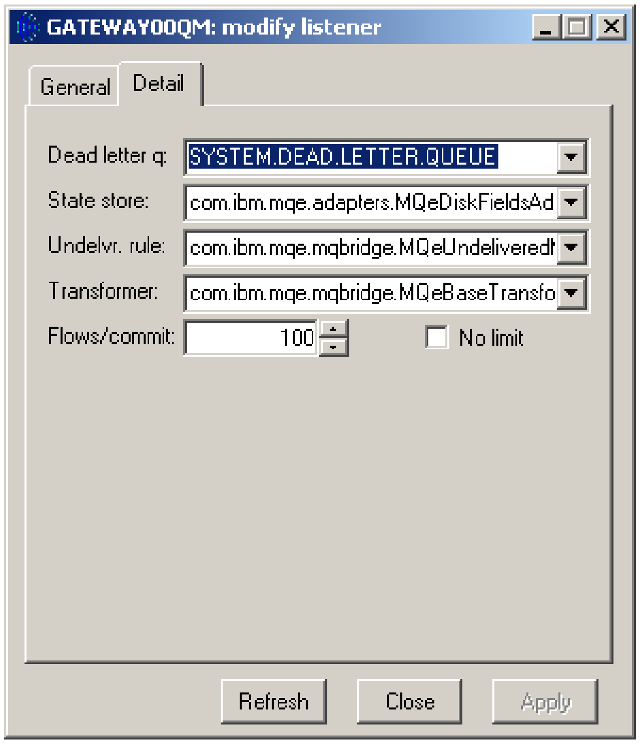
The properties are similar to those for creating a new connection definition in *File→New→Connection*. Some properties cannot be changed after the object has been created, e.g. connection name, local queue manager name.

The tabs and properties shown will vary according to the connection type and individual property values. If any property on the *Detail* tab is changed, <u>then all other such properties</u> are also changed. For example, if the adapter class, address or port is changed, then the adapter options are also changed.

The *Apply* button updates the existing properties, the local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window. *Work offline* allows connection definitions to be modified on remote queue managers that are not currently available on the network.

Listener properties

A typical *Listener property page* is:



Dead letter q: SYSTEM.DEAD.LETTER.QUEUE
State store: com.ibm.mqe.adapters.MQeDiskFieldsAd
Undelvr. rule: com.ibm.mqe.mqbridge.MQeUndeliveredI
Transformer: com.ibm.mqe.mqbridge.MQeBaseTransfo
Flows/commit: 100
No limit [ ]

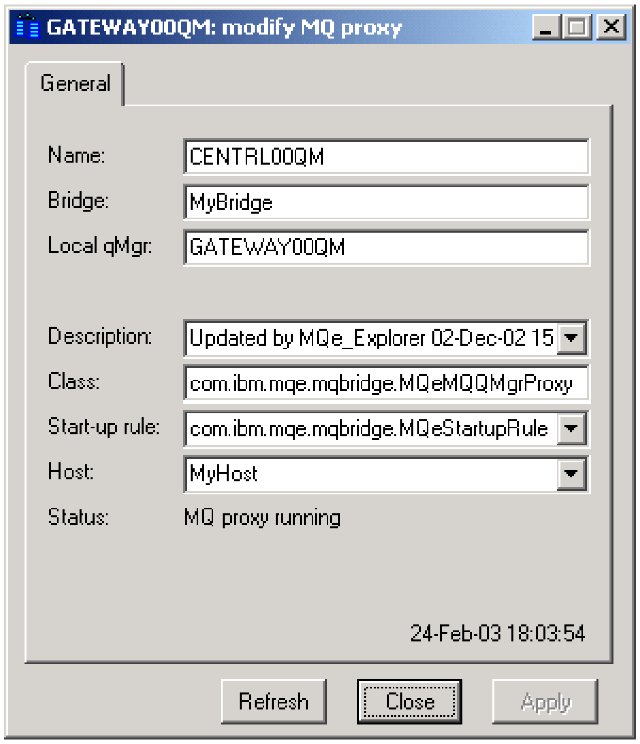
Figure 4-31: Typical listener property page

The properties are similar to those for creating a new listener definition in *File→New→Listener*. Some of these cannot be changed after the object has been created, e.g. listener name, client connection name, proxy name, bridge name and local queue manager name, class.

The *Apply* button updates the existing properties, local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

MQ proxy properties

A typical *MQ proxy property page* is:



GATEWAY00QM: modify MQ proxy

General

Name:	CENTRL00QM
Bridge:	MyBridge
Local qMgr:	GATEWAY00QM
Description:	Updated by MQe_Explorer 02-Dec-02 15
Class:	com.ibm.mqe.mqbridge.MQeMQQMgrProxy
Start-up rule:	com.ibm.mqe.mqbridge.MQeStartupRule
Host:	MyHost
Status:	MQ proxy running

24-Feb-03 18:03:54

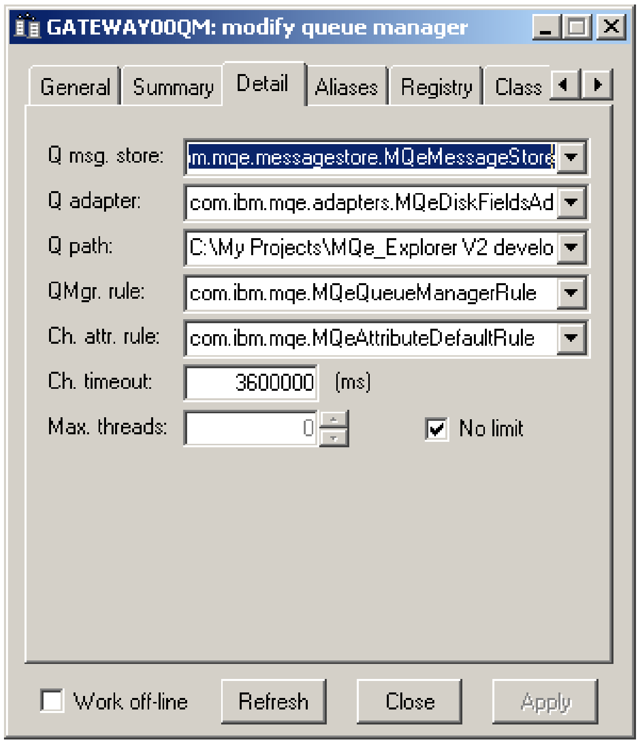
Figure 4-32: Typical MQ proxy property page

The properties are similar to those for creating a new MQ proxy definition in *File→New→MQ proxy*. Some of these cannot be changed after the object has been created, e.g. proxy name, bridge name, local queue manager name and class.

The *Apply* button updates the existing properties, local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Queue manager properties

A typical *Queue manager property page* is:


GATEWAY00QM: modify queue manager

Q msg. store: m.mqe.messagestore.MQeMessageStore

Q adapter: com.ibm.mqe.adapters.MQeDiskFieldsAd

Q path: C:\My Projects\MQe_Explorer V2 develo

QMgr. rule: com.ibm.mqe.MQeQueueManagerRule

Ch. attr. rule: com.ibm.mqe.MQeAttributeDefaultRule

Ch. timeout: 3600000 (ms)

Max. threads: 0 [✓ No limit]

[ ] Work off-line

Figure 4-33: Typical queue manager property page

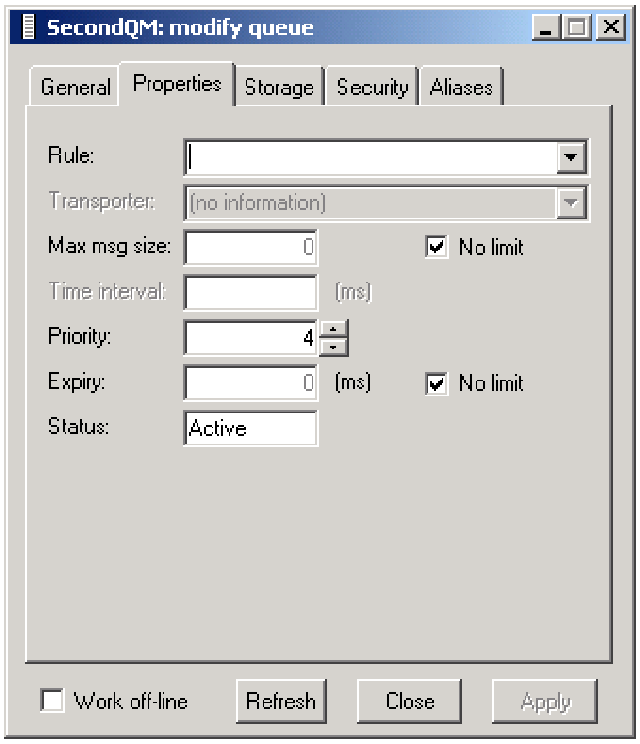
The properties are similar to those for creating a new queue manager definition in *File→New→Queue Manager*. Some of these cannot be changed after the object has been created, e.g. name, registry information, security, class.

The available tabs and properties vary, being dependent upon whether the queue manager is local or remote, whether MQe_Explorer originally created it, and upon many other factors.

The *Apply* button updates the existing properties, local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window. *Work offline* allows queue manager definitions to be modified on remote queue managers that are not currently available on the network.

Queue properties

A typical *Queue property page* is:


SecondQM: modify queue

General | Properties | Storage | Security | Aliases

Rule:			
Transporter:	(no information)		
Max msg size:	0		✓ No limit
Time interval:		(ms)	
Priority:	4		
Expiry:	0	(ms)	✓ No limit
Status:	Active		

☐ Work off-line | Refresh | Close | Apply

Figure 4-34: Typical queue property page

The properties are similar to those for creating a new queue definition in *File→New→Queue*. Some of these cannot be changed after the object has been created, e.g. queue name, local queue manager name, partner queue manager name, class, date created.

The tabs and properties shown will vary according to the queue type and individual property values.

The *Apply* button updates the existing properties, local cache and display; *Refresh* refreshes the available values in the dialog; *Close* removes the window. *Work offline* allows queue definitions to be modified on remote queue managers that are not currently available on the network.

Exit

If MQe_Explorer is running as a master, this command closes the local queue manager and then terminates MQe_Explorer, before finally closing the JVM. If classes have been loaded from MQe_Explorer and set to run on non-daemon threads, MQe_Explorer will not terminate until those classes have completed their execution.

If MQe_Explorer is running as a slave, the queue manager is not closed, nor is the JVM.

4.2 Edit

Clear

Clear deletes the all the messages on the selected queue in the tree view or list view panes. It performs an immediate refresh to get the latest queue message content and then deletes all the messages found. This does not guarantee that the queue will then be empty, since messages may subsequently arrive on the queue. The display is refreshed after the operation.

Cut

Cut copies one (or more) messages selected in the list view pane to the clipboard; then deletes the selected messages (if still present on the source queue). Note that the copies are made from the cached messages held by MQe_Explorer – and at the time that they are cut it may be true (but immaterial) that the source queue no longer holds these messages.

The display of messages is updated to reflect the cut operation – but <u>not refreshed</u> since this would change the message numbering in force.

Copy

Copy copies to the clipboard one or more messages selected in the list view pane. Note that the copies are made from copies already held by MQe_Explorer – and at the time that they are copied it may be true (but immaterial) that the source queue no longer holds these messages.

Delete

Delete deletes the selected object(s) in the tree view or list view panes, provided that they are of one of the following object types: *connection*, *comms. listener*, *bridge*, *client connection*, *listener*, *MQ proxy*, *queue* or *message*. Objects, apart from messages, can only be deleted one at a time.

Store, forward or async. proxy queues cannot be deleted if they contain messages. Sometimes these messages will be the result of MQe_Explorer using these queues to manage remote queue managers; in these cases wait until the messages expire and then delete the queues. If the queues are on the local queue manager, reloading it and MQe_Explorer will not normally remove the messages, since MQe_Explorer attempts to contact all remote queue managers during initialization. Admin message expiry is set through the *Tools→Options→Advanced-2* menu.

With the exception of messages, the display is refreshed after the delete operation. For messages, the display is updated but <u>not</u> refreshed, since refreshing would change the message numbering.

Paste

Paste copies the message(s) on the clipboard, <u>resets their UIDs</u> (to avoid duplicates), and puts them on the target queue. The clipboard contents are not cleared after a paste operation – thus messages may be re-pasted. The target queue cache and display are then refreshed (note that this unavoidably changes the message numbering in force).

Select All

Select All selects all the messages in the list view pane, provided that at least one message is already selected.

4.3 View

Console

Console provides a means to view and manage applications running against the local queue manager. Typically such applications (or classes) will have been loaded through the use of the *Tools→Load* menu item. Further details of the console capabilities are given in the chapter *Console* on page 103.

Details

This view option affects the list view pane – all objects are displayed as a row of column values. The first column holds the name together with an associated icon. The remaining columns display object properties. The columns available and their default widths are set through the *Tools→Options* menu; to change these values use the tab appropriate to the object type.

Columns with no data can be set to zero width via the *View→Hide* Empty menu; additional rows per object alias name can be enabled through the *View→Use Multiple* menu.

Hide Empty

This view option affects the list view pane and causes columns with no data to be displayed as columns of zero width. The effect is to reduce the need for scrolling in the list view pane.

List

This view option affects the list view pane – all objects are displayed as the object name together with an associated icon, with a single object per row. No object properties are available.

Offline Admin

This menu item displays the *Offline administration window*, used for viewing and managing offline administration requests and responses. See *Offline administration* on page 98.

Refresh

Refresh refreshes the displayed information for the selected object and its children, allowing the current state of an object to be determined, given that it may have been externally updated, e.g. refreshing a queue object shows recent messages on that queue, as well as the current state of the queue object properties.

Refresh of the bridges, comms. listeners, connections or local queues folders, refreshes the parent queue manager object properties along with the relevant children; thus a refresh operation on the bridges folder also refreshes all information on the bridges object and each bridge, MQ proxy, client connection and listener for that queue manager.

Refreshing of a connection refreshes details of that connection; it also refreshes the parent queue manager object properties and all remote queue definitions.

In most cases when properties are changed through MQe_Explorer, an automatic refresh and display update takes place. Notable exceptions are the message *Cut* and *Paste* operations, when MQe_Explorer does not automatically refresh in order to preserve the current message numbering scheme.

One alternative to clicking *Refresh* from the *View* menu is to display the property pages for an object and then click the *Refresh* button on those pages. This approach is a more efficient way of viewing the current object properties since child object properties are not also refreshed.

Small Icons

This view option affects the list view pane – all objects are displayed as the object name together with an associated icon, with multiple objects per row. No object properties are available.

Trace

This menu item displays the *Trace window*, used for debugging MQe applications. See *Trace* on page 106.

Translate Admin

This view option affects the list view pane only and translates certain message field names:

- Admin parameter names in messages are translated into meaningful names, e.g. *admact* becomes 'Action'.
- Reserved field names are translated into meaningful names, e.g. **Msg_Style* becomes 'Style'.

The translated names are shown in single quotes.

Use Multiple

This view option affects the display of objects with alias names.

Normally in the detailed list view mode (*View→Details*), a single row (the base row) is used to display an object. With the option, enabled an additional (alias) row is shown for each alias name. For both base and alias rows, the *Base* column shows the name of the base object and the *Aliases* column the list of alias names. For base rows the *Name* column (always the first) shows the base name along with an object icon; for alias rows it shows the alias name along with a shortcut object icon. For example, for a queue manager with an alias name, the original row will have the queue manager icon and an alias row will be have the shortcut queue manager icon .

In the small icons view mode (*View→Details)*, or the list view mode (*View→Details)*, additional entries are present for each alias name.

Normal object operations can be applied to alias rows or entries, for example, a right hand context menu can be displayed, a double click display the child objects, etc. In the details list view both base and alias rows participate in sort and re-order operations.

4.4 Action

Get New Credentials

Queue managers created with a private registry and using certificate-based security, have a WTLS certificate in their registry that authenticates the identity of the queue manager, and another that authenticates the certificate supplier. These certificates were obtained through auto-registration with the MQe mini-certificate server and occurred at the first queue manager re-start after certificate-based security was enabled. If such queue managers also use *com.ibm.mqe.attributes.MQeWTLSCertAuthenticator* class authenticators on queues, and have the target registry property set to 'Queue', then they will also have additional certificates present that authenticate queue identities.

Mini-certificates issued by the MQe mini-certificate server expire at a pre-determined time after issue; normally they are renewed through the *Renew Credentials* action – this method retains the existing public and private keys. In cases where the server that issued the original certificates is not available, the credentials cannot be renewed, but must be replaced. The *Get New Credentials* action causes a request to be issued to the mini-certificate server to replace the existing mini-certificates, using new public and private keys. The old certificates are renamed[16] and remain in the private registry. The mini-certificate server must have been prior authorized to issued the new credentials; associated with the issue is a certificate-request PIN. This PIN must be supplied during the renewal process. For more details see the *Security* tab of the *Queue Manager* on page 42 and also the *WebSphere MQ Everyplace Configuration Guide*.

The *Get New Credentials* action applies to the selected object in the tree or list view pane. Only the queue manager hosting MQe_Explorer and its child queue objects may have their credentials updated in this way.

Ping

Ping is used to determine if the selected queue manager is alive. A message is sent synchronously[17] to the admin queue of the queue manager and a reply is requested. If the message cannot be sent, an error message will be issued. When the response is received a message is then displayed, with the identity of the remote queue manager and with the overall elapsed time between the ping being requested and the response being received.

Note that ping uses the network connectivity defined for the remote queue manager on the hosting queue manager. If a ping request cannot be sent, or if a response is not subsequently received, this may indicate either a network problem between the hosting queue manager and the target queue manager, or alternatively that the target queue manager may be unresponsive. If queue-level confirmation is required then a test message can be sent using *File→New→Message*.

[16] The rename adds a time prefix to the existing name, as determined by the algorithm: *Long.toString(new Date().getTime()) + "_"*
[17] This assumes that the remote admin queue is named according to the convention and that the local queue manager does not have a conflicting asynchronous remote queue defined.

Renew Credentials

Queue managers created with a private registry and using certificate-based security, have a WTLS certificate in their registry that authenticates the identity of the queue manager, and another that authenticates the certificate supplier. The certificates were obtained through auto-registration with the MQe mini-certificate server and occurred at the first queue manager re-start after certificate-based security was enabled. If such queue managers also use *com.ibm.mqe.attributes.MQeWTLSCertAuthenticator* class authenticators on queues, and have the target registry property set to 'Queue', then they will have additional certificates present that authenticate queue identities.

Mini-certificates issued by the MQe mini-certificate server expire after a pre-determined time after issue. The *Renew Credentials* menu item causes a request to be issued to the mini-certificate server to renew the mini-certificates, keeping the same public and private keys. The old certificates are renamed[18] and remain in the private registry. The mini-certificate server must have been prior authorized to issued the renewed credentials; associated with the renewal is a certificate-request PIN. This PIN must be supplied during the renewal process. For more details see the *Security* tab of the *Queue Manager* on page 42, and also the *WebSphere MQ Everyplace Configuration Guide*.

The *Renew Credentials* action applies to the selected object in the tree or list view pane. Only the queue manager hosting MQe_Explorer and its child queue objects may have their credentials renewed in this way.

Start

The *Start* menu item issues a start command to the selected object that affects both that object and any children. The following objects may be started: comms. listeners, the bridges object (represented by the *Bridges folder*), bridges, MQ proxies, client connections and listener objects.

Stop

The *Stop* menu item issues a stop command to the selected object that affects both that object and any children. The following objects may be stopped: comms. listeners, the bridges object (represented by the *Bridges folder*), bridges, MQ proxies, client connections and listener objects.

Trigger Local

Trigger Local causes the immediate sending of any messages awaiting transmission on the local queue manager. Normally the current queue manager rule initiates message triggering; with the default rule in force, this happens on queue manager startup or when a new message arrives on the queue. The *Trigger* command provides a convenient way of re-trying message transmission without sending additional messages or re-cycling the queue manager.

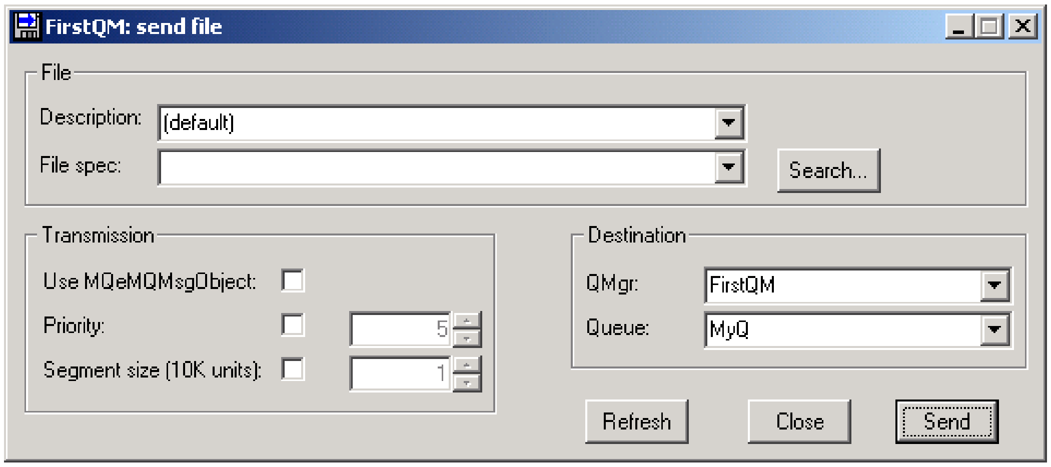
Note that *Trigger Local* only affects the queue manager used to host MQe_Explorer.

[18] The rename adds a time prefix to the existing name, as determined by the algorithm: *Long.toString(new Date().getTime()) + "_"*

Send File

Action→Send File enables the *Send file* dialog used to send copies of file to a target queue. The currently selected queue in the list or tree view panes is assumed to be the queue to receive the file, but this can be overridden. By default, it will use the message class *com.ibm.mqe.mqe_explorer.MQeFileTransferMsg*[19]; optionally it will use the *com.ibm.mqe.mqemqmessage.MQeMQMsgObject* class instead.

The *MQeFileTransferMsg* option supports the segmentation of files across multiple messages, integrity checking of the data upon receipt to detect corruption, and the transport of additional associated information, such as a description and the source location. These facilities are not available with *MQeMQMsgObject*; however such messages will flow across the MQe gateway to WebSphere MQ queue managers without the need for custom transformation rules.


FirstQM: send file

File
Description: (default)
File spec:

Transmission
Use MQeMQMsgObject: ☐
Priority: ☐ 5
Segment size (10K units): ☐ 1

Destination
QMgr: FirstQM
Queue: MyQ

Figure 4-35: Send file dialog

The following input fields are present:

- *File* group:
  - *Description* – a description of the file.
    - *(default)* generates a description that automatically includes the name of the sending queue manager, the date and time.
  - *File specification* – an ASCII string specifying the location of the file.
    - The *Search* button brings up a file dialog and can be used to load this field.
- *Destination* group:
  - *Queue manager* – the name of the target queue manager. This may be entered or selected from the drop-down list. Any name must identify a queue manager to which addressability exists from the queue manager hosting MQe_Explorer, i.e. it must be one of the following: the name of the hosting queue manager (or an alias); a connection name (or an alias); a destination name owned by a store or forward queue (or an alias).

[19] Further details on this class and its use for file transfer are given in *Appendix C: MQeFileTransferMsg class* on page 176.

    - *Queue* – the name of the target queue. This may be entered or selected from the drop-down list. Any name entered must either be known to the queue manager hosting MQe_Explorer (in conjunction with the target queue manager name) as a queue name (or alias), or otherwise will be assumed to identify a synchronous proxy queue.

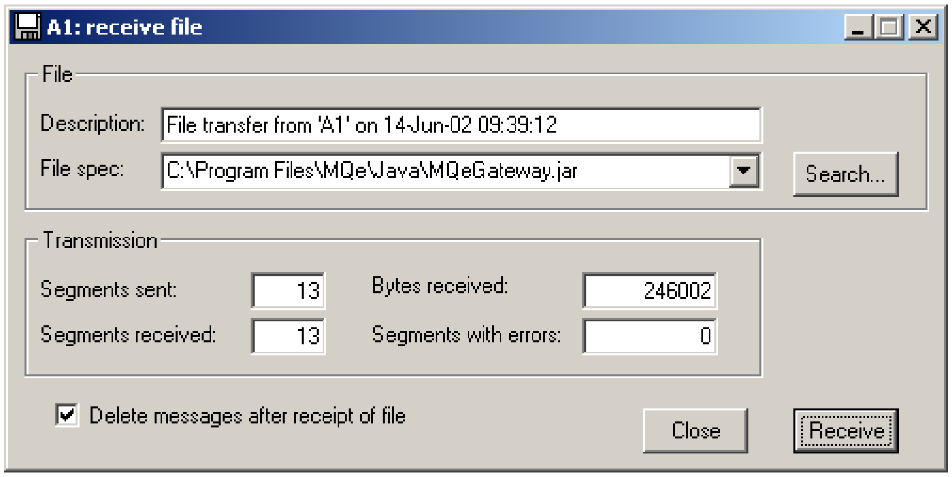
- *Transmission* group:
    - *Use MQeMQMsgObject* – the message object class used is *com.ibm.mqe.mqemqmessage.MQeMQMsgObject* (this class may be useful when sending to MQ queue managers as it is supported by the default bridge transformer).
    - *Priority* – if the box is unchecked the priority will default to 5; if checked the selected priority (0 – 9) is added to the message.
    - *Segment size* – if the box is unchecked the file will be transmitted in a single message; if checked the segment size (number of bytes per message) can be set in multiples of 10240 bytes.

The *Send* button sends a copy of the file; *Refresh* refreshes the available values in the dialog; *Close* removes the window.

Receive File

Action→Receive File enables the *Receive file* dialog used to receive files from messages (and typically sent through the *Action→Send File* dialog or an equivalent). A message containing the file (or a segment thereof) must have been previously selected in either the tree or list view panes.

The function available depends upon whether the file was sent via message(s) of the *com.ibm.mqe.mqe_explorer.MQeFileTransferMsg* or the *com.ibm.mqe.mqemqmessage.MQeMQMsgObject* class (for more details see *Send File* on page 67).


A1: receive file

File

Description: File transfer from 'A1' on 14-Jun-02 09:39:12

File spec: C:\Program Files\MQe\Java\MQeGateway.jar

Transmission

Segments sent:	13	Bytes received:	246002
Segments received:	13	Segments with errors:	0

☑ Delete messages after receipt of file

Figure 4-36: Receive file dialog

The following fields are present:

- *File* group:
    - *Description* – a description of the file.
    - *File specification* – an ASCII string initially loaded (where possible) with the source location of the file; this value must be edited into the target file location.
        - The *Search* button brings up a file dialog and can be used to load this field.
- *Transmission* group:
    - *Segments sent* – the number of segments (messages) sent.
    - *Segments received* – the number of segments (messages) received.
    - *Bytes received* – the total number of bytes received.
    - *Segments with error* – the number of segments (messages) detected as being corrupt.
- *Delete messages after receipt of file* – if checked, the messages are deleted after the file has been successfully retrieved.

The *Retrieve* button saves the file (and optionally deletes the messages); *Close* removes the window.

4.5 Tools

Demo Mode

Demo Mode is provided as a way of automatically generating messages on a queue – such that MQe_Explorer can then be used to view them. It avoids the need to have application programs running when learning about MQe messaging or when demonstrating MQe (or MQe_Explorer).

Demo Mode, where possible, makes a copy of <u>all admin messages sent or received</u> by MQe_Explorer. The copies are placed on the *AdminReplyQ* of the hosting queue manager.

Copies are constructed such that:

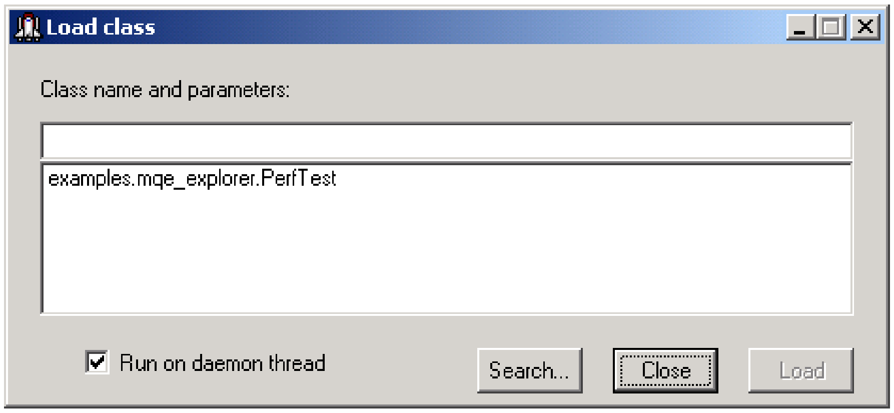
- The message class of the copy matches that of the original.
- If the message has been received, the original *UID* is copied into a new field called **Original_UID* and the UID is then reset. The UID in sent messages is not known at the time of copying; the UID in the copy is unique to that copy.
- The value of the original **Msg_CorrelId* field is set to an array of zero bytes.
- The original **Msg_CorrelId* field value is copied into a new field called **Original_CID*.
- A new UNICODE **Copy_Info* field is added with the date/time when the copy was taken.
- All other fields are copied unchanged.

The changes to the *UID* are made because MQe requires that all *UIDs* be unique. The changes to **Msg_CorrelID* are to avoid the copy of the message being interpreted by MQe_Explorer as a valid admin message.

Since MQe_Explorer uses message expiry in its admin messages there is a danger that the admin messages may have expired on the queue before they are inspected; in which case they will disappear from the *AdminReplyQ*. To avoid this possibility, turn off demo mode and then refresh the view of the *AdminReplyQ* as soon as possible after the admin operation is complete. MQe_Explorer caches the results of MQe object refreshes and the contents of these caches do not expire. Alternatively, the lifetime of admin messages can be controlled through the *Tools→Advanced-2* menu.

Load

The *Load* option enables Java classes to be loaded into the current JVM and started, thus providing for one or more MQe applications to be run concurrently with MQe_Explorer against the local queue manager. The window below is used to load the classes:



examples.mqe_explorer.PerfTest

Figure 4-37: Load Java class dialog

The list box contains the default list of classes (changeable via *Tools→Options→Classes*). The name of a class to be loaded can be selected from the list box or entered on the input line (with or without the *.class* suffix); alternatively the file system can be searched using the *Search* button. Search will load a modified form of the full path of the selected file into the input area (the modifications change the "\" characters in the path to "." and remove the drive prefix). This modified string must then be edited (if necessary) into a proper Java class name that is consistent with the *classpath* system variable[20] – such that the Java class loader can locate it. Loaded classes (or load attempts) are added to the list box beneath the input area; a class can be re-loaded by selecting its name in the list box and clicking *Load*.

MQe_Explorer will start a new execution thread for the class and then invoke its static main method. It is not responsible for subsequently closing that application, which should occur before MQe_Explorer itself is closed. If you do close MQe_Explorer first, the local queue manager will be shut down whilst the loaded application is still running – which is likely to result in application errors. MQe_Explorer will not finally exit until all of the application's execution threads have been destroyed. It is not recommended that you exit from the Java Virtual Machine from within an application – this prevents a graceful shutdown of MQe.

By default, the application will be run on a daemon thread; if the *Run on daemon thread* check box is unchecked, then a user thread will be used. The status of running classes can be viewed in the MQe_Explorer console, accessed via the *View→Console* menu item. The *View→Trace* command may be used to trace application execution errors and exceptions.

[20] The current value of *classpath* is displayed in *System Information*, accessible from the *Help→About MQe_Explorer* menu item.

The sample performance tool supplied with MQe_Explorer is an MQe application that can be loaded in this way; the class name to be used is: *examples.mqe_explorer.PerfTest*. This tool enables simple performance measurements; more details are provided in *Appendix A: Performance tool* on page 162.

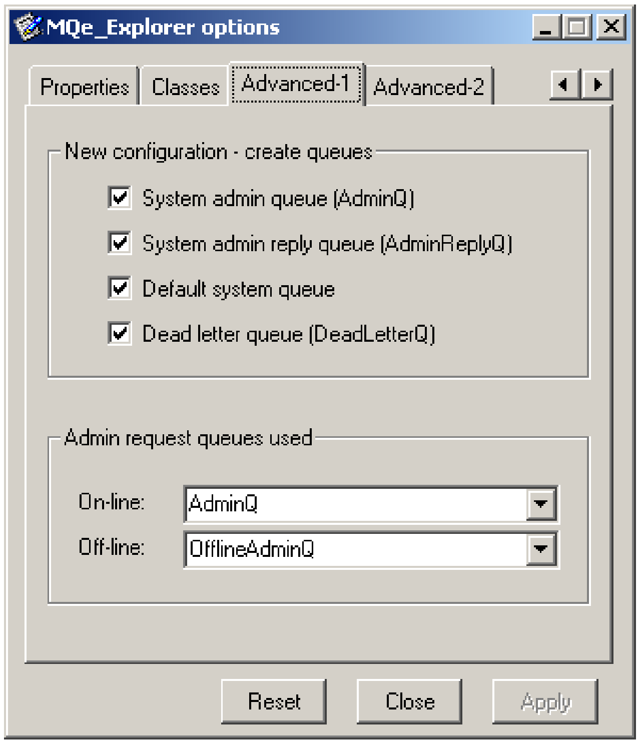
Further information on writing applications that can be loaded by MQe_Explorer is given in *Programming MQe_Explorer* applications on page 158.

Options

The *Options* menu changes MQe_Explorer settings, with the results being stored in the associated options file. The *Tools→Options* displays a tabbed control, each being for a related group of settings. Changes only take effect after the *Apply* button is clicked; in a very few cases the changes are not actioned immediately – where this applies is detailed in the relevant option description. The *Close* button removes the window and any pending changes not yet applied are lost. The *Reset* button resets all current and saved options to their default values.

Advanced-1

This tab controls the queues created and used by MQe_Explorer:



New configuration - create queues

- [✓] System admin queue (AdminQ)
- [✓] System admin reply queue (AdminReplyQ)
- [✓] Default system queue
- [✓] Dead letter queue (DeadLetterQ)

Admin request queues used

On-line: AdminQ

Off-line: OfflineAdminQ

Figure 4-38: The options dialog – Advanced-1 tab

The *New configuration – create queues* group, controls the queues that MQe_Explorer will create when a new queue manager is to be configured; the *Admin request queues used* group, controls the names of the queues to which MQe_Explorer will send admin messages (for more information see *Admin and reply queues* on page 116).

New configuration – create queues:

AdminQ: The queue used to receive admin requests (creation mandatory).

AdminReplyQ: The queue used to receive replies to admin requests (creation mandatory).

Default system queue: The queue named SYSTEM.DEFAULT.LOCAL.QUEUE (created if the box is checked).

DeadLetterQ: The queue used to hold undeliverable messages (created if the box is checked).

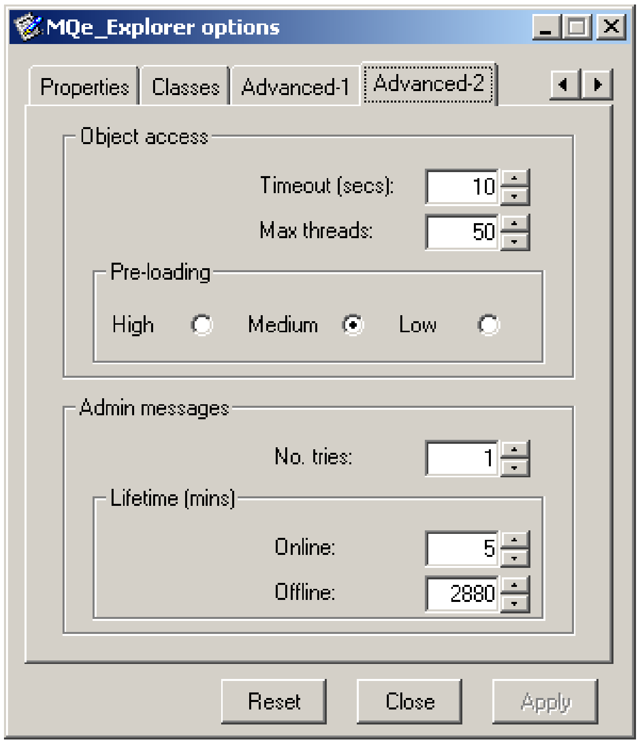
Admin request queues used *(note: changes only take effect when the host queue manager is re-loaded):*

Online: The name of the admin queue to which online requests are sent.

Offline: The name of the admin queue to which offline requests are sent.

Advanced-2

This tab controls the way in which MQe_Explorer asynchronously accesses remote objects:



Object access

Timeout (secs): 10

Max threads: 50

Pre-loading

High ○ Medium [✓ ●] Low ○

Admin messages

No. tries: 1

Lifetime (mins)

Online: 5

Offline: 2880

Figure 4-39: The options dialog – Advanced-2 tab

The *object access* group influences MQe_Explorer behavior when interfacing to local and remote MQe objects; the *admin messages* group sets the properties of the admin message used:

Object access:

Timeout: the time in seconds that MQe_Explorer will wait before assuming that a response has not been received.

Max threads: the maximum number of threads that will be used by MQe_Explorer to process asynchronous work items. The current number of threads can be seen by displaying the Console window (*View→Console*); all such threads belonging to the *Loaders* group.

Preloading: the extent to which MQe_Explorer will acquire details of related objects in advance of a specific user request for that item of information. High will normally provide the best interactive performance, but at the expense of network traffic and storage; low ensures that MQe_Explorer avoids acquiring un-requested data wherever possible.

Admin messages:

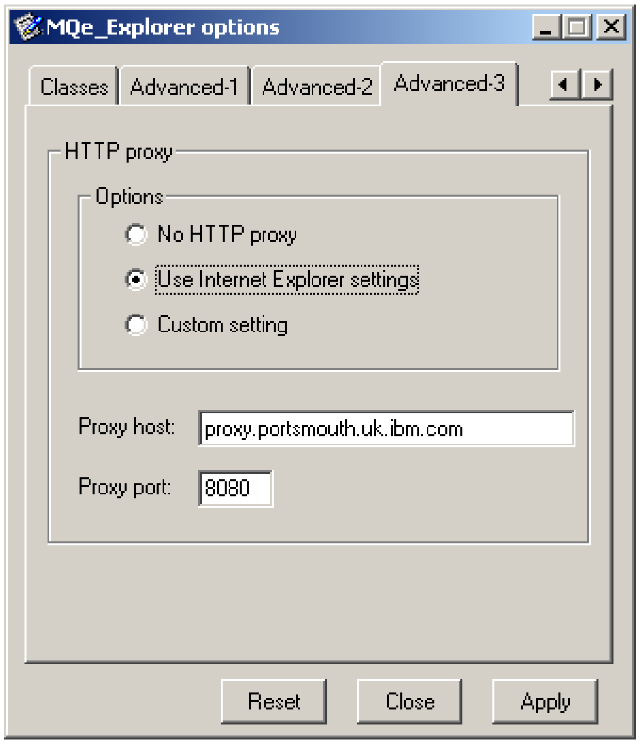
No. of tries: the number of tries permitted to an admin operation by MQe.

Online lifetime: the time in minutes that an admin message sent to an online admin queue will live, before being expired by MQe.

Offline lifetime: the time in minutes that an admin message sent to an offline admin queue will live, before being expired by MQe.

Advanced-3

This tab allows control of the HTTP proxy that may be used in MQe communications over the HTTP protocol:



HTTP proxy

Options

No HTTP proxy [✓ Use Internet Explorer settings] Custom setting

Proxy host: proxy.portsmouth.uk.ibm.com

Proxy port: 8080

Figure 4-40: The options dialog – Advanced-3 tab

Options:

No HTTP proxy: communications will not be routed via an HTTP proxy.

Use Internet Explorer settings: the options set in the Windows Internet Explorer *Tools/Internet Options/Settings* or *Tools/Internet Options/LAN Settings/Proxy Server* (as appropriate) will be used. The values are captured at MQe_Explorer start-up time; subsequent changes are ignored.

Custom setting: the values in the *Proxy host* and *Proxy port* fields are used.

Fields:

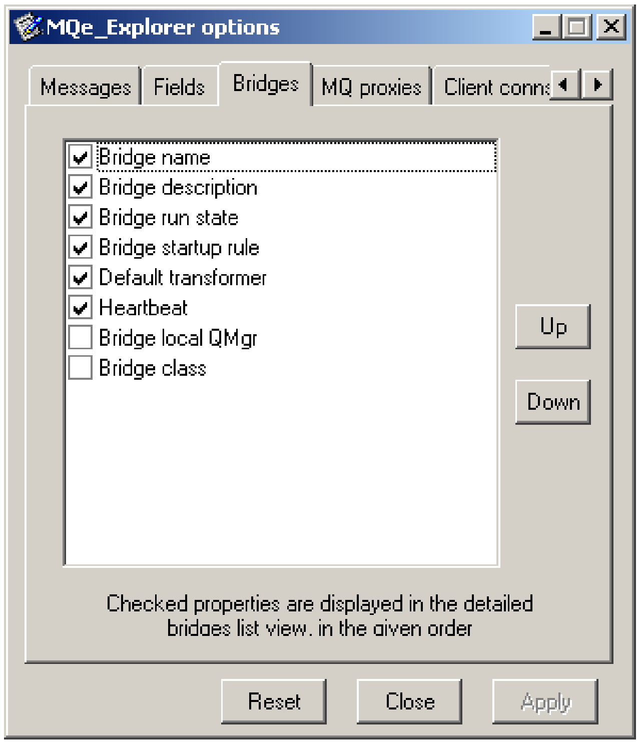
Proxy host: the IP name or numeric address of the proxy.

Proxy port: the IP proxy port number.

By default, no HTTP proxy is used.

Bridges

This tab controls the detailed list view display of bridge properties (*View→Details*), i.e. when the Bridges object is selected in the tree view pane. The panel is:



- [x] Bridge name
- [x] Bridge description
- [x] Bridge run state
- [x] Bridge startup rule
- [x] Default transformer
- [x] Heartbeat
- [ ] Bridge local QMgr
- [ ] Bridge class

Checked properties are displayed in the detailed bridges list view, in the given order

Figure 4-41: Options dialog – Bridges tab

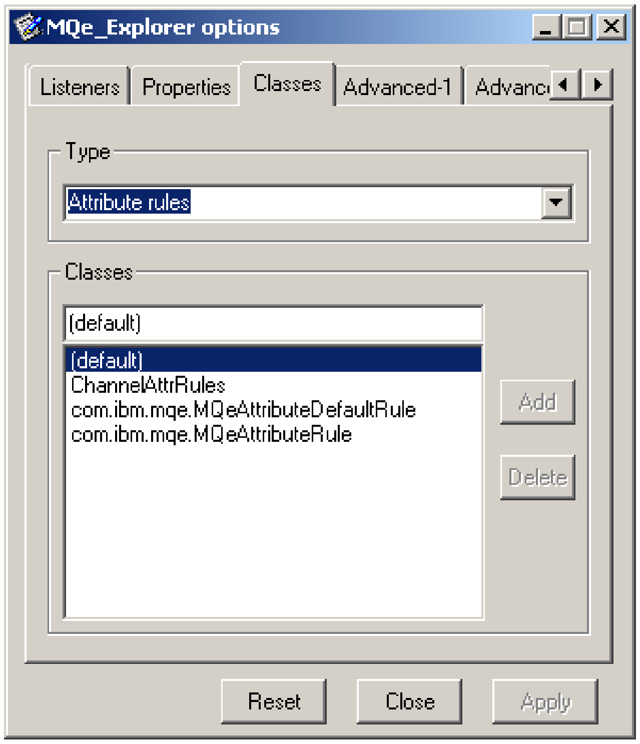
The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Classes

This tab allows the modification of the contents of certain list boxes presented by MQe_Explorer in property pages and other panels. The list box contents are class names (or aliases) and the feature can be used to add or remove entries. Coded values, i.e. those shown in brackets such as "*(default)*", can be neither added nor deleted.

A sample panel is:



Type: Attribute rules

Classes:

(default)

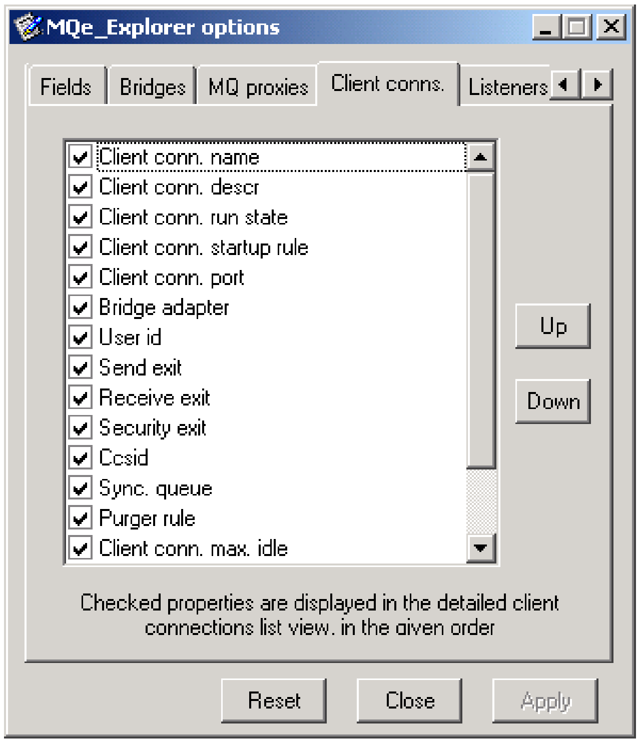
```
(default)
ChannelAttrRules
com.ibm.mqe.MQeAttributeDefaultRule
com.ibm.mqe.MQeAttributeRule
```

Figure 4-42: Options dialog – Classes tab

The *Add* button adds additional values entered in the input field; the *Delete* button deletes selected values.

Client connections

This tab controls the detailed list view display of field properties (*View→Details*), i.e. when an MQ proxy queue manager node is selected in the tree view pane. The panel is:



- ☑ Client conn. name
- ☑ Client conn. descr
- ☑ Client conn. run state
- ☑ Client conn. startup rule
- ☑ Client conn. port
- ☑ Bridge adapter
- ☑ User id
- ☑ Send exit
- ☑ Receive exit
- ☑ Security exit
- ☑ Ccsid
- ☑ Sync. queue
- ☑ Purger rule
- ☑ Client conn. max. idle

Checked properties are displayed in the detailed client connections list view. in the given order

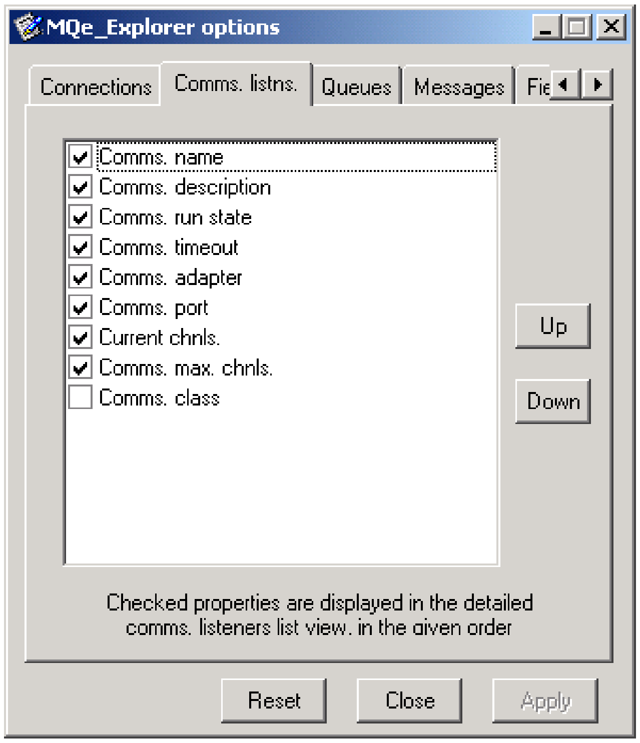
Figure 4-43: Options dialog – MQ client conns tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Communications listeners

This tab controls the detailed list view display of connection properties (*View→Details*), i.e. when the *Comms. listener* folder node is selected in the tree view pane. The panel is:



- [✓] Comms. name
- [✓] Comms. description
- [✓] Comms. run state
- [✓] Comms. timeout
- [✓] Comms. adapter
- [✓] Comms. port
- [✓] Current chnls.
- [✓] Comms. max. chnls.
- [ ] Comms. class

Checked properties are displayed in the detailed comms. listeners list view. in the given order

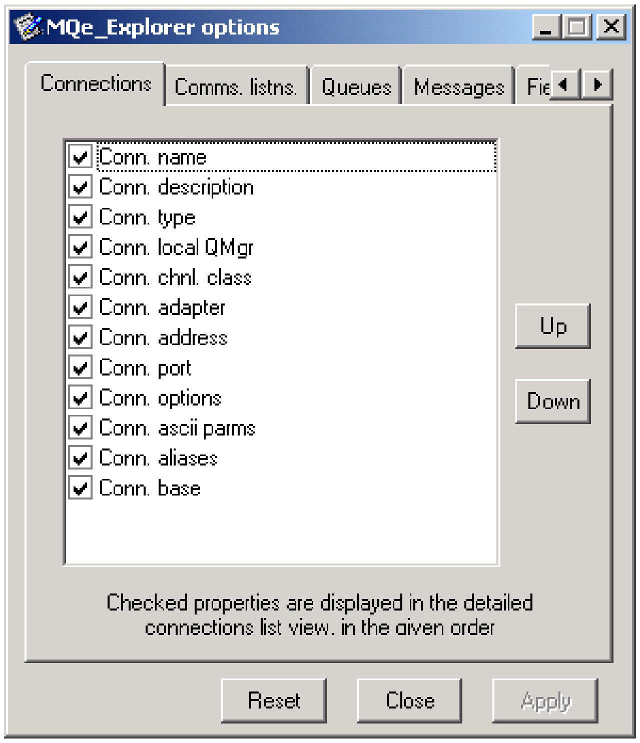
Figure 4-44: Options dialog – Comms. listeners tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Connections

This tab controls the detailed list view display of connection properties (*View→Details*), i.e. when the *Connections* folder node is selected in the tree view pane. The panel is:



- [✓] Conn. name
- [✓] Conn. description
- [✓] Conn. type
- [✓] Conn. local QMgr
- [✓] Conn. chnl. class
- [✓] Conn. adapter
- [✓] Conn. address
- [✓] Conn. port
- [✓] Conn. options
- [✓] Conn. ascii parms
- [✓] Conn. aliases
- [✓] Conn. base

Checked properties are displayed in the detailed connections list view. in the given order

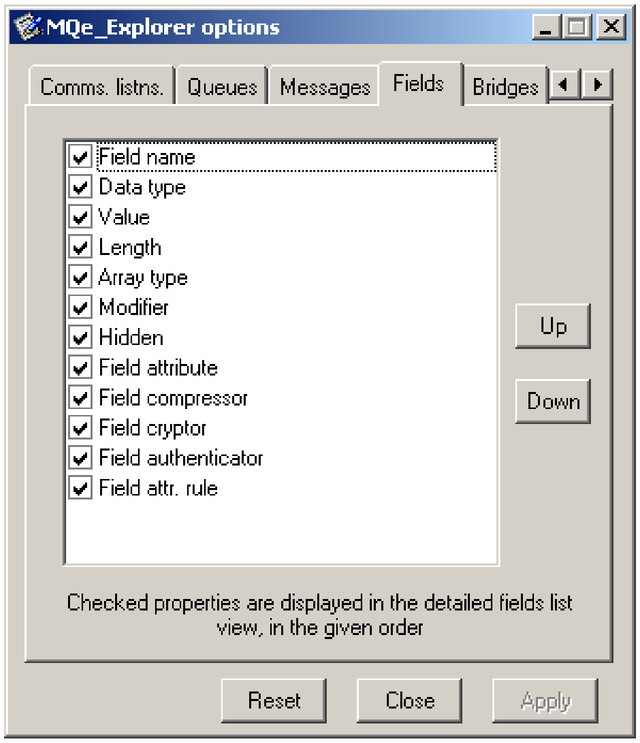
Figure 4-45: Options dialog – Connections tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Fields

This tab controls the detailed list view display of field properties (*View→Details*), i.e. when a *message* or a *fields* node is selected in the tree view pane. The panel is:



- ✓ Field name
- ✓ Data type
- ✓ Value
- ✓ Length
- ✓ Array type
- ✓ Modifier
- ✓ Hidden
- ✓ Field attribute
- ✓ Field compressor
- ✓ Field cryptor
- ✓ Field authenticator
- ✓ Field attr. rule

Checked properties are displayed in the detailed fields list view, in the given order

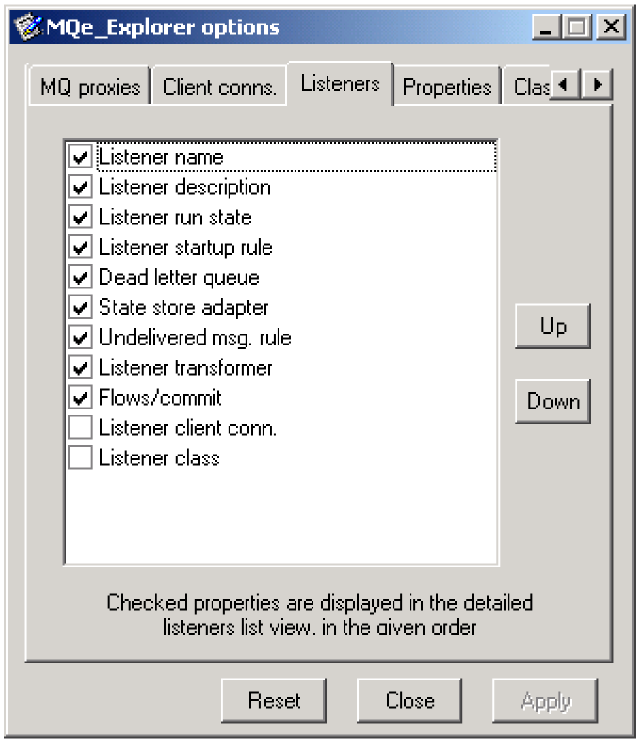
Figure 4-46: Options dialog – Fields tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Listeners

This tab controls the detailed list view display of field properties (*View→Details*), i.e. when an *client connection* node is selected in the tree view pane. The panel is:



☑ Listener name
☑ Listener description
☑ Listener run state
☑ Listener startup rule
☑ Dead letter queue
☑ State store adapter
☑ Undelivered msg. rule
☑ Listener transformer
☑ Flows/commit
☐ Listener client conn.
☐ Listener class

Checked properties are displayed in the detailed listeners list view, in the given order

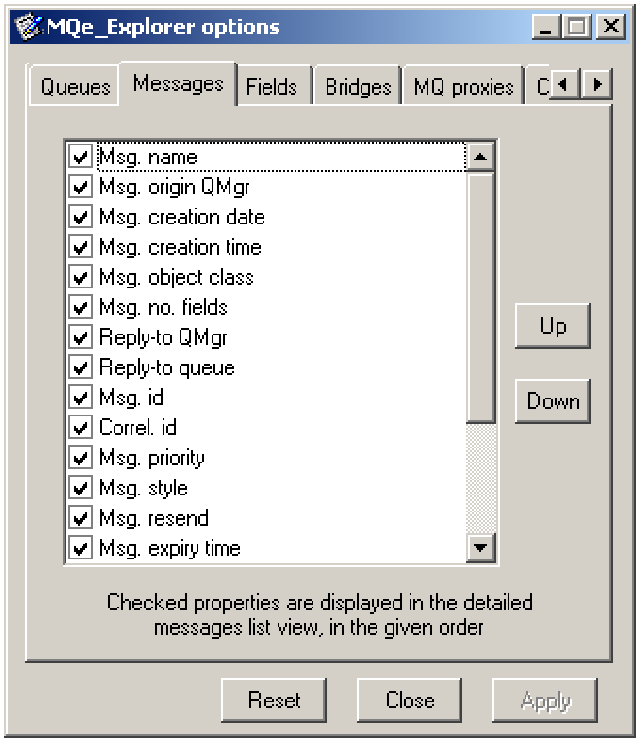
Figure 4-47: Options dialog – Listeners tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Messages

This tab controls the detailed list view display of message properties (*View→Details*), i.e. when an *application* or *bridge* queue node is selected in the tree view pane. The panel is:



☑ Msg. name
☑ Msg. origin QMgr
☑ Msg. creation date
☑ Msg. creation time
☑ Msg. object class
☑ Msg. no. fields
☑ Reply-to QMgr
☑ Reply-to queue
☑ Msg. id
☑ Correl. id
☑ Msg. priority
☑ Msg. style
☑ Msg. resend
☑ Msg. expiry time

Checked properties are displayed in the detailed messages list view, in the given order

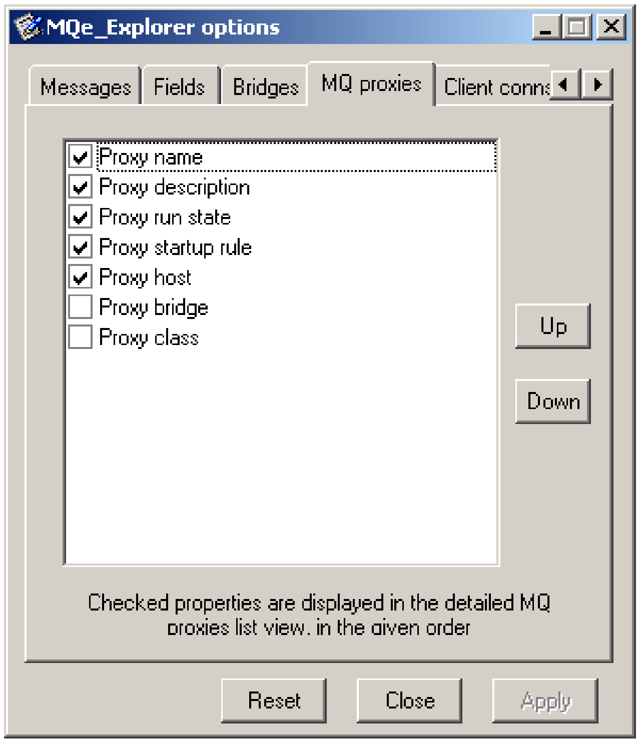
Figure 4-48: Options dialog – Messages tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

MQ proxies

This tab controls the detailed list view display of field properties (*View→Details*), i.e. when an *bridge* node is selected in the tree view pane. The panel is:



- [✓] Proxy name
- [✓] Proxy description
- [✓] Proxy run state
- [✓] Proxy startup rule
- [✓] Proxy host
- [ ] Proxy bridge
- [ ] Proxy class

Checked properties are displayed in the detailed MQ proxies list view, in the given order

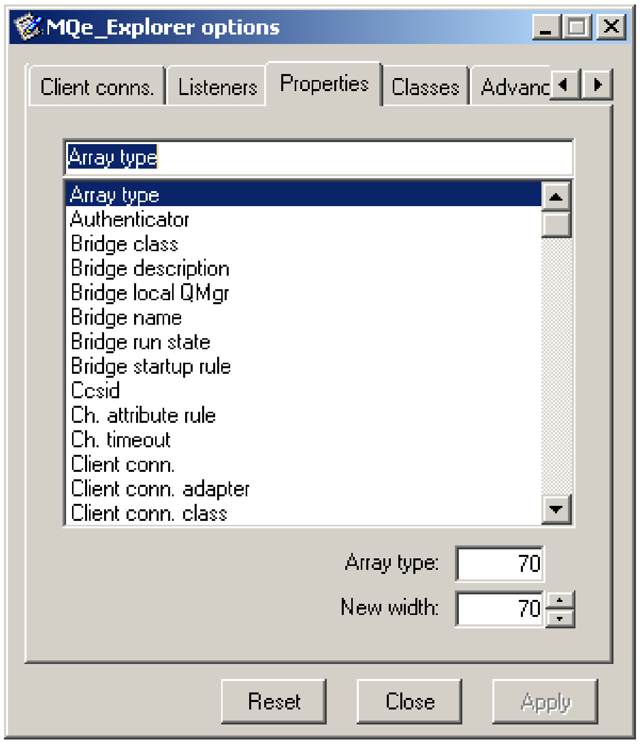
Figure 4-49: Options dialog – MQ proxies tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Properties

This tab controls the column width for a property when displayed in the detailed view format (*View→Details*) of the list view pane. To change a property either select it in the list box, or type the first few characters of its name in the input field (above the list box) and then select it. The selected property name will be displayed below the list box along width its current width, as illustrated below:



Array type

Array type
Authenticator
Bridge class
Bridge description
Bridge local QMgr
Bridge name
Bridge run state
Bridge startup rule
Ccsid
Ch. attribute rule
Ch. timeout
Client conn.
Client conn. adapter
Client conn. class

Array type: 70

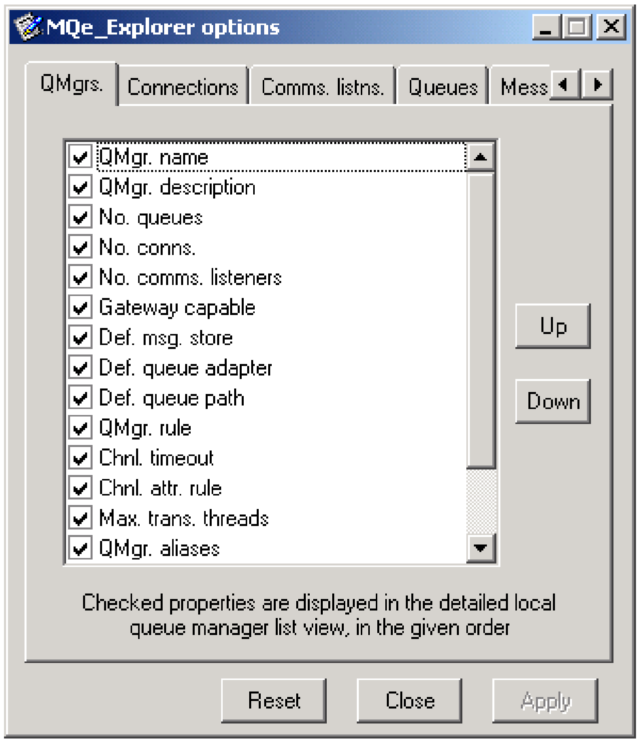
New width: 70

Figure 4-50: Options dialog – Properties tab

A new width can be set in the *New width* field. Changes only take effect when the *Apply* button is clicked. The default widths for an object property are controlled by this tab, as are the widths for a currently displayed object.

Queue managers

This tab controls the detailed list view display of queue manager properties (*View→Details*), i.e. when the *MQe root* node is selected. The panel is:



- ✓ QMgr. name
- ✓ QMgr. description
- ✓ No. queues
- ✓ No. conns.
- ✓ No. comms. listeners
- ✓ Gateway capable
- ✓ Def. msg. store
- ✓ Def. queue adapter
- ✓ Def. queue path
- ✓ QMgr. rule
- ✓ Chnl. timeout
- ✓ Chnl. attr. rule
- ✓ Max. trans. threads
- ✓ QMgr. aliases

Checked properties are displayed in the detailed local queue manager list view, in the given order

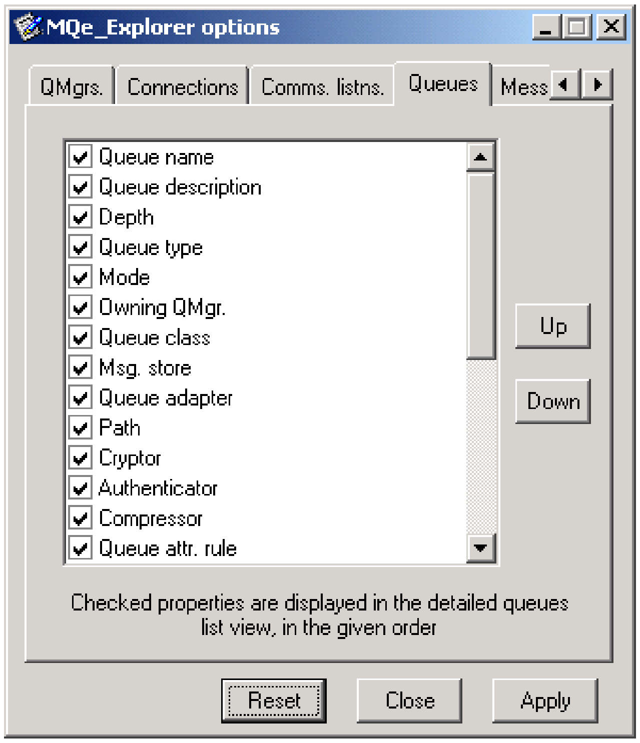
Figure 4-51: Options dialog – Local queue managers tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

Queues

This tab controls the detailed list view display of queue properties (*View→Details*), i.e. when the *Local queues* folder node is selected in the tree view pane. The panel is:



- ✓ Queue name
- ✓ Queue description
- ✓ Depth
- ✓ Queue type
- ✓ Mode
- ✓ Owning QMgr.
- ✓ Queue class
- ✓ Msg. store
- ✓ Queue adapter
- ✓ Path
- ✓ Cryptor
- ✓ Authenticator
- ✓ Compressor
- ✓ Queue attr. rule

Checked properties are displayed in the detailed queues list view, in the given order

Figure 4-52: Options dialog – Queues tab

The order of the columns shown in the list view pane matches the order of the column names shown in the tab list. The first column cannot be re-ordered; others can be re-ordered by being selected and then moved up or down by clicking either the *Up* or *Down* buttons.

Columns can be deleted from the display by un-checking their associated check box.

4.6 Window

Close All

Close All closes all open MQe_Explorer windows, excepting the main window and any of the following (if present): console, trace and options windows.

Minimize All

Minimize All minimizes all open MQe_Explorer windows, excepting the main window.

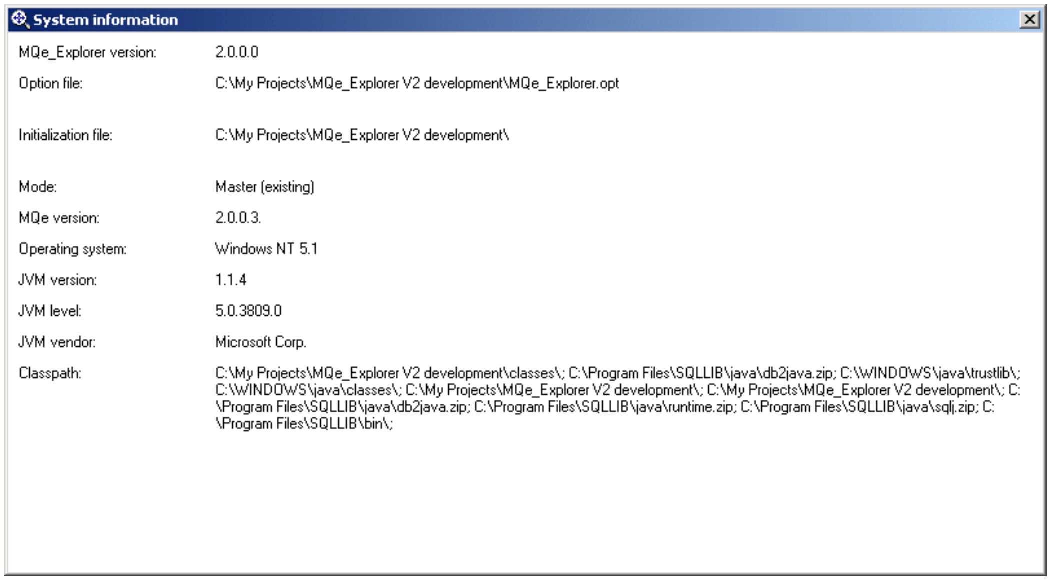
Restore All

Restore All restores all minimized MQe_Explorer windows.

4.7 Help

About MQe_Explorer

This menu item lists the program name, product identifier and version. The window is closed via the *system close* icon in the top right hand corner. The *System Info* button provides information on the local environment – IBM service staff may require this. The information available is:



MQe_Explorer version:	2.0.0.0
Option file:	C:\My Projects\MQe_Explorer V2 development\MQe_Explorer.opt
Initialization file:	C:\My Projects\MQe_Explorer V2 development\
Mode:	Master (existing)
MQe version:	2.0.0.3.
Operating system:	Windows NT 5.1
JVM version:	1.1.4
JVM level:	5.0.3809.0
JVM vendor:	Microsoft Corp.
Classpath:	C:\My Projects\MQe_Explorer V2 development\classes\; C:\Program Files\SQLLIB\java\db2java.zip; C:\WINDOWS\java\trustlib\; C:\WINDOWS\java\classes\; C:\My Projects\MQe_Explorer V2 development\; C:\My Projects\MQe_Explorer V2 development\; C:\Program Files\SQLLIB\java\db2java.zip; C:\Program Files\SQLLIB\java\runtime.zip; C:\Program Files\SQLLIB\java\sqlj.zip; C:\Program Files\SQLLIB\bin\;

Figure 4-53: System information panel

In the formatting of the *classpath* variable value, MQe_Explorer adds a space after every semi-colon – this is to enable the string to word spill across multiple rows, if necessary. If any values (such as file paths) are not completely visible, because they are long strings without embedded blanks, then the value may be clicked and the cursor moved to the right – the value will then scroll in the field. All values can be selected and copied to the system clipboard.

Supervisor mode

Closing the Help panel via the *system close* icon puts MQe_Explorer into supervisor mode; MQe+ indicates this in the status bar panel. Supervisor mode operation may be requested by IBM service personnel.

5 The list view pane

The list view pane displays the children of the object currently selected in the tree view pane. If the detailed view mode has been set, then a tabular view of their properties is also shown.

The list view pane offers a number of functions:

- Empty columns are hidden by enabling the *View→Hide Empty* menu item (or equivalent toolbar button).
- Additional alias rows are shown by enabling the *View→Use Multiple* menu item (or equivalent toolbar button).
- Admin field names are translated to meaningful strings by enabling the *View→Translate Admin* menu item (or equivalent toolbar button).
- Rows are sorted (in increasing and decreasing column order) by clicking (or re-clicking) the column header.
- Column widths are changed by dragging the column boundaries.
- Default column widths for properties are set in *Tools→Options→Properties.*
- Columns are temporarily re-ordered by dragging the column titles.
- Columns are permanently re-ordered (or hidden) using *Tools→Options→…* (depending upon the type of property).

For most objects only one row may be selected at a time; however for message rows, multiple select is enabled.

Drag and drop operations (i.e. a move with either no key or with the *Shift* key held down; a copy with the *Control* key held down) are supported for message(s) from the list view pane to certain queue nodes in the tree view pane. The precise details of message move and copy parallel those for the equivalent operations undertaken via the clipboard (see the section on the *Edit* menu operation on page 62). The conditions for a tree view node to be acceptable as a message drop target are:

- It must be a queue of type:
  - Application.
  - Async. proxy.
  - Sync. proxy.
  - Bridge.
- It must not already own the messages.

The status bar indicates the progress and completion status of drag and drop operations.

5.1 Bridges

Information on bridges is displayed in the list view pane when the *Bridges folder* is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Bridges*; their default width is defined in *Tools→Options→Properties)*. In default order:

Bridge name:
The name of the bridge.

Bridge description:
A description of the bridge.

Bridge run state:
Object stopped or started.

Bridge startup rule:
The rule class (or alias) that determines whether the object and its children are started at object creation time or when the queue manager is opened.

Default transformer:
The default transformer class (or alias) used for message conversion.

Heartbeat:
Bridge resources heartbeat pulse interval (minutes).

Bridge local QMgr:
The name of the queue manager owning the definition.

Bridge class:
The object class (or alias).

5.2 Client connections

Information on client connections is displayed in the list view pane when an *MQ proxy* is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Client Conns.*; their default width is defined in *Tools→Options→Properties)*. In default order:

Client conn. name:
The name of the client connection.

Client conn. descr:
A description of the client connection.

Client conn. run state:
Object stopped or started.

Client conn. startup rule:
The rule class (or alias) that determines whether the object and its children are started at object creation time or when the queue manager is opened.

Client conn. port:
The IP port number used by the target WebSphere MQ queue manager.

Bridge adapter:
The bridge adapter class (or alias) used to move messages from MQe to the target WebSphere MQ queue manager.

User id:
The user id. used by WebSphere MQ.

Send exit:
The send exit specified at the remote end of the WebSphere MQ client channel.

Receive exit:
The receive exit specified at the remote end of the WebSphere MQ client channel.

Security exit:
The security exit specified at the remote end of the WebSphere MQ client channel.

Ccsid:
The CCSID property used by WebSphere MQ.

Sync. queue:
The name of the synchronization queue on the WebSphere MQ queue manager used by the MQ bridge.

Purger rule:
The rule class (or alias) that is used when a message on the sync. queue indicates a failure of MQe to confirm a message.

Client conn. max. idle:
The time after which an idle connection to WebSphere MQ is discarded and the resources returned to the pool (minutes).

Purger intvl:
The time between successive purges of the synchronization queue (minutes).

Client conn. proxy:
The name of the MQ proxy owning the definition.

Client conn. class:
The object class (or alias).

5.3 Communication listeners

Information on communication listeners is displayed in the list view pane when the *Comms. listeners* folder is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Comms. Listeners*; their default width is defined in *Tools→Options→Properties)*. In default order:

Comms. name:
The name of the comms. listener.

Comms. description:
A description of the comms. listener.

Comms. run state:
Object stopped or started.

Comms. timeout:
The time after which an idle incoming connection will be timed out (milliseconds).

Comms. adapter:
The adapter class (or alias) of the communications protocol adapter.

Comms. port:
The IP port number used by the comms. listener to service incoming connection requests.

Comms. no. chnls:	The current number of channels associated with this comms. listener.
Comms. max. chnls:	The max. number of channels allowed for this comms. listener.
Comms. class:	The object class (or alias) that handles incoming connections.

5.4 Connections

Information on connections is displayed in the list view pane when the *Connections* folder is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Connections*; their default width is defined in *Tools→Options→Properties)*. In default order:

Conn. name:	The name of the connection (and therefore the name or an alias of the remote queue manager). If the *View→Use Multiple* option is enabled then *Name* may also be an alias name for the connection (accompanied by a shortcut connection icon).
Conn. description:	A description of the connection.
Conn. type:	The connection type (direct, indirect, etc).
Conn. local qMgr:	The name of the local queue manager owning the connection definition.
Conn. chnl. class:	The channel class (or alias) to be used in the connection.
Conn. adapter:	The class (or alias) of the communications protocol adapter.
Conn. address:	For a direct connection, the numeric or string IP address of the machine hosting the remote queue manager; for an indirect connection, the name of the queue manager to be used.
Conn. port:	The IP port number used by the remote queue manager to service incoming connection requests.
Conn. options:	Options to be passed to the communications protocol adapter.
Conn. ascii parms:	ASCII parameters to be passed to the communications protocol adapter.
Conn. aliases:	Alias names for the connection.
Conn. base:	The name of the connection.

5.5 Fields

Information on fields is displayed in the list view pane when either a message or a fields field is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Fields*; their default width is defined in *Tools→Options→Properties*). In default order:

Field name:	The name of the field.
Data type:	The data type of the field.
Value:	The value of the field.
Length:	The length of the array or string.
Array type:	Indicates whether an array is static or dynamic.
Modifier:	The field modifier.
Hidden:	Indicates whether the field is hidden.
Field attribute:	The attribute object associated with the fields object.
Field compressor:	The compressor class associated with the attribute object.
Field cryptor:	The cryptor class associated with the attribute object.
Field authenticator:	The authenticator class associated with the attribute object.
Field attr. rule:	The rule class associated with the attribute object.

5.6 Listeners

Information on listeners is displayed in the list view pane when a *client connection* is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Listeners*; their default width is defined in *Tools→Options→Properties)*. In default order:

Listener name:	The name of the listener.
Listener description:	A description of the listener.
Listener run state:	Object stopped or started.
Listener startup rule:	The rule class (or alias) that determines whether the object and its children are started at object creation time or when the queue manager is opened.

Dead letter queue:
The WebSphere MQ queue used to hold messages that cannot be delivered from WebSphere MQ to MQe.

State store:
The specification of permanent storage used to hold state information as messages are moved from WebSphere MQ to MQe.

Undelivered msg. rule: The class (or alias) determining the action to be taken when a message cannot be delivered from WebSphere MQ to MQe.

Listener transformer:
The class (or alias) used for message conversion from WebSphere MQ to MQe.

Flows per commit:
the number of messages flowed after which the MQ sync. queue (if used) is cleaned.

Listener client conn:
the name of the client connection owning the definition.

Listener class:
The object class (or alias).

5.7 Messages

Information on messages is displayed in the list view pane when either an *application queue* or a *bridge queue* is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Messages*; their default width is defined in *Tools→Options→Properties)*. These columns show not just the message object properties but also anticipate that certain common fields may be also present. The full set of fields for a particular message is available by selecting the message itself in the tree view pane. In default order:

Msg. name:
A number assigned by MQe_Explorer to a message, reflecting its priority and position in the queue. Thus message number 1 is the first available message – true at the time that the cached snapshot was taken.

Msg. origin QMgr:
The queue manager that created the message.

Msg. creation date:
The date the message was created.

Msg. creation time:
The time the message was created.

Msg. object class:
The object class of the message.

Msg. no. fields:
The number of fields in the messages.

Reply-to QMgr:
The name of the queue manager to which a reply should be sent.

Reply-to queue:
The name of the queue to which a reply should be sent.

Msg. id:
The UID of the message.

Correl. id:
The correlation id.

Msg. priority:
The value of the priority field.

Msg. style:
The style of the message (for example, *Reply*, *Request* etc).

Msg. resend:
Indicates whether the message is a resend of a previous message.

Msg. expiry time:
The relative time when the message will expire (millisecs), or the date when it will expire.

Msg. lock id:
The lock id.

Msg. attribute:
The attribute object associated with the message object.

Msg. compressor:
The compressor class associated with the attribute object.

Msg. cryptor:
The cryptor class associated with the attribute object.

Msg. authenticator:
The authenticator class associated with the attribute object.

Msg. attr. rule:
The rule class associated with the attribute object.

5.8 MQ proxies

Information on MQ proxies is displayed in the list view pane when a *bridge* is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→MQ Proxies*; their default width is defined in *Tools→Options→Properties)*. In default order:

Proxy name:
The name of the MQ proxy.

Proxy description:
A description of the MQ proxy.

Proxy run state:
Object stopped or started.

Proxy startup rule:
The rule class (or alias) that determines whether the object and its children are started at object creation time or when the queue manager is opened.

Proxy host:
The IP address of the target WebSphere MQ queue manager.

Proxy bridge:
The name of the bridge owning the definition.

Proxy class:
The object class (or alias).

5.9 Queue manager folders

The queue manager folders are displayed when a *queue manager* is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available; in default order:

Folder name: The folder name.

Number: The number of objects associated with the folder*:*

- *Bridges:* The number of bridges present.
- *Connections*: The number of connections.
- *Comms*. listeners: The number of communications listeners.
- *Local queues*: The number of local queues.

Status: *Stopped* or *running* (only applicable to the bridges object).

5.10 Queue managers

Information on queue managers is displayed in the list view pane when the *MQe root node* is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Local qMgrs*; their default width is defined in *Tools→Options→Properties)*. In default order:

QMgr name:
The name of the local queue manager. If the *View→Use Multiple* option is enabled then *Name* may also be an alias name for the queue manager (accompanied by a shortcut queue manager icon).

QMgr description:
A description of the queue manager.

No. queues:
The number of queues (both local and remote) owned by the queue manager.

No. conns:
The number of connections owned by the queue manager.

No. comms. listeners:
The number of communications listeners owned by the queue manager.

Gateway enabled:
Indicates whether the queue manager can support gateway function.

Def. msg. store:
The default message store class (or alias) for a queue.

Def. queue adapter:
The default storage adapter class (or alias) for a queue.

Def. queue path:
The default path describing the physical location of queues.

QMgr rule:
The rule class (or alias) to be used by the queue manager.

Chnl.. timeout:
The time after which an outgoing idle channel will be timed out (milliseconds).

Chnl. attr. rule:
The rule class (or alias) to be associated with the channel attribute.

Max. trans. threads:
The maximum number of threads that will be spawned to service the transmission needs of the queue manager.

QMgr aliases:
Alias names for the queue manager.

QMgr base:
The name of the queue manager.

QMgr class:
The class (or alias) of the queue manager.

Version:
The version of MQe hosting the queue manager.

5.11 Queues

Information on local queues is displayed in the list view pane when the *Local queues* folder is selected in the tree view pane. In the detailed view (*View→Details*) the following columns are available (the ones shown and their order is defined in *Tools→Options→Queues*; their default width is defined in *Tools→Options→Properties).* Note that the columns used are also common to those for remote queues; not all columns are relevant to all queue types. In default order:

Queue name:
The name of the queue. If the *View→Use Multiple* option is enabled then *Name* may also be an alias name for the queue (accompanied by a shortcut queue icon).

Queue description:
A description of the queue.

Depth:
The number of messages on the queue.

Queue type:
Indicates the type of the queue, for example: application, admin, etc.

Partner QMgr[21]*:*
For a sync. or async. proxy queue or bridge queue – the message destination; for a forward queue – the next hop queue manager; for a home server queue – the queue manager from which messages will be pulled.

Queue class:
The class (or alias) implementing the queue object.

Msg. store:
The message store class (or alias).

[21] In MQe terminology the partner queue manager is referred to as the queue queue manager.

Queue adapter:	The queue adapter class (or alias).
Path:	The path locating the physical storage of queues – passed to the queue adapter.
Cryptor:	The cryptor class (or alias) associated with the queue.
Authenticator:	The authenticator class (or alias) associated with the queue.
Compressor:	The compressor class (or alias) associated with the queue.
Queue attr. rule:	The attribute rule class (or alias) associated with the queue.
Max. msg. size:	The maximum size of a message on the queue.
Max. depth:	The maximum number of messages on the queue.
Priority:	The default priority to be associated with a message.
Status:	Indicates whether the queue is *active* or *inactive*.
Creation date:	The date/time the queue object was created.
Queue rule:	The rule class (or alias) to be used by the queue.
Target registry:	The registry to be used by the authenticator.
Expiry:	The time after which messages expire (milliseconds).
Time interval:	For a home server queue, the time between attempting to retrieve messages; for an admin queue, the time between message processing attempts (milliseconds).
Transporter:	The transporter class (or alias) to be used (this class handles the getting and putting of messages to/from remote queues).
Destinations:	The queue manager destinations for which a store or forward queue will hold messages.
Bridge:	The bridge object that handles the target MQ queue.
MQ QMgr:	The target MQ queue manager.
Client conn:	The client connection.
MQ queue:	The target MQ queue.

Transformer:	The transformer class (or alias) converting the message from MQe to MQ format.
Max. idle:	The maximum time that the MQ bridge queue can hang on to an idle connection before it is returned to the connection pool (seconds).
Queue aliases:	A list of alias names for the queue.
Queue base:	The name of the queue.
Local QMgr:	The name of the queue manager that owns the queue definition.

5.12 Remote queues

Information on remote queues is displayed in the list view pane when a *Connection* is selected in the tree view pane. In the detailed view (*View→Details*) the columns available are as for the *Local queues* list view pane (see page 90). The properties shown and their order is defined as for local queues, i.e. in *Tools→Options→Queues*; default widths are defined in *Tools→Options→Properties)*.

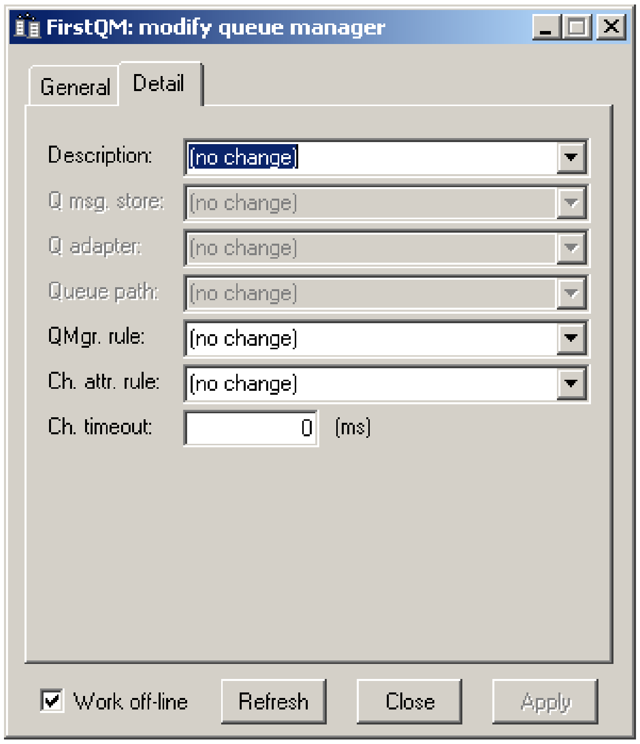
6 Offline administration

MQe_Explorer supports the offline administration of MQe objects[22] by asynchronously sending admin messages to queue managers that are not currently online. In order to use offline administration both the local queue manager hosting MQe_Explorer and the remote queue manager must have been appropriately configured (see *Managing remote queue managers* on page 113). With the notable exception of object deletion, offline administration uses property pages – just as for online administration. The most obvious difference however is that that MQe_Explorer can no longer pre-load these pages with the current property values.

Offline working is activated in various ways. To change the properties of an unavailable queue manager (identified by the icon), right click on the queue manager icon (or use the menus) and select the *Properties* menu item. To manipulate other objects belonging to the unavailable queue manager, expand the tree to the first folder level (*Bridges*, *Comms. listeners*, *Connections*, *Local queues*) in the tree or list view. Then, for example, right clicking on the *Connections* folder (or via the menus) enables connection properties to be created, deleted or changed. Similarly, the *Local queues* folder enables both and remote queues to be created deleted or changed; likewise for communications listeners. Note that these menu items are only enabled for unavailable queue managers. Offline administration is not supported for bridges and related objects.

An alternative technique is to bring up the appropriate properties page and check the *Offline working* box in the bottom left.

Where possible, the properties are pre-loaded with the *(no change)* option[23], as seen below:


FirstQM: modify queue manager

Description: (no change)
Q msg. store: (no change)
Q adapter: (no change)
Queue path: (no change)
QMgr. rule: (no change)
Ch. attr. rule: (no change)
Ch. timeout: 0 (ms)

[✓] Work off-line

Figure 6-1: Offline administration – modify queue manager

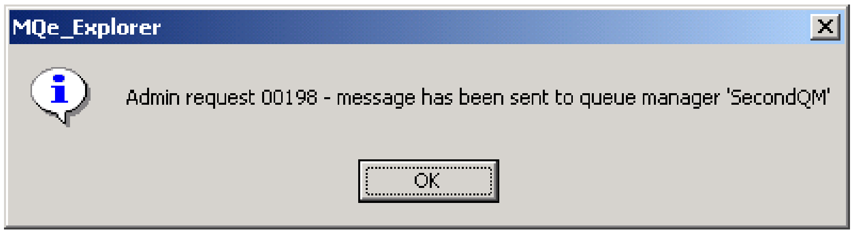
[22] Offline administration of MQe bridge-related objects is not supported.
[23] Numeric values cannot be pre-loaded in this way. However if the operator does not change the displayed value, the resulting admin message will not set that parameter.

In some cases collections of parameters are modified in their entirety. For example, when changing a connection, the IP address alone cannot be changed (such that other adapter-related parameters are left with their existing values). Unless otherwise specified, all adapter parameters will be set to the defaults shown in the property pages.

Through offline working it is not possible to change aliases, nor to change the destinations associated with store or forward queues.

When changes are applied the messages are sent asynchronously to the remote admin queue using the offline admin queue name set up in the *Tools→Options→Advanced-1→Admin request queue used* group. For the configuration aspects of this *see Admin and reply queues* on page 113.

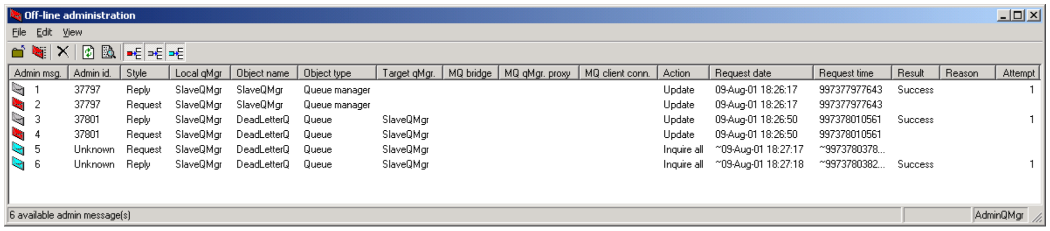
A message box appears assigning an admin request number to the request – this number increases non-sequentially and wraps. It is not settable by the operator but it is significant in tracking the progress of an offline admin request.



Admin request 00198 - message has been sent to queue manager 'SecondQM'

Figure 6-2: Offline administration – admin request assignment

The offline administration window shows the progress of offline admin requests. This window is accessed through the *View→Offline* admin menu item. A row appears for each request, and for each response[24]; the request number correlates the rows. The particular message types shown are determined by the checked *View* menu items (or their associated toolbar buttons).



Admin msg.	Admin id.	Style	Local qMgr	Object name	Object type	Target qMgr.	MQ bridge	MQ qMgr. proxy	MQ client conn.	Action	Request date	Request time	Result	Reason	Attempt
1	37797	Reply	SlaveQMgr	SlaveQMgr	Queue manager					Update	09-Aug-01 18:26:17	997377977643	Success		1
2	37797	Request	SlaveQMgr	SlaveQMgr	Queue manager					Update	09-Aug-01 18:26:17	997377977643			
3	37801	Reply	SlaveQMgr	DeadLetterQ	Queue	SlaveQMgr				Update	09-Aug-01 18:26:50	997378010561	Success		1
4	37801	Request	SlaveQMgr	DeadLetterQ	Queue	SlaveQMgr				Update	09-Aug-01 18:26:50	997378010561			
5	Unknown	Request	SlaveQMgr	DeadLetterQ	Queue	SlaveQMgr				Inquire all	~09-Aug-01 18:27:17	~9973780378...			
6	Unknown	Reply	SlaveQMgr	DeadLetterQ	Queue	SlaveQMgr				Inquire all	~09-Aug-01 18:27:18	~9973780382...	Success		1

6 available admin message(s) AdminQMgr

Figure 6-3: The offline administration window

The example[25] shows two requests, 37797 and 37801, identified by the icon ; replies have been received. Successful replies are associated with the icon ; unsuccessful replies with . Two 'unknown' messages are also shown, indicated by the – in this case these are copies that have been requested of messages used by MQe_Explorer internally (see later). The following columns are displayed:

Admin message:

A sequential numbering of the admin requests and responses, with an

[24] In principle multiple responses can be received to an admin request; in MQe v1 – v2 only a single response may be expected.
[25] For simplicity, a number of columns have been set to zero width and others re-ordered.

associate status icon (operation successful, partially unsuccessful or failed).

Admin id:
An admin identifier for the request/response.

Style:
Request or response.

Local queue manager:
The name of the queue manager holding the managed object.

Object name:
The name of the managed object (for the bridges object the name of the local queue manager is displayed).

Object type:
The type of the managed object (queue manager etc).

Partner queue manager:
For a sync. proxy queue - the destination.
For an async. proxy queue - the destination.
For a forward queue – the forward-to queue manager.
For a home server queue – the get-from queue manager.
For a connection – the destination of the connection.
For an MQ proxy – the MQ queue manager.
For a client connection – the MQ queue manager.
For a listener – the MQ queue manager.

Bridge:
The name of the associated bridge.

MQ proxy:
The name of the associated MQ proxy.

Client connection:
The name of the associated client connection.

Action:
*T*he action to be performed (update, delete, inquire, etc).

Request date:
The time the admin request was made, formatted as date/time. Where the admin id is unknown, the time is approximated to the request object creation time.

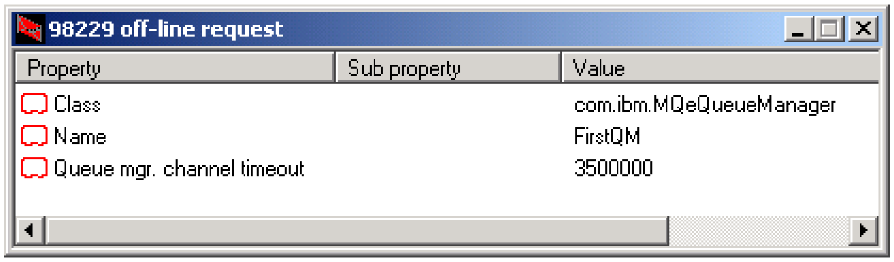
Request time:
The time the admin request was made, formatted as an integer. Where the admin id is unknown, the time is approximated to the request object creation time.

Result:
*O*ne of: “Success”, “Mixed”, “Fail”.

Reason:
An elaboration of *Result*.

Attempt:
The admin operation attempt number for which this is the response.

Further details on requests and responses are shown if the relevant row is double clicked (or activated through the *View→Detail* menu item, or button). The properties to be managed are displayed – together with details of any errors that occurred, for example:



Property	Sub property	Value
Class		com.ibm.MQeQueueManager
Name		FirstQM
Queue mgr. channel timeout		3500000

Figure 6-4: Offline request detail

The offline administration has many features in common with the main MQe_Explorer window. Columns can be re-ordered by dragging a column heading across other headings; clicking a column header sorts (and re-sorts) the data; one or more rows can be selected as required. Rows can be deleted or the display cleared. *File→Resend* or the button retries the admin operation; *View→Refresh* or the button refreshes the display. Context menus are supported.

The status bar at the bottom of the display has three panels; the left panel shows the number of status messages, the middle panel identifies the number of rows selected, and the right panel the hosting queue manager name (shown in [] brackets when MQe_Explorer is running as a slave).

6.1 Message properties

The offline admin window displays all messages on the *AdminReplyQ* that have all of the following properties:

- The message class is *MQeAdminMsg* (or a subclass thereof).
- The message contains the field **MQe_Explorer* (i.e. it was created by MQe_Explorer).
- The message contains the field **Mode* (i.e. it is used by MQe_Explorer for administration or is a copy of such a message).

These conditions mean that all offline requests and replies will be shown, as will all copies made of MQe_Explorer admin messages (i.e. those copies made by setting *Tools→Demo Mode*).

Messages are classified as follows:

- Messages with a non-zero **Msg_CorrelID* and a **Msg_Style* of ‘Request’ are *Requests*.
- Messages with a non-zero **Msg_CorrelID* and a **Msg_Style* of ‘Reply’ are *Replies*.
- All other messages are *Others*.

7 Offline administration menus

The offline admin window shares many of the properties of the main list view pane. Thus columns may be sorted (or re-sorted) by clicking, columns may be re-ordered by dragging their headers, and rows may be double-clicked for additional details. Certain operations support multiple selected rows.

7.1 Edit

Clear

Clear deletes all the messages displayed in the offline admin window; other offline messages may still exist however, depending upon the current value of the view options (i.e. *Requests*, *Replies*, *Others*).

Delete

Delete deletes all the selected messages in the offline admin window.

Select All

Select All selects all the messages in the offline admin window.

7.2 File

Exit

Exit removes the offline admin window.

7.3 View

Detail

Detail provides additional information on both admin request and replies.

Refresh

Refresh updates the admin offline replies window with current messages.

Requests

If checked, then offline request messages are displayed.

Replies

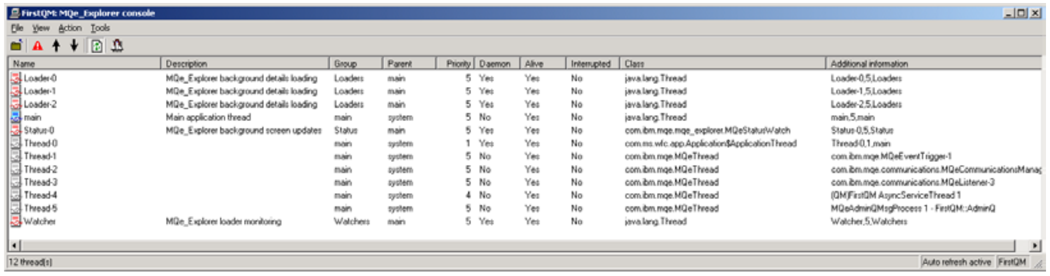
If checked, then offline reply messages are displayed.

Others

If checked, then offline other messages are displayed (i.e. messages that are not identifiable as either offline admin requests or replies).

8 Console

The MQe_Explorer console provides a means of viewing and managing local applications. Since all applications are classes that have been loaded and assigned to threads, the console presents the information as a status list of JVM threads. The JVM allows MQe_Explorer to have knowledge only of threads that it owns, for example, the main thread used for its execution, or threads that either it has created, or that descend from its own threads. This means that the console cannot always show all threads present in a JVM. However, if MQe_Explorer was used to instantiate the JVM (and indeed the queue manager), then the console will present a complete picture. An example of a console window is shown below:



Name	Description	Group	Parent	Priority	Daemon	Alive	Interrupted	Class	Additional information
Loader-0	MQe_Explorer background details loading	Loaders	main	5	Yes	Yes	No	java.lang.Thread	Loader-0,5,Loaders
Loader-1	MQe_Explorer background details loading	Loaders	main	5	Yes	Yes	No	java.lang.Thread	Loader-1,5,Loaders
Loader-2	MQe_Explorer background details loading	Loaders	main	5	Yes	Yes	No	java.lang.Thread	Loader-2,5,Loaders
main	Main application thread	main	system	5	No	Yes	No	java.lang.Thread	main,5,main
Status-0	MQe_Explorer background screen updates	Status	main	5	Yes	Yes	No	com.ibm.mqe.mqe_explorer.MQeStatusWatch	Status-0,5,Status
Thread-0		main	system	1	Yes	Yes	No	com.ms.wfc.app.Application$ApplicationThread	Thread-0,1,main
Thread-1		main	system	5	No	Yes	No	com.ibm.mqe.MQeThread	com.ibm.mqe.MQeEventTrigger-1
Thread-2		main	system	5	No	Yes	No	com.ibm.mqe.MQeThread	com.ibm.mqe.communications.MQeCommunicationsManag
Thread-3		main	system	5	No	Yes	No	com.ibm.mqe.MQeThread	com.ibm.mqe.communications.MQeListener-3
Thread-4		main	system	4	No	Yes	No	com.ibm.mqe.MQeThread	(QM)FirstQM AsyncServiceThread 1
Thread-5		main	system	5	No	Yes	No	com.ibm.mqe.MQeThread	MQeAdminQMsgProcess 1 - FirstQM::AdminQ
Watcher	MQe_Explorer loader monitoring	Watchers	main	5	Yes	Yes	No	java.lang.Thread	Watcher,5,Watchers

12 thread(s) | Auto refresh active | FirstQM

Figure 8-1: The console window

Each row represents a thread with the icons distinguishing various kinds. indicates a thread running a class loaded by the user, a thread used by MQe_Explorer, the main thread and is used for any other threads present (including those used by MQe). The columns give information about the thread:

Name:	The name for threads created by MQe_Explorer. The name normally has a suffix which is a hexadecimal number increased sequentially as each thread of that type is created (starting at 0). Threads running user-loaded classes are named with the class name.
Description:	Further information on those threads known to MQe_Explorer.
Group:	The name of the group to which the thread belongs.
Parent:	The name of the parent thread.
Priority:	Thread priority (typically limited to the range of 0 – 10).
Daemon:	Indicates if the thread is a daemon, as opposed to a user thread.
Alive:	Indicates if the thread is alive.

Interrupted:

Indicates if the thread is interrupted.

Class:

The thread class (threads running user-loaded classes and launched by MQe_Explorer always have the class name *java.lang.Thread*).

Additional information:

The edited output from a *toString()* call on the thread object (this gives useful information on MQe threads).

The status bar has three sub-panels; at the left is a status area used for messages and information; next is the auto-refresh status (see below); the rightmost sub-panel shows the name of the local queue manager (enclosed in [] if MQe_Explorer is running as a slave).

By default the panel is auto-refreshed, i.e. updated in the background without the need for user-interaction (and indeed the thread doing this – and other updates – is visible in the panel under the name *Status-x*).

Columns can be re-arranged by dragging the column headers; they can be sorted (and re-sorted) by clicking the same headers – however auto-refresh must be disabled before sorting is permitted.

Threads can be selected one at a time by clicking anywhere on the row. When selected, various operations are available:

Increase/decrease priority:

Increases/decreases the likelihood of the thread being scheduled.

Interrupt:

Interrupts the thread. The effect of this is entirely dependent upon the class being executed; interrupt behavior must be programmed into the class. In the case of the *Performance tool* shipped with MQe_Explorer, interrupting the thread will cause the tool to close immediately (if running a test) or to abort as soon as a new test is started.

Operations can be initiated from the menus, the toolbar, or by right clicking and bringing up the context menu. Certain operations are not allowed on some threads in order to protect the integrity of the system. If auto-refresh is enabled, then it is automatically suspended whilst thread operations are pending.

The Console window allows user classes to be launched through the *Tools→Load* menu item (or via the toolbar) – this is an alternative route to the class load function available from the main MQe_Explorer window.

9 Console menus

The console window shares many of the properties of the main list view pane. Thus columns may be sorted (or re-sorted) by clicking; likewise columns may be re-ordered by dragging their headers. Only one console window may be present.

9.1 Action

Increase Priority

Increase Priority increases the selected thread priority by one, and then updates the display.

Decrease Priority

Decrease Priority decreases the selected thread priority by one, and then updates the display.

Interrupt

Interrupt interrupts the selected thread, and then updates the display.

9.2 File

Exit

Exit removes the console window.

9.3 Tools

Load

Load is identical to *Tools→Load* available in the main MQe_Explorer window.

9.4 View

Refresh

When checked auto-refresh is enabled (i.e. the thread display is updated automatically); when unchecked, no updates take place.

10 Trace

The trace window provides a means of debugging MQe applications (and MQe_Explorer and MQe). Enabling trace through the *View→Trace* menu item of the main window, replaces any existing MQe trace handler with an MQe_Explorer trace handler. This exploits the *com.ibm.mqe.trace.MQeTraceRenderer* and *com.ibm.mqe.trace.MQeTraceToReadable* classes, to direct the trace output to a subclass of *java.io.PrintStream* that presents the information in a rich text format (*.rtf*) display area. Data from this area may be cut, copied or pasted to/from other applications, for example Microsoft Notepad, Wordpad or Word. Text may also be highlighted, colored, annotated, cleared, searched etc. MQe_Explorer supports a subset of text editing capabilities, including word select, deletion, backspacing, tab insertions, etc. The contents of the area can be saved as a file in plain or rich text format.

Additionally or alternatively, the trace output can be logged to a file; logging is continuous once started until stopped and this independent of whether the visual trace is interrupted or edited. Both output methods however reflect the same choice of events to be traced and both dynamically respond to changes in event selection. The file output is available in text or binary formats.

MQe_Explorer supports the following trace events; combinations of these events are permitted:

All events.

Custom trace group events.

Information group events.

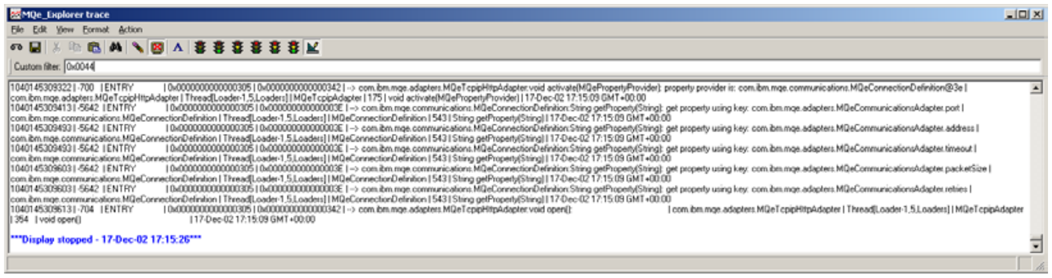
Error group events.

Exception group events.

Warning group events.

MQe default trace events.

An example of the trace window is shown below:



Custom filter: 0x0044

```
1040145309322 | -700  | ENTRY        | 0x0000000000000305 | 0x0000000000000342 | -> com.ibm.mqe.adapters.MQeTcpipHttpAdapter.void activate(MQePropertyProvider): property provider is: com.ibm.mqe.communications.MQeConnectionDefinition@3e |
com.ibm.mqe.adapters.MQeTcpipHttpAdapter | Thread[Loader-1,5,Loaders] | MQeTcpipAdapter | 175 | void activate(MQePropertyProvider) | | 17-Dec-02 17:15:09 GMT+00:00
1040145309413 | -5642 | ENTRY        | 0x0000000000000305 | 0x000000000000003E | -> com.ibm.mqe.communications.MQeConnectionDefinition:String getProperty(String): get property using key: com.ibm.mqe.adapters.MQeCommunicationsAdapter.port |
com.ibm.mqe.communications.MQeConnectionDefinition | Thread[Loader-1,5,Loaders] | MQeConnectionDefinition | 543 | String getProperty(String) | | 17-Dec-02 17:15:09 GMT+00:00
1040145309493 | -5642 | ENTRY        | 0x0000000000000305 | 0x000000000000003E | -> com.ibm.mqe.communications.MQeConnectionDefinition:String getProperty(String): get property using key: com.ibm.mqe.adapters.MQeCommunicationsAdapter.address |
com.ibm.mqe.communications.MQeConnectionDefinition | Thread[Loader-1,5,Loaders] | MQeConnectionDefinition | 543 | String getProperty(String) | | 17-Dec-02 17:15:09 GMT+00:00
1040145309493 | -5642 | ENTRY        | 0x0000000000000305 | 0x000000000000003E | -> com.ibm.mqe.communications.MQeConnectionDefinition:String getProperty(String): get property using key: com.ibm.mqe.adapters.MQeCommunicationsAdapter.timeout |
com.ibm.mqe.communications.MQeConnectionDefinition | Thread[Loader-1,5,Loaders] | MQeConnectionDefinition | 543 | String getProperty(String) | | 17-Dec-02 17:15:09 GMT+00:00
1040145309603 | -5642 | ENTRY        | 0x0000000000000305 | 0x000000000000003E | -> com.ibm.mqe.communications.MQeConnectionDefinition:String getProperty(String): get property using key: com.ibm.mqe.adapters.MQeCommunicationsAdapter.packetSize |
com.ibm.mqe.communications.MQeConnectionDefinition | Thread[Loader-1,5,Loaders] | MQeConnectionDefinition | 543 | String getProperty(String) | | 17-Dec-02 17:15:09 GMT+00:00
1040145309603 | -5642 | ENTRY        | 0x0000000000000305 | 0x000000000000003E | -> com.ibm.mqe.communications.MQeConnectionDefinition:String getProperty(String): get property using key: com.ibm.mqe.adapters.MQeCommunicationsAdapter.retries |
com.ibm.mqe.communications.MQeConnectionDefinition | Thread[Loader-1,5,Loaders] | MQeConnectionDefinition | 543 | String getProperty(String) | | 17-Dec-02 17:15:09 GMT+00:00
1040145309613 | -704  | ENTRY        | 0x0000000000000305 | 0x0000000000000342 | -> com.ibm.mqe.adapters.MQeTcpipHttpAdapter.void open():                    | com.ibm.mqe.adapters.MQeTcpipHttpAdapter | Thread[Loader-1,5,Loaders] | MQeTcpipAdapter
| 354  | void open()                        | 17-Dec-02 17:15:09 GMT+00:00

***Display stopped - 17-Dec-02 17:15:26***
```

Figure 10-1: The trace window

The window has a limit on how much trace data can be displayed – and be aware that excessive data impacts performance. It is good practice to log the entire trace activity to a file and concurrently stop/start the display trace as needed, using the toolbar button (or the menu item *View→Display Trace*). A button is also available to clear the trace display (or the menu item *Edit→Clear*).

Trace events are selected using a range of a selection of traffic light buttons (or items from the *Action* menu. If custom trace events are required the custom trace group is specified in the edit box on the rebar line, directly underneath the toolbar. The specification format is either hexadecimal (using the Java hexadecimal format syntax with the 0x prefix, e.g. 0x000000000000000A – leading zeros can be omitted), or decimal (e.g. 10).

Applications can write directly to the trace window as a *java.io.PrintStream* object; for more details see *Class com.ibm.mqe.mqe_explorer.MQeExplTrace* on page 172.

11 Trace menus

11.1 Action

All

All causes all trace events to be captured and displayed or logged as appropriate.

Custom

Custom causes trace events belonging to the custom trace group to be captured and displayed or logged as appropriate. The custom trace group is specified in the edit box on the rebar line, directly underneath the toolbar; the specification format is either hexadecimal (using the Java hexadecimal format syntax with the 0x prefix, e.g. 0x000000000000000A – leading zeros can be omitted) or decimal (e.g. 10).

Default

Default causes those trace events type determined by MQe as belonging to the default trace group (i.e. all events except those relating to *WebSphere MQ*) to be captured and displayed or logged as appropriate.

Error

Error causes those trace events belonging to the error trace group to be captured and displayed or logged as appropriate.

Exception

Exception causes those trace events belonging to the exception trace group to be captured and displayed or logged as appropriate.

Information

Information causes those trace events belonging to the information trace group to be captured and displayed or logged as appropriate.

Warning

Warning causes those trace events belonging to the warning trace group to be captured and displayed or logged as appropriate.

11.2 Edit

Clear

Clear clears the trace display area.

Copy

Copy copies the currently selected text in the display area to the system clipboard. The text is stored in both rich text and plain text formats.

Cut

Cut cuts the currently selected text in the display area to the system clipboard. The text is stored in both rich text and plain text formats.

Delete

Delete deletes the currently selected text in the display area.

Find

Find locates a text string in the display area and, if found, leaves the result selected. If *Find* is invoked with text selected, then that selected string is primed as the search string. All search strings used in an invocation of *Find* are available for re-use in the drop down list box. Repeat identical searches begin where the previous search finished, until all the target text area has been searched. Options are provided for case matched and whole word searches.

Paste

Paste pastes text from the system clipboard to the display area. If text at the destination is selected, it is replaced; otherwise the clipboard contents are inserted at the current text cursor position.

Select All

Select All selects the entire contents of the display text area.

11.3 File

Exit

Exit stops tracing, close the current log file (if any), removes the trace window and restores any previous MQe trace handler.

Log Trace Events

Log Trace Events causes the trace output to be written to a log file, either in plain text format (*.txt*) or as an MQe binary trace file *(.trc)*. The events logged are controlled by the settings chosen in the *Action* menu. The logging continues independently of stopping and starting trace display (View→Display), or editing actions in the display area.

After *Log Trace Events* have been used to stop the logging of trace events, logging can be subsequently re-started to a new file, again with a choice of format.

Plain text files, in addition to capturing trace, also capture when display was stopped and started. A plain text file is closed immediately logging is stopped; there is no maximum size.

Binary log files do not capture when the display was stopped or started. Each binary trace log file is limited to a maximum size of 2Mbytes; when this limit is reached a new trace file is automatically created. The first file uses the given name suffixed with the string “000”, each subsequent file increases the suffix by one, i.e. the second file has the suffix “001”. Binary log files are normally closed when MQe_Explorer itself is closed, unless the file is deemed to be full.

Save Display As

Save Display As saves the contents of the display area as either a plain text file (.*txt*) or a rich text file (.*rft*).

11.4 Format

Font

Font formats the currently selected text in the display area. Font, style, size, color and effects may be set.

11.5 View

Display Trace

Display Trace toggles the display of trace information; a message record the time that display was stopped or started. Logging is unaffected.

12 Advanced topics

12.1 Invocation

Invocation options

MQe_Explorer can be invoked in three different ways:

1. *By executing MQe_ExplorerV2.exe*

   Only MQe_Explorer and the WebSphere MQ client classes are contained within the executable. The *classpath* is required to identify the source of the MQe classes and any additional classes that may be specified.

2. *By calling the static main method of the MQe_Explorer.class*

   All classes are located via the *classpath*. The main method is declared as:

   ```
   public static void main(String args[ ])
   ```

   This technique allows MQe_Explorer to run inside an already created JVM. If an existing queue manager is present then MQe_Explorer will use it. In this mode the queue manager name shown in the status bar is enclosed in [ ] brackets to indicate that MQe_Explorer is running as a slave application. As a slave, certain operations are grayed out, for example:

   File→Open, File→New, File→Close

   When invoking MQe_Explorer in this way, the calling application should create a new thread to be used for the MQe_Explorer invocation.

Invocation parameters

Irrespective of the way in which MQe_Explorer is invoked, it may take up to three run time parameters:

1. *Option file specification: /o <file path> (or: -o <file path>)*

   Where <*file path*> is the option file path specification. The file type must be *.opt* – if this extension is not present in the string it will be added. If a relative path is used (for example: .\) this will be taken to be relative to the location of the current directory (typically the location of the *.exe* file). If the named file cannot be found, a default file will be created using that file path specification.

2. *Initialization file specification: /i <file path> (or: -i <file path>)*

   Where <*file path*> is the initialization file path specification. The file type may be any type but *.ini* would be conventional. If a relative path is used (for example: .\) this will be taken to be relative to the location of the current directory (typically the location of the *.exe* file).

3. *Message suppression specification: /s <suppression keyword> (or: -s <suppression keyword>)*

   Where valid values of <*suppression keyword*> *are:*

   suppressIO – suppress all background loading messages that report I/O errors.

Thus the following is acceptable:

```
MQe_ExplorerV2.exe    /i  .\MyMQe\FirstQM.ini   /o  C:\Options\MQe_Explorer.opt
```

Additional control of MQe_Explorer operation is available; these interfaces are documented in MQe_Explorer on page 172.

12.2 Option files

All settings that have been created through the *Tools→Options* menu are stored in an option file identifiable by its *.opt* extension. By default this file is named *MQe_Explorer.opt* and by default is expected to be located in the same directory as the executable *MQe_ExplorerV2.exe*. If the file cannot be found then a new one will be created and initialized with default values.

If multiple instances of MQe_Explorer are run from a single *.exe* then, by default, they will share a single option file. This is permitted; note however that the file is only loaded when an MQe_Explorer instance is initialized and it is saved whenever the *Apply* button is clicked after one (or more) options have been changed. The stored file will therefore reflect the state of the last MQe_Explorer to change an option, but any such changes will not affect other already running instances.

If individual instances are to use their own option file, then the file path must be passed as a run time parameter, see *Invocation* on page 111 for more details. The option file in use is listed in the *System Information*, accessible from the *Help→About MQe_Explorer* menu item.

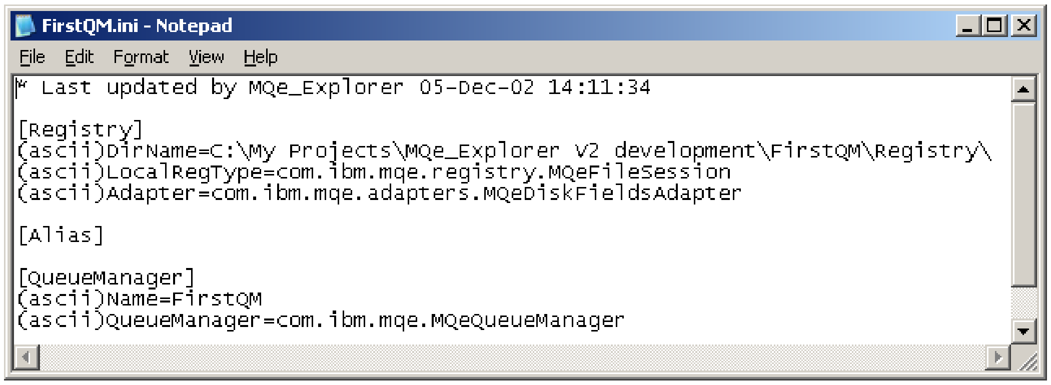
12.3 Initialization files

MQe_Explorer supports the use of standard MQe initialization files (typically identified by a *.ini* extension) to establish the queue manager environment and to identify the queue manager to be loaded. The following sections are supported, other sections are ignored:

[Alias]	*defines alias names for MQe classes.*
[PreLoad]	*identifies classes to be pre-loaded when the queue manager is activated.*
[QueueManager]	*identifies the queue manager by name.*
[Registry]	*locates the registry and sets parameters (including security).*

The *[QueueManager]* and *[Registry]* sections must be present; others are optional. Note that the standard MQe alias definitions do not need to be supplied in the [Alias] section; only additional user-defined aliases need to be included.

The *FirstQM.ini* file, created in *Configuring a first queue manager* on page 7 has the minimal content:



```
* Last updated by MQe_Explorer 05-Dec-02 14:11:34

[Registry]
(ascii)DirName=C:\My Projects\MQe_Explorer V2 development\FirstQM\Registry\
(ascii)LocalRegType=com.ibm.mqe.registry.MQeFileSession
(ascii)Adapter=com.ibm.mqe.adapters.MQeDiskFieldsAdapter

[Alias]

[QueueManager]
(ascii)Name=FirstQM
(ascii)QueueManager=com.ibm.mqe.MQeQueueManager
```

Figure 12-1: The *FirstQM* initialization file

Initialization files can be edited with the Windows *Notepad* editor. However for queue managers on Windows platforms, it should not be necessary to edit such files – MQe_Explorer provides higher level functions to manage their content.

Within an initialization file, paths can be specified for certain parameters, for example, the location of the directory containing the queue manager registry. If a relative path is used (.\) it will be taken as being relative to the location of the *.ini* file itself[26].

Passwords can be placed in MQe initialization files. To avoid the adverse security implications of this technique, MQe_Explorer implements a special value for such passwords. If the value is set to '*(prompt)*' then MQe_Explorer will prompt for the actual password during queue manager loading. Passwords must meet these criteria:

- *Be at least 6 characters in length.*
- *Contain at least 4 unique characters.*
- *Not match various excluded values.*

The initialization file in use is listed in the *System Information*, accessible from the *Help→About MQe_Explorer* menu item.

12.4 Managing remote queue managers

Requirements

Managing remote queue managers from MQe_Explorer requires set-up on both the local queue manager (i.e. the one hosting MQe_Explorer) and on each remote queue manager. Typically the host queue manager is configured as a server; the remote queue managers are configured as either servers or as clients. Assuming that the default names of queues are being used, the basic requirements are:

- The host queue manager:
    - It requires connectivity (directly or indirectly) to the remote queue managers, i.e. for each remote queue manager there must one of the following enabled:

[26] Note that this differs from the default MQe behavior for initialization file processing; note also that it differs from the way in which relative paths in run time parameters are interpreted. MQe_Explorer will issue a warning message if relative paths are in use.

        - A *connection definition* for that remote queue manager (either *direct* or *indirect*) – where the remote queue manager is a server.
        - The remote queue manager must feature as a *destination* in a store queue definition – where the remote queue manager is a client.

      This provision provides basic connectivity from the host queue manager to the remote queue manager.

    - If online administration is required (the normal case), MQe will automatically create a *sync. proxy queue* when required on the host queue manager, to the *admin queue* of the remote queue manager. If offline administration is required then an *async. proxy queue* is needed instead – this must be entered manually. By default, the name of the remote admin queue for online access is assumed to be 'AdminQ'; likewise the remote admin queue for offline access is assumed to be addressable as 'OfflineAdminQ'. Proxy queues with these names allow admin messages to be addressed to the admin queue of the remote queue manager. Both online and offline admin can be enabled to the same remote queue manager if required (e.g. by having the remote admin queue called 'AdminQ' with an alias of 'OfflineAdminQ' and by having matching proxy queues defined on the host).
    - It must be able to receive messages from each remote queue manager. This requirement is usally satisfied by the host queue manager having one (or more) *comms. listeners* that can receive incoming connection requests from the remote queue manager (note that these comms. listeners are typically shared by the remote queue managers being managed; a comms. listener is not required per remote queue manager)

- Each remote queue manager:
    - It requires connectivity (directly or indirectly) to the host queue manager, i.e. there must be:
        - A *connection definition* for the host queue manager (either *direct* or *indirect*).

      This provision provides connectivity from the remote queue manager to the host queue manager.

    - If online administration is required, MQe will automatically create a *sync. proxy queue* when required on the remote queue manager, to the *admin reply queue* of the host queue manager, called 'AdminReplyQ' by default. If offline administration is required then an *async. proxy queue* is needed to that same queue instead – this must be entered manually. Additionally for asynchronous admin only, and assuming default names, an *alias* of 'OfflineAdminQ' needs to be given to the remote admin queue.

      The proxy queues allow messages to be addressed to the admin queue of the remote queue manager; the alias on the remote queue allows offline admin requests to be received.

    - It must be able to receive messages from the remote queue manager. This requires one of the following:

- A *comms. listeners* that can receive incoming connection requests from the host queue manager – where the remote queue manager is a server.
- A home server queue that pulls messages from the host store queue – where the remote queue manager is a client.

When setting up the host queue manager and the remote queue manager it is essential that the corresponding definitions match. For example, a connection definition always matches a comms. listener definition, with the connection definition describing the request, and the comms. listener the receipt of that same request. In this case the port, protocol adapter (not just the protocol), and adapter options must match exactly. Likewise, a sync. or an async. proxy queue matches an application queue definition; the details here are not required to be identical but the names must match (or at least the name of one must correspond to an alias of the other).

Practical aspects

The easiest way to configure queue managers to either host MQe_Explorer or to be managed by it, is to use MQe_Explorer itself to perform the configuration locally. This means opening (or even creating) each queue manager with MQe_Explorer and then creating the various objects locally on each queue manager. However, this is not always practical and anyway MQe_Explorer is limited to opening/creating only those queue managers that run on Windows platforms. Despite that, it is simple and sensible to use MQe_Explorer to create and configure its own host, and it may be also used to create/configure some of the other queue managers for later administration.

Other remote queue managers can be configured for remote administration in one of two ways:

- Use the various utilities that ship with MQe to configure the remote queue manager sufficiently so that it can communicate with MQe_Explorer (using the requirements above). Then use MQe_Explorer remotely for further configuration and management.

  This method is mandatory for *clients* and an option for *servers*.
- Start the remote queue manager and then get MQe_Explorer to configure it remotely itself; this is only possible if the remote queue manager is a *server*. The following steps should be followed:
  - Ensure that the remote queue manager is active and that its comms. listener is running.
  - Create a connection definition to the remote queue manager using MQe_Explorer on the host. The remote queue manager will now appear in the tree list pane, with a red icon showing that it is not available.
  - Right click on the red icon and display its connection folder – there will be a delay whilst MQe_Explorer makes a further attempt to connect to it and get a response.
  - Right click on the connections folder and bring up the new connection dialog. Uncheck the work offline check box in the bottom left. Fill in the details needed for a connection back from the remote queue manager to the host queue manager. Click *Apply*; there will be a timeout delay and then MQe_Explorer will report that the response has been delayed.
  - At this point the remote queue manager has a connection back to MQe_Explorer. Click on its red icon again and this time MQe_Explorer will be able to exchange messages; after a pause the icon will go black. The remote queue manager is now configured for admin.

12.5 Admin and reply queues

MQe_Explorer sends admin messages to remote queue managers; it receives replies back on the admin reply queue of its hosting queue manager. The admin queue to which messages are addressed depends upon the settings in the *Tools→Options→Advanced-1* tab, *Admin request queues used* group. By default, the names are:

Online working*: AdminQ*

Offline working*: OfflineAdminQ*

These names cannot be varied on an individual target queue manager basis. If such unique names are required then an appropriate queue aliases should be added, corresponding to the name(s) that MQe_Explorer will use.

The admin reply queue of the hosting queue manager used to receive admin replies is called *AdminReplyQ* and this is not configurable; moreover, in order to avoid problems the properties of this queue cannot be modified locally through MQe_Explorer.

If MQe_Explorer is used to create (or configure) a queue manager it will always create the *AdminReplyQ* and an admin queue called *AdminQ*. If MQe_Explorer is run on a pre-configured queue manager that does not have these queues, then it will add them. It does not explicitly create *OfflineAdminQ* because MQe_Explorer has no use for it on the local queue manager.

12.6 Operational aspects

MQe_Explorer caches MQe object details in order to improve responsiveness. This technique allows a static (but refreshable) view of elements of the network to be explored with important usability benefits. Since messages are constantly arriving at, and being deleted from queues, a truly dynamic presentation of the network object status would inhibit effective user interaction.

The object cache is loaded asynchronously and its operation may be influenced through the *Tools→Options→Advanced-2* settings.

The status of the asynchronous loading is displayed in the status bar (in the third panel from the left at the bottom of the main window). The number displayed gives the number of asynchronous outstanding work items remaining to be processed; this number will be zero under normal circumstances. However demanding queries, such as a message refresh on a deep queue, may temporarily cause it to become non-zero; in these cases, some data may be temporarily shown as unavailable.

Information on remote queue managers that have failed to respond can be obtained by double clicking the status bar – the following panel is displayed:

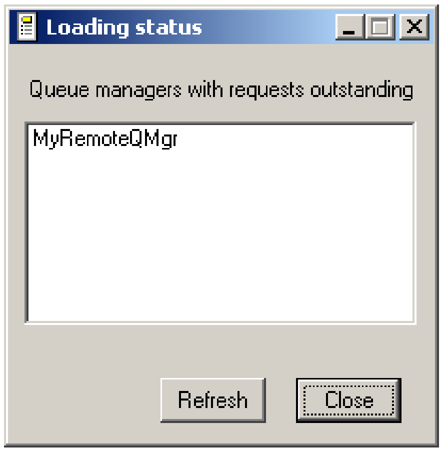


Figure 12-2: Queue manager responsiveness

12.7 MQe_Explorer messages

MQe_Explorer uniquely identifies the messages it uses to distinguish them from any others in the network, and to avoid confusion on shared queues. In addition to the standard features provided by MQe, MQe_Explorer uses the following:

- All messages except tests and file transfers, include a field with a coded name[27] **MQe_Explorer*.
- All messages except tests and file transfers, include a field with a coded name **Msg_ CorrelID*, used to stamp requests belonging to the same session.
- All messages except tests, pings and file transfers, include a field with a coded name **Mode*, indicating whether *online* or *offline* admin is required.
- All messages except tests, pings and file transfers, include a field with a coded name **Proxy_Hash*, *used to* identify the cached object to receive the response.
- All messages except tests, pings and file transfers, include a field with a coded name **Refresh_Objects,* used to identify the child cached objects to be refreshed on receipt.
- All messages except tests, pings and file transfers, include a field with a coded name **Unique_Id,* used to correlate responses with requests.
- Pings include a long field with a coded name **Ping_Time*, used to pass the ping request time.
- File transfers include the following fields: *Code*, *Data*, *Description*, *Digest*, *Id*, *Path*, *Segment* and *Segments*.
- Messages that are copied (as a result of *Tools→Demo Mode* being checked) have the **Msg_CorrelID* field set to a zero value and the UID re-set. Additionally they have the following fields added:

[27] This is a reserved UNICODE string value that is short, but unprintable. MQe_Explorer translates it when displaying field names – all such translated names begin with the * character.

- A UNICODE field with a coded name that translates to **Copy_Info* and a value detailing when the copy was made
- A byte array field with a coded name that translates to **Original_CID* and a value of the original **Msg_CorrelID* field
- A fields field with a coded name that translates to **Original_UID* and a value of the original message UID.

13 Sample scripts

The following scripts[28] are supplied to introduce the features of WebSphere MQ Everyplace and to gain familiarity with the MQe_Explorer:

1. *Concepts and objects*
    - An overview of WebSphere MQ Everyplace messaging including the basic objects and their properties.
    - A general introduction to the use of MQe_Explorer.
2. *Basic messaging*
    - Direct and indirect connections between queue managers.
    - Synchronous and asynchronous messaging.
    - Use of the home server, store and forward queues.
3. *Advanced messaging*
    - Use of the home server, store and forward queues including intermediate network queuing.
4. *Gateway configuration and usage*
    - Use of MQe_Explorer to configure a gateway queue manager and how an MQe gateway can be used to exchange messages with WebSphere MQ.

Note: In the following scripts blue text is used to highlight actions to be taken.

[28] The scripts (arbitrarily) use MQe messages flowing over a raw TCP/IP protocol to *local host*; with minor configuration changes they can be modified to exploit HTTP. When using a physical network to connect machines with different IP addresses, ensure that the influence of any SOCKS stacks and/or HTTP proxies is fully taken into account. Whilst all these configurations can be supported, the presence of firewalls, SOCKS and HTTP proxies can require explicit MQe configuration changes. The handling of HTTP proxies by MQe_Explorer through the Tools→Options menu is described in the section *Advanced-3* on page 73.

13.1 Concepts and objects

Overview

This script introduces the key components of messaging and the object structure of WebSphere MQ Everyplace (MQe) and is also useful as an introduction to the MQe_Explorer. It alludes to the many advanced facilities of MQe but makes no attempt to demonstrate all of the features. It is written such that it can be run on a standalone machine by using multiple queue managers listening on different ports of the local host.

Script

Introduction

It is assumed that the computer has no queue managers defined – and therefore no queues, no messages etc. The machine is in the state expected following an initial install of the MQe product, followed by MQe_Explorer. MQe_Explorer is used to create everything needed. Firstly a queue manager must be created[29].

Creating a queue manager

Double click the MQe_Explorer desktop icon . The MQe_Explorer window appears.

This window looks like the *Windows File Explorer* with menus, a toolbar and left and right hand panes. To create new queue manager:

Click the 🗋 on the toolbar (the *New folder* button).

The new queue manager creation dialog is displayed. As a minimum, the new queue manager name needs to be entered.

Enter the new queue manager name "FirstQM". Check that the path is *C:\Program Files\MQe\Java\MQe_Explorer* – or edit the path to this value[30]. Leave the other fields with their default values.

A message appears giving the location of the new queue manager's initialization file. This is the file that must be opened if you subsequently wish to restart this queue manager.

Click OK on the initialization file message.

Now many things have happened. A configured and running queue manager has been created. A message appears in the status area at the bottom left "1 queue manager(s)" – meaning that MQe_Explorer only knows about one queue manager. On the same status bar at the far right is a panel containing the string "FirstQM" – this gives the name of the queue manager used by this instance of MQe_Explorer for its admin operations.

[29] If you followed *Configuring a first queue manager* on page 7 you already have the first queue manager *FirstQM* defined.
[30] This is the recommended path name but you may change it – the significance of this parameter is that it identifies where your queue manager and associated files will be stored.

Hover the cursor over the various status bar elements (starting at the left and moving to the right).

The leftmost panel is already understood, the next panel shows the nature of the local environment, the next the number of objects selected in a multi-select operation, the next the outstanding work items "0" i.e. no work items are outstanding. MQe_Explorer asynchronously requests information on various objects (such as queue managers, queues, connections etc.) and caches the results; the zero indicates that all the background requests have been processed. The last panel shows the name of the local queue manager.

In the left main panel (the tree pane) is displayed the root of a tree – a map of the MQe network. So far there is just a root represented by the icon (called *MQe root*) and a ⊞ sign alongside that indicates there is more information to be seen.

In the right hand panel (the list view pane) a row shows details of the *FirstQM* queue manager just created.

Expand the application window by pulling the bottom left hand corner down and to the right.

Now the full description of the queue manager is displayed – confirming that MQe_Explorer created it a few seconds ago. Other properties are apparent.

Move the scroll bar in the list view pane to the right.

More properties are visible; the queue manager has a *Default queue adapter* (which is truncated). The full adapter class name can be seen if the column width is increased.

Drag the right hand side of the *Def. queue adapter* column header to the right. The column width increases. Then move the scroll bar so that the next set of columns is clearly visible (number of queues, connections, etc).

The new queue manager has four queues – these are queues that MQe_Explorer has created by default because they are generally useful – there is no MQe requirement that such queues must be present. No connections are defined (unsurprisingly) – a connection definition describes how one queue manager communicates with another. Similarly no communications listeners are defined – a communications listener listens to incoming connection requests from other queue managers. It can be seen that the queue manager is gateway capable, i.e. it has access to the resources necessary for gateway configuration and so could be enabled to talk to a WebSphere MQ queue manager network. Other properties can be inspected.

Bring up the *Windows Explorer* and navigate to the directory *C:\Program Files\MQe\Java\MQe_Explorer* and display its contents.

A new folder has been created called *FirstQM* – holding the new queue manager.

In the *Windows Explorer* open the folder *C:\Program Files\MQe\Java\MQe_Explorer\FirstQM* in the left hand pane and expand all the sub-directories so that the full folder structure is visible.

A collection of files and folders has been created – divisible into queue folders and registry folders. The registry holds the detailed configuration information – the queue folders hold the messages. The registry is a sophisticated component of MQe that can be configured to provide secure storage of MQe information.

Leave the *Windows Explorer* and return to the *MQe_Explorer* window.

In the list view pane there are some queue manager properties that are blank (that is, not set) such as *QMgr aliases*. Others are set – for example *Channel timeout*.

Properties of a simple queue manager

The details of the *FirstQM* queue manager can be investigated.

In the tree pane click on the ⊞ sign next to *MQe root*.

A sub node appears for the queue manager *FirstQM*.

In the tree pane click on the *FirstQM* node.

A ⊞ sign appears alongside indicating there are more elements below in the tree. The detail in the right hand pane gives more information, showing the detail associated with the selected element in the tree. In this case four folders are shown representing child objects of the queue manager, namely:

1. The MQe bridges object – shown as stopped, with no child bridge objects.
2. The collection of communications listeners, shown as having no children configured.
3. The collection of connections, again shown as having no children configured.
4. The collection of local queues, showing that four exist.

Double click on *Local queues* in the list view pane.

The left hand pane shows the expanded tree with the *Local queues* node selected. The right hand pane shows that there are four local queues (*AdminQ*, *AdminReplyQ, DeadLetterQ and SYTEM.DEFAULT.LOCAL.QUEUE*[31]). Three queues have standard application queue icons like this: ▤ . The icon for the *AdminQ* is: ▤ , colored to indicate that this is a rather special queue because of its admin implications and to ensure that it is not unintentionally modified. Other types of queues will be seen later with different associated icons. The *AdminQ* is the one to which admin messages must be sent. MQe_Explorer will send to this queue when it wants to query the *FirstQM* queue manager or to change the queue manager configuration. *AdminReplyQ* is used to receive the results of admin requests; *DeadLetterQ* is a queue typically used to hold messages that cannot be otherwise delivered[32].

Scroll the list view pane to see the properties of queues.

Queues have many properties – a *Queue description*, *Depth*, *Queue type* etc. These properties are shown as snapshots, taken at the time when MQe_Explorer first accessed them – to get a current view the display must be refreshed (either implicitly or explicitly) – this will be done later after changes have been made.

Many of the properties are not set, e.g. *Compressor*, and it is convenient to be able to inhibit the display of such empty columns.

Click on the button (*Hide Empty columns*) on the toolbar and then scroll the right hand pane.

Now only columns that contain useful information are displayed. Columns can also be re-ordered.

---

[31] This strangely named queue is provided for compatibility with MQSeries queue managers. By default, all queue managers (i.e. both MQe and MQSeries) will have such a queue available. If not required it can be deleted, alternatively it need not be created by default (see *Tools→Options→Advanced-1*).

[32] The identity of the admin request queues used by MQe_Explorer can be changed (see *Tools→Options→Advanced-1*).

Drag the column heading of *Queue type* left into the middle of *Queue description*.

The columns have been re-ordered. Column re-ordering done in this way is only a temporary operation lasting until the list view is changed; the same is true for column width changes done by dragging column boundaries. Permanent changes of order, width and visibility are available through the *Tools→Options* menu.

Click once on the *Queue type* column header, and then click again.

Clicking on a column header sorts the data in that column; clicking again sorts in the reverse order.

Repeat the operation on the *Queue name* column.

In the right hand pane, right click on the *AdminReplyQ* row. In the sub-menu select the *Properties* item.

The property window for the *AdminReplyQ* appears. The title indicates that the queue belonging to the *FirstQM* queue manager and can be modified. The various tabs display its properties, grouped logically into *General*, *Properties*, *Storage*, *Security* and *Aliases*. On the *General* tab the date and a time appear at the bottom right; this time stamp shows when the information was obtained from the queue object – all such property windows carry this refresh information.

Click the various tabs, ending up back on the *General* tab.

Many properties are grayed out because that they are not relevant to a queue of this type – thus, for example, a local queue does not have a destination qMgr. property (such a property would be valid for a sync. or async. proxy queue). Other properties are read-only and may not be changed – in this case, for example, no properties may be changed on the *AdminReplyQ* queue whilst it is being used by MQe_Explorer to receive admin replies – normally local queues are not so restricted.

Other queues have different properties, for example the *DeadLetterQ*.

Click the *Close* button. In the right hand pane right click on the *DeadLetterQ* row. In the sub-menu select the *Properties* item. Explore all the tabs and finish back on the *General* tab.

This shows a typical application queue – parameters such as *Maximum message size* and *Maximum depth* can be changed (to enter a number in these properties the *No limit* box must be unchecked). For example, to change the *Description*:

Click on the *Description* field and edit the value to "Default Dead Letter Queue". Click the *Apply* button.

After a brief pause the cursor changes back to the default style to indicate that the operation is complete. Looking in the list view pane of the main MQe_Explorer window, the description of the *DeadLetterQ* has also changed there – a refresh of the display has occurred. Such refreshes are implicit when an object property is changed.

Many objects in MQe can have alias (alternative) names in addition to their base name – again, the property pages are used to create aliases, for example, for the *DeadLetterQ*.

Click on the *Aliases* tab, type the string "DLQ" in the typing area and click the *Add* button. Then click the *Apply* button. In the list view pane scroll so that the *Aliases* column is visible.

The alias name has been added to the queue properties.

There is alternative format for information display in the right hand pane of the main window that makes alias names more conspicuous. Instead of a row per object, it is possible to have a row per name.

In the tool bar click the button (*Use extra rows*). Click the *Close* button on queue properties dialog to remove the window.

An extra row now appears in the list view pane – with a variant of the queue icon . This new icon indicates a shortcut to that queue, that is, an alias. As before, the list can be sorted by name.

Click the *Name* column heading a few times to view sorting by name.

Queues normally contain messages. Drilling down a level enables messages to be seen.

Double click the *AdminReplyQ* node in the list view pane.

The list view pane is now empty and the status bar indicates that there are no messages. This is not too surprising since no messages have been explicitly created (although MQe_Explorer has been using messages itself). However an option exists for displaying the messages being used by MQe_Explorer.

Click on *Tools* in the menu bar, and then click on *Demo Mode*.

The effect of demo mode is that MQe_Explorer will make a copy of all admin messages sent and received and then put these copies in the *AdminReplyQ*. In order to see the effect of this we must cause admin messages to be issued (and then copied); manually refreshing a queue will have this effect. However for other reasons, here we want to do something that changes a queue property.

In the tree pane right click on the *DeadLetterQ* node. In the sub-menu select *Properties* and on the second tab increase the *Priority* by one. Click *Apply* then *Close*. Then in the tree view pane right click on the *AdminReplyQ* and select *Refresh*.

Six messages appear in the right hand pane. The first is the priority change request, typically the third an "Inquire all" on the *DeadLetterQ*, and the fifth an "Inquire all" on the *AdminReplyQ*. Messages two, four and six are typically the respective replies, though the order may vary depending upon the system characteristics.

Scroll the right hand pane to see the properties.

There are many properties, such as the *Origin queue manager* (with a value "FirstQM*"* as we might expect). There is a *Message* creation *date* and *Message* creation *time*, *Message object class*, etc. Although the display claims the messages have between 20 and 22 fields there are that many displayed – the *Hide Empty columns* option is causing some columns to be suppressed.

Move the cursor into the tree pane.

The hover help in the tree pane shows the class of the queue; a similar display occurs with messages. When a message (or fields object within a message) is selected in the tree pane, the hover help shows the class of the associated object.

In the toolbar re-click on the *Hide Empty columns* button (so that it is no longer depressed).

More columns appear – mainly to do with various security settings for a message. What is being shown at this level are various properties of the message itself (such as *Message creation date*, *Message object class* etc.) and a number of common fields that messages might be expected to contain (such as *Message priority*, *Correl Id* etc). Of course messages are not obliged to have any fields at all, apart from the unique identifier that MQe adds. That unique identifier is actually displayed as the two columns *Message origin queue manager* and *Message creation time*. In order to make the time display more useful a third column is also generated called *Message creation date* – but this is only a reformatting of the value in the *Message creation time*.

The column *Message name* contains an icon and a number. That number is not present in the message – it is a number assigned by MQe_Explorer itself to the message to make it memorable. Strictly it relates to the proxy object MQe_Explorer holds that relates to the message – from a snapshot of the queue that has been taken. In an active system, where multiple applications are running, the original message may already have been removed from the queue – however if the display kept up with reality management would be impossible. MQe_Explorer therefore keeps copies of messages so that they can be displayed or manipulated. In some situations occasionally a message will be displayed that has only has the properties Message *name*, *Message origin queue manager*, *Message creation time* and *Message creation date*, with all other information shown as unavailable. The explanation for this is that such a message used to exist on the queue but in between MQe_Explorer getting hold of its UID and requesting the rest of the information, the message was taken removed from the queue (or it expired).

There is another way of looking at messages on the *AdminReplyQ* that only applies to this particular queue. The view is part of the offline administration support. When objects are offline at the time of administration the results will come back at some future time – the offline administration support provides a means to track these actions. Although in this case we are not dealing with offline administration, the provision allows for the inspection of any admin messages on the *AdminReplyQ* (excepting those that are only property enquiries or their replies). Select the menu item *View→Offline Admin*.

A new window appears with no messages present. Maximise to full size. On its toolbar click the button; "other" (i.e. non offline) admin messages present on the *AdminReplyQ* will then be added to the display.

A number of messages are shown; the actual number depending upon several factors. We saw that there were six messages on the AdminReplyQ – however all of these messages have an expiry time set, so they will disappear after that time has elapsed. By default MQe_Explorer uses an online admin message expiry time of 5 minutes, so if you waited too long after refreshing the AdminReplyQ some, or all of those messages may have expired. You may want to repeat some of the earlier actions to get a new set of messages. The second factor is that the offline admin support in MQe_Explorer takes an independent copy of the queue information, so that it is unaffected by any actions that you might perform through property pages or the main MQe_Explorer window. You will see at least this inquire on the queue and the corresponding reply.

Assuming that you have access to the previous six messages as well, you will now see them displayed in an alternative format more suited to the particular needs of administration. The "Update" request on the *DeadLetterQ* is present, as is the reply indicating. Double clicking the request shows more details of the request – the priority change can be seen; double clicking the reply shows details of any errors that occurred – in this case no errors are present. The column *Request time* shows the time the original offline admin request was made – in these cases there is no actual offline request time to display and so the message creation time is displayed instead – hence the times shown are approximate and the "~" suffix is used to indicate this. However the times correlate exactly with the *Message creation time* field in the corresponding message view that can be seen in the main display window.

Double click the two rows in turn to see the detailed information – then close the detailed windows. Then close the offline display window and return to the main MQe_Explorer window.

One powerful feature of the MQe_Explorer is its ability to drill into a message to show the structure.

Double click the first of the messages in the list view pane.

Now the first level of message content is displayed; each row corresponds to a field in the message.

Scroll horizontally (and vertically if necessary) to see all the data presented.

There is a lot of information presented. Firstly the *Field names* – some are prefixed with an * such as **Msg_CorrelI*. This notation indicates that the actual name in the message is an (unprintable) encoded value that corresponds to an MQe or MQe_Explorer defined constant; MQe_Explorer displays the constant's name prefixed with an *. Encoded names are typically used to save space in messages and standard codes exist for all common field names.

Many field names can be (optionally) translated to more meaningful strings. For the names that occur in admin messages MQe_Explorer is able to provide a more meaningful interpretation – click the button on the toolbar. The name *admact* now appears as 'Action', in quotes to indicate that a translation has taken place; likewise **Msg_CorrelId* becomes 'Correl. id'.

The message displayed is a copy of an admin message – an "Update" request used to change the properties of the *DeadLetterQ*. The value of the *Action* field tells admin that this is an update (as opposed to a delete, inquire, etc.), *Max attempts* says how many times the operation should be attempted (here just once), *Expire time* the message expiry time in millisecs, *Local queue manager* is the name of the queue manager to which the request is directed, and so on. Many fields are added by MQe_Explorer for its own purposes; examples of these are *Admin mode*, *Proxy hash* and *Unique id*.

Extend the application window vertically so that no scrolling is required to see all the rows.

One row is of particular interest – *Original correl. id*. MQe_Explorer created the copy from the original message but changed the value of this field because it would have caused application problems if the original value had been preserved; however the value was saved in a new field so that it was not lost. The *Copy info* field was added to record the details of the copy operation.

If a reply message is examined it will be seen that a similar action is taken with the UID. Received messages have their UIDs set and MQe insists that they be unique across the network. MQe_Explorer copies the value into a new field to preserve the information.

Click the button from the left on the toolbar (*Up one level*) and then go back to the message details by double clicking the message.

The *Up one level* button provides a useful short cut to the parent display details.

Click the *Data type* column heading on the message display.

Various data types are visible: *ASCII*, *Boolean*, *Byte*, *Byte array*, *Fields, Int, Long* and *UNICODE* fields. They all have a value – except *Fields* fields (which will be investigated later), a length and other properties. Some are arrays – such as the *Correl. Id* – for this information the array of bytes data type is chosen to maintain compatibility with other WebSphere MQ products. *Correl. Id* is a static array, that is an array that has fixed bounds; other arrays can be dynamic with flexible bounds. Other properties exist – the more interesting of which are the security attributes (such as *Compressor*, *Authenticator* etc. that can be applied to a *Fields* field and used to protect component elements of a message).

A basic message is itself a *Fields* object. The *Fields* field within a message represents a nested element. Double clicking can expand it.

Double click the 'Parameters' row.

This fields object holds the detailed parameters of the admin message. Three fields are visible: the *Name* of the queue, its *Queue qMgr* name, and the value of the *Queue message priority* that was set.

At this point it is appropriate to exit demo mode - otherwise too many messages will be collected.

Click the *Tools* menu and disable *Demo Mode*.

Creating a new queue

A new queue can easily be created.

Right hand click the *Local queues* node in the tree view. Select the *New queue* menu item. A new queue definition window appears. Rearrange the various MQe_Explorer windows on the screen so that they are all fully visible.

MQe_Explorer is not modal. Although we are in the in the middle of creating a new queue – other objects can still be explored, as previously.

Click around the original window to show it is still active. Return to the new window.

Queues have many properties – and these depend upon the nature of the queue. Initially we will create an application queue.

Type the string "MyQ" into the *Name* field and "A new queue" into the *Description*. Explore the drop-downs on all the tabs. Finish with the *General* tab showing.

Note that *Time interval* and many other properties are not available – not all properties are relevant to application queues. Simple default properties can be changed as follows.

On the *Storage* tab uncheck the *No limit* against *Max depth* and set the *Max depth* value to 10. On the *Properties* tab use the spin button to increase the *Priority* property to 6. Click the *Create* button.

Note that the window stays in case another queue is to be created. In the main MQe_Explorer window the new queue is evident – both in the left and the right hand panes. In the right pane it is apparent that it has the custom properties set. A test message can be sent to the new queue to confirm that it is operational.

Right click the *MyQ* row in the right hand pane and select the *New message* menu item. A new window appears.

Everything is already set up for a default message - clicking *Send* would send the message. However there are useful options to explore.

Explore the drop-downs etc.

A custom message could be sent, with a user-selected field and /or encoding, to a different queue manager target (if there were any), and to different queues etc. Note that queue manager and queue names can be entered – in addition to those available from the drop-down list (this allows for the use of alias names, as well as for dynamic discovery of queue details in synchronous messaging). In our case however, a default message can be sent.

Click the *Send* button. A dialog appears with information on the message sent – click *OK*.

To see if the message arrived:

Right click the *MyQ* row in the list view pane and select *Refresh*.

The *Depth* property changes to 1 – reflecting the new message.

Double click the MyQ row to see the message.

The message has four fields and expected values for the other properties. Note that test messages have a message expiry value set – this value is set to be the same as that for online admin messages (see *Tools→Options→Advanced-2*). A consequence of this is that if there are long delays between executing steps in this script, test messages may disappear of their own accord.

Double click the message row. The contents appear – explore them.

The contents of the three fields are as expected. More messages can be sent.

Click the *Send* button on the *Send test message* dialog. Then click *OK* on the resulting message box. Repeat the set of operations once more.

In the main window, tree view pane, nothing new is visible.

On the *MyQ* node in the tree pane, right click, and then select the *Refresh* sub-menu item. Both panes are updated – and three messages exist.

Messages can be deleted, copied and moved.

Select the first message in the right hand pane and then hit the delete key. Click *Yes* on the confirm dialog – the message is deleted. Select the message 2 and then with shift held down, select message 3. Both messages are selected. Do a drag copy operation (Control key down with a mouse select) and drag the messages from the right hand pane into the tree pane – and drop on the *DeadLetterQ*. Whilst over other nodes the drop will have been disabled – but is permitted over valid drop targets such as the *DeadLetterQ*. The status bar contains details of the operation. Click the *DeadLetterQ* in the tree view pane and *Refresh* the view. The copies of the messages are visible in the list view pane.

These copied messages are not identical to the original messages because they do not have the same UID. Messages can be moved in a similar way, and can also be cut and pasted via the clipboard.

Using the menu item *Window→Close All*, close all windows except the main MQe_Explorer window.

Communicating with other queue managers

It is more interesting to experiment when there are multiple queue managers. A second queue manager called *SecondQM* will be created (on the same machine) and the two queue managers will then communicate with each other.

Firstly *FirstQM* needs information on *SecondQM*; this is provided by defining a *connection* (sometimes known as a remote queue manager definition). The connection allows *FirstQM* to make a *connection request* to *SecondQM*. In order to respond to connection requests *SecondQM* must have a *comms. listener* configured. Since later in this script, *SecondQM* is going to make connection requests to *FirstQM*, it follows that *FirstQM* will also need a comms. listener configured.

To create the connection definition, in the tree pane right click the *Connections* node. Select the *New Connection* menu item. A window appears. Explore the tabs finishing back on *General*.

The name of this connection is *SecondQM* since that is the name of the remote queue manager. The *Description* can be any ASCII string. For all other properties on this tab the defaults will be used. *SecondQM* will be defined as a second queue manager running on the same machine as *FirstQM*.

Enter "SecondQM" into the *Name* field and "Second queue manager" into the *Description*. Leave the *Type* field with the default value of "Direct". Move to the *Detail* tab.

The *Detail* tab describes the nature of the connection parameters. Many defaults can be taken, but the basic protocol and addressing information must be specified.

In the *IP address* field enter the address of *SecondQM* (i.e. either the numeric address 127.0.0.1 or the name "localhost" can be used). In the *Port* field enter the port number 8083 which will be listened on by *SecondQM*. Leave all other fields with their default values. Click the *Create* button followed by the *Close* button. Changes appear in both panes of the main window, followed by a dialog appears describing a connection exception – with the information that the connection was refused – click *Yes* to hide future messages of that type.

The dialog appeared because *SecondQM* is not running and the connection request made by *FirstQM* was not honored. This will not prevent the queue managers talking in the future when all the necessary configuration changes have been made. One consequence of our use of defaults is that we have determined that the connection will use the TCP/IP protocol.

In the tree pane a new queue manager node has appeared – *SecondQM* – equivalent in the hierarchy to *FirstQM*; however it has a modified icon alongside that indicates that MQe_Explorer has not yet been able to contact it. The MQe network now comprises two queue managers. Moreover the *MQe root\FirstQM\Connections* node now has a ⊞ sign alongside to indicate more data is available at a lower level.

Click on the ⊞ sign that has appeared in the tree to see the *SecondQM* connection node that appears below it; ensure the *Connections* folder is selected.

The list view pane shows the details of the new connection.

Double click the connection row in the right hand pane to show any child objects.

The pane is empty and the status bar indicates that there are zero queues that relate to this connection. No remote queues on *SecondQM* have yet been defined on *FirstQM* and so this is correct. More interesting is the *SecondQM* node in the tree pane directly under root.

Click the *SecondQM* queue manager node (the last node in the tree in the left hand pane). A ⊞ sign appears alongside and the right hand pane shows the next level of structure. Expand the tree and click on any of the resultant folders – there will be a delay before anything happens.

By default the delay is 10 seconds; every time a click occurs on a queue manager node that has failed to respond, MQe_Explorer will attempt a retry. The delay on displaying folder contents is the time spent before concluding that no response is forthcoming. The status bar displays "No data available" – another indication that *SecondQM* cannot be contacted. This does not mean that it cannot be managed – offline administration capability permits MQe_Explorer to asynchronously send messages to remote queue managers that are unavailable at the time.

Right click the *Local queues* folder under the *SecondQM*.

The context menu has a number of commands for dealing with queues on *SecondQM*; similarly the *Connections* and *Comms. listeners* folders offer similar options. In fact neither *FirstQM* nor *SecondQM* is fully set-up at this stage for offline administration because a suitable async. proxy queue for admin has not been defined, nor has a connection back from *SecondQM* to convey the results of any operations, and nor have any comms. listeners.

At this point it is appropriate to create the comms. listener on *FirstQM*. This is similar to creating a connection definition.

Right click on the *Comms. Listeners* folder in the tree and select the *New Comms. Listener* menu item from the context menu. In the dialog that appears enter "MyCommsListener1" into the *Name* field, and the value "8082" into the *Port* field. Click *Create* to create the comms. listener, followed by *Close* to remove the dialog.

By using defaults we have created a comms. listener that will use the TCP/IP protocol. The name of the comms. listener is only significant as a way of referring to the definition – multiple comms. listeners can exist. This is different from a connection definition, where there is no choice over the name.

Details of the newly created comms. listener appear in the list view pane. The *Comms. run state* property shows it to be stopped, i.e. it is not listening. Newly created comms. listeners must be started if they are to be used; when a queue manager is loaded any existing comms. listeners are started automatically.

Start the comms. listener by selecting it in the list view pane and clicking the start icon ▸ on the tool bar.

Configuration of *FirstQM* is complete; it has a running comms. listener that can accept incoming connection requests, i.e. it can behave as a *server*, and it can make a connection request to *SecondQM*, i.e. it can behave as a *client*.

At this point it is appropriate to create the *SecondQM* queue manager.

Using the *Windows→Close All* command close all windows except the main MQe_Explorer window.

Bringing up a second queue manager

Double click the MQe_Explorer desktop icon. A second MQe_Explorer window appears. Click the 🗋 button – the create queue manager dialog appears. Using the same steps as before, create a new queue manager *SecondQM* .

Before the queue manager is created a message will be issued giving the name and location of an initialization file. This time it is useful to examine the contents of such files.

Bring up *Notepad* (*Start menu→Programs→Accessories→Notepad*). Open the file *C:\Program Files\MQe\Java\MQe_Explorer\SecondQM.ini* (you will need to change the file types displayed to see the file in the directory).

The contents of this *.ini* file are very similar to the *FirstQM.ini* file that was created earlier. Much of the content is due to the defaults that have been taken. Notice that it contains information on where the registry is located and the name of the queue manager. This information is used whenever the queue manager is restarted.

Close *Notepad*. Arrange the two MQe_Explorer windows side by side so that the contents of each are equally visible.

Now two queue managers running on a single machine, each in their own JVM. Likewise there are two instances of MQe_Explorer running as well – only one of which is needed for admin (either could be chosen). Other applications besides MQe_Explorer could be launched into these JVMs – the menu command *Tools→Load* will load classes (which equates to applications) and launch them on an execution thread) but that goes beyond the scope of this script. The two instances of MQe_Explorer can be distinguished by the name of their local queue manager, shown in the bottom right panel of the status bar, and also by the string shown in the associated button in the Windows task bar. Although *FirstQM* has all the information it needs, the converse is not yet true.

Using the same steps as before, but now on *SecondQM*, configure a connection definition to *FirstQM*. The key differences are that the name of the connection definition must be "FirstQM" and the port number must be 8082 – this is the port being listened on by *FirstQM*.

Also on *SecondQM*, configure a comms. listener; the only difference needed here is that the port number must be 8083 – this is the port that *FirstQM* is expecting *SecondQM* to be listening on. After creation, start the new comms. listener.

For the purposes of this script, both *FirstQM* and *SecondQM* now need prompting to try again to talk to the other. In a normal situation, only one instance of MQe_Explorer would be in use and it would be that instance that needs to establish communication with its target queue managers (say *FirstQM*) – the fact that *SecondQM* is not also interrogating *FirstQM* would be of no consequence. However, for this script it is interesting to show that *FirstQM* can manage *SecondQM* and that concurrently, *SecondQM* can manage *FirstQM*.

On *FirstQM*, click the *SecondQM* local queue manager icon in the tree; likewise on *SecondQM*, click on the *FirstQM* local queue manager icon in the tree.

After a pause, the red queue manager icons change from to – this means that information about each of those queue managers has been requested and received.

On FirstQM, expand all the nodes of the tree under *MQe root\SecondQM*.

The full object structure of the *SecondQM* queue manager is visible from *FirstQM*. Any delays in the expansion are a consequence of *FirstQM* getting additional information on *SecondQM*.

On the *FirstQM* queue manager, select the node *MQe root\FirstQM\Connections\SecondQM* in the tree view pane and click *Refresh*.

FirstQM has gained a number of sync. proxy queues; these have been created automatically by through the process of queue discovery, prompted by *FirstQM* either sending messages to the remote application queues or by its attempting to browse them.

To demonstrate that the queue managers can exchange messages, a test message can be sent from *FirstQM* to *SecondQM*.

On *FirstQM*, right click *MQe root\SecondQM\Local queues\DeadLetterQ* and choose *New message*. Taking all the defaults, click the *Send* button. A confirmation message appears; click *OK* and then remove the panel.

On *SecondQM* select the node *MQe root\SecondQM\Local queues\DeadLetterQ* and *Refresh* the node.

The list view pane shows the message just sent from *FirstQM*. It can be drilled into as demonstrated previously. The message could also have been seen from *FirstQM* by selecting *MQe root\FirstQM\Connections\SecondQM\DeadLetterQ* on that queue manager (a *Refresh* of the parent may be needed – depending upon whether it has previously been accessed).

Advanced queue properties

Now that a remote queue manager exists, queues can be re-visited and more advanced queue types explored.

On *FirstQM*, in the tree pane right click on *MQe root\SecondQM\Local queues* and select the *New Queue* item.

The new queue dialog is displayed, primed to (remotely) create an application queue on *SecondQM*. A *forward queue* can be defined. Imagine that two additional remote queue managers *ThirdQM* and *FourthQM* exist. Messages are to reach them by being sent to *FirstQM*. A *forward queue* can be configured to handle this situation.

Select *Forward* in the *Type* field; enter "ForwardQ" in the *Name* field and "Forward queue" in the *Description* field. In the *Forward to* field enter (or select from the drop-down list) "FirstQM". On the *Destinations* tab type "ThirdQM" and click *Add*. Then type "FourthQM" and click *Add*. Then click *Create*.

In the tree pane there is a new node under the connection to *FirstQM*, i.e. *MQe root\SecondQM\Connections\FirstQM\ForwardQ* with a new icon . This indicates a local[33] queue that gathers messages for the target queue managers and passes them on to *FirstQM*. It is assumed that *FirstQM* is to be further configured to have connection definitions for these two new queue managers; alternatively it could have another forward queue that passed messages for them on elsewhere.

Similarly a store queue can be defined that gathers messages but does not forward them. Typically such a queue would hold them ready for them to be pulled by a *Home server queue*.

Re-using the new queue dialog, ensure *Store* is selected in *Type* field, type "StoreQ" in the *Name* field and the "Store queue" in the *Description* field. On the *Destinations* tab type "FifthQM" and click *Add*. Then click *Create* and then click *Close*.

The new queue appears as a local queue with a new icon . A matching *Home server queue* could now be created on the queue manager *FifthQM* to get messages from this store queue – if such a queue manager had been created. Instead, we will pretend that this store queue also holds messages for *FirstQM*. Then on *FirstQM* the complementary *Home server queue* would be defined as follows.

On *FirstQM* select the node *MQe root\FirstQM\Connections\SecondQM* and right click to define a new queue. In the *Name* field type "StoreQ", the *Description* is "Home server queue" and the *Type* is set to *Home server*. Click *Create* followed by *Close*.

The new home server queue is visible along with other remote queues associated with that remote queue manager – and the associated icon is . However, since this queue is not appropriate because it will never actually get any messages for *FirstQM*, it can be deleted.

Select the queue in either the tree or the list view pane and delete it by hitting the delete key. Click *OK* on the confirmation messages.

Although some interesting queue types have been configured they have not been used for moving messages. Likewise only synchronous messaging has been demonstrated. Further investigation is deferred to later scripts[34].

Queues have many useful properties – and all can be exploited by rules. The security attribute can be easily illustrated[35]. A queue will be created that is protected by an authenticator – in this case the sample NT authenticator that is supplied with MQe.

[33] Although it is a local queue, it appears in the tree under a connection definition, indicating where it sends its messages.
[34] Script *Basic messaging* on page 136 illustrates direct & indirect connections and synchronous and asynchronous messaging. Script *Advanced messaging* on page 143 illustrates pushing and pulling messages, use of intermediate queues and the details of home server, store and forward queues.
[35] This section cannot be run on Windows 98 systems – the sample MQe NT authenticator is inappropriate; nor can it be run unless MQe has been installed.

On *FirstQM* select the node *MQe root\FirstQM\Local queues* and bring up the *New queue definition* dialog. In *Name* type "ProtectedQ", in *Description* type "NT authenticator protected queue". On the *Security* tab select the *Authenticator* value *examples.attributes.NTAuthenticator*. Click *Create* followed by *Close*. The new queue is seen in the right hand pane. Double click the *ProtectedQ* row in that pane. A panel appears requiring a valid NT user id and password before access is permitted[36].

The queue is now protected through the example NT authenticator. In practice a custom authenticator would be used with an appropriate user interface – or some other way of ensuring the identity of the user.

The *Expiry* property is another useful attribute, ideal for preventing transient data filling queues with out of date information.

Using the *New queue definition* dialog: in *Name* type "TransientQ", in *Description* type "Transient queue". On the *Properties* tab uncheck the *No limit* against *Expiry* and enter a value of 10000 (i.e. 10 secs.) in the field. Click *Create* followed by *Close*. The new queue is seen in the tree. Select the new queue and send it a test message using the right context menu. Quickly refresh the queue content display using the button in the toolbar – the new message will be seen. Refresh again a few seconds later – the message will have disappeared.

Messages only persist on the *TransientQ* queue for 10 seconds as specified – then they are deleted.

Miscellaneous features

On the *Tools* menu select *Options* and explore the *Options* dialog.

The initial tabs, *QMgrs, Connections, Comms. listeners,* etc. provide layout control over the various list view panes – the columns displayed and their respective order. The *Properties* tab is used to set the default widths used for these properties. The *Classes* tab controls the contents of list boxes and other related input fields and thus allows MQe_Explorer to be customized. The three *Advanced* tabs control the queues that are generated and/or used, and the number of threads allowed, timeouts etc. and the proxy settings when HTTP is used as a protocol.

This first script does not demonstrate many MQe_Explorer and MQe capabilities, such as: offline working; the loading of concurrent applications into a queue manager; exploitation of rules; and advanced features such as security.

[36] Enter valid details here (*Domain name*, *User id* and *Password*) and if set-up has been correct, access will be granted – note that if you are not part of a Windows 2000 domain (for example, running standalone) you must leave the *Domain* field with its default value intact. If you have failed to set-up Windows security correctly and you attempt to validate a user id/password you will get an Exception message – to recover click *Continue* and then click elsewhere in the left hand pane tree. If you know that you have not set-up Windows security then click *Cancel* in the authentication dialog – this will avoid the exception.

Resetting the script

To reset the script to its initial state delete the following files and directories (and their associated contents and sub-directories):

C:\Program Files\MQe\Java\MQe_Explorer\FirstQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\FirstQM

C:\Program Files\MQe\Java\MQe_Explorer\SecondQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\SecondQM

13.2 Basic messaging

Overview

This script illustrates some of the simpler messaging network configuration and delivery options that are available with MQe. Familiarity with the operations detailed in the *Concepts and objects* script is assumed.

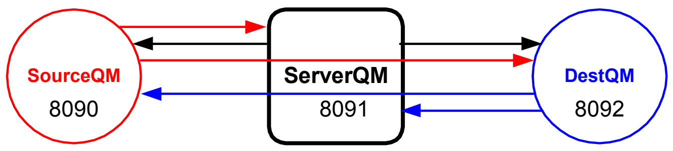
The following features are shown:

- Synchronous messaging (with both direct routing and indirect routing)
- Asynchronous messaging (with both direct and indirect routing) using:
    - *Per queue* queuing (i.e. sync. and async. proxy queues).
    - *Shared queue* queuing (i.e. store and forward queues).

Note: The script uses test messages that have built in expiry – hence, if long delays occur between steps, the described behavior may not be seen.

Scenario (direct & indirect routing)

The script sets up the following configuration:



Connection definitions

ServerQM (direct)	*SourceQM (direct)*	*ServerQM (direct)*
DestQM (via ServerQM)	*DestQM (direct)*	*SourceQM (via ServerQM)*

Figure 13-1: Basic messaging scenario

Three queue managers are used (*SourceQM*, *ServerQM* and *DestQM*) each of which can send messages to the others. The script generates direct connections as follows:

SourceQM to *ServerQM*

DestQM to *ServerQM*

ServerQM to *SourceQM*

ServerQM to *DestQM*

and the indirect connections:

SourceQM to *DestQM* (via *ServerQM*)

DestQM to *SourceQM* (via *ServerQM*)

Script

Introduction

MQe supports both *direct* connections and *indirect* connections. A *direct* connection exists when one queue manager has a single transport link to another queue manager. An *indirect* connection means that no direct transport exists, but instead one or more intermediate queue managers must be traversed between the source and target queue managers. Indirect connections permit transport protocol changes to occur between the source and the target queue managers; they can also allow intermediate queuing to be configured.

Creating the queue managers

Three queue managers are required: *SourceQM*, *ServerQM* and *DestQM*.

Click three times on the MQe_Explorer desktop icon and then create three new queue managers with the following characteristics.

1. ***QMgr. name*: SourceQM**
   ***All other boxes*: default values**
2. ***QMgr. name*: ServerQM**
   ***All other boxes*: default values**
3. ***QMgr. name*: DestQM**
   ***All other boxes*: default values**

Re-arrange the windows such that they are organised leftwards and stacked vertically in the order *SourceQM* (top), *ServerQM* (middle) and *DestQM* (bottom).

The default admin time-outs should be reduced to minimize delays in waiting for a response (note that this script frequently attempts to talk to queue managers that are unavailable).

On each of the queue managers, from the *Tools/Options* menu, on the *Options* property pages, display the *Advanced-2* tab. In *Object access* reduce the *Timeout* to 3 seconds. Click *Apply*, then *Close*[37,38].

Creating the connections and comms. listeners

The direct connections should be created first. Using the respective instances of MQe_Explorer, create the following connections:

1. ***On queue manager* SourceQM:**

***Name*:**	**ServerQM**
***Description*:**	**Server queue manager**
***Type*:**	**Direct**
***IP address*:**	**127.0.0.1**
***Port*:**	**8091**

[37] This script (unusually) uses a shared options file across the three queue managers. Changes to the settings for the first queue manager change the saved (but not loaded) values for the others – clicking *Apply* for all three will ensure that they each use the changed values. This shared usage is not good practice but is convenient here.
[38] These settings are suitable for this script; it is not intended to suggest that they are suitable for operational use.

2. *On queue manager* ServerQM:

Name:	SourceQM
Description:	Source queue manager
Type:	Direct
IP address:	127.0.0.1
Port :	8090

3. *On queue manager* ServerQM:

Name:	DestQM
Description:	Destination queue manager
Type:	Direct
IP address:	127.0.0.1
Port:	8092

4. *On queue manager* DestQM:

Name: ServerQM	
Description:	Server queue manager
Type:	Direct
IP address:	127.0.0.1
Port:	8091

Create the indirect connections:

1. *On manager* SourceQM:

Name:	DestQM
Description:	Destination queue manager
Type:	Indirect
Via qMgr:	ServerQM

2. *On queue manager* DestQM:

Name:	SourceQM
Description:	Source queue manager
Type:	Indirect
Via qMgr:	ServerQM

Create the comms. listeners:

1. *On queue manager* SourceQM:

Name:	CommsListener1
Port:	8090

2. *On queue manager* ServerQM:

Name:	CommsListener1
Port:	8091

3. *On queue manager* DestQM:

Name:	CommsListener1
Port:	8092

Start the three comms. listeners and click all the red local queue manager icons in each MQe_Explorer instance so that they each query the others.

The three queue managers have been created and communications configured. After a brief delay, the available queue manager icons have replaced all the unavailable queue manager icons in the trees and they have exchanged information.

Expand the tree in the *SourceQM* instance of MQe_Explorer.

A detailed view of the network is available – and a similar view will be available from the other queue managers. A number of sync. proxy queues have already been created automatically by MQe – these are a consequence of queue discovery occurring during MQe_Explorer's use of admin operations. Exactly what is visible depends upon what had been created at the time the tree was expanded – to get the current situation *Refresh* should be used. For example:

On *SourceQM* refresh nodes *MQe root\SourceQM\Connections\ServerQM* and *MQe root\SourceQM\Connections\DestQM*.

Two sync. proxy queues (*AdminQ* and *AdminReplyQ*) have been created for each connection.

From this point, *SourceQM* will arbitrarily be used for the admin operations.

Minimize *ServerQM* and *DestQM*. On *SourceQM* refresh the tree from *MQe root\DestQM*, *MQe root\ServerQM* and *MQe root\SourceQM*.

A custom target application queue can now be created and used to investigate the sending of messages, both directly or indirectly, and synchronously or asynchronously.

Creating the queues

Using MQe_Explorer on *SourceQM*, remotely create an application queue on *DestQM* with the following properties:

1.
***Name*:**	**DestQ**
***Description*:**	**Destination queue**
***Type*:**	**Application**
***Aliases*:**	**SyncDestQ, AsyncDestQ**

Using the toolbar, or *View* menu, set *Use multiple rows for aliases*. Three extra rows are now visible in the right hand pane (*DestQ*, *SyncDestQ* and *AsyncDestQ*).

DestQ is a standard application queue, its only slightly unusual aspect is it has two alias names[39] - these will be exploited in sync. and async. proxy queues on *SourceQM* to enable synchronous and asynchronous access to the same physical queue.

Still using MQe_Explorer on *SourceQM*, create the following proxy queues (i.e. right click on *MQe root\SourceQM\Connections\DestQM*).

1.
***Name*:**	**SyncDestQ**
***Description*:**	**Synchronous access to DestQ**
***Type*:**	**Sync. proxy**
***Local qMgr*:**	**SourceQM**
***Queue qMgr*:**	**DestQM**

[39] The script could be re-worked to use just one alias name – but using two makes the steps more symmetrical and hence clearer.

2.	***Name*:**	**AsyncDestQ**
	***Description*:**	**Asynchronous access to DestQ**
	***Type*:**	**Async. proxy**
	***Local qMgr*:**	**SourceQM**
	***Queue qMgr*:**	**DestQM**

The new queues can be seen in the list view pane.

The queue managers are configured so that messages can be sent either synchronously and asynchronously from *SourceQM* to *DestQM*. They will go indirectly (that is, via *ServerQM* because that is the connectivity defined between *SourceQM* and *DestQM)*.

Direct and indirect, synchronous and asynchronous messaging

Using *SourceQM*, by right clicking on *MQe root\SourceQM\Connections\DestQM\AsyncDestQ* and *MQe root\SourceQM\Connections\DestQM\SyncDestQ*[40] send the following two test messages:

1.	***Contents*:**	**A message sent asynchronously from SourceQM**
	***Destination queue mgr*:**	**DestQM**
	***Destination queue*:**	**AsyncDestQ**
2.	***Contents*:**	**A message sent synchronously from SourceQM**
	***Destination queue mgr*:**	**DestQM**
	***Destination queue*:**	**SyncDestQ**

Both messages are successfully sent. To confirm that they arrived they can be inspected on their destination queue.

Collapse the MQe_Explorer tree on *SourceQM* and then expand *MQe root\DestQM\Local queues*. Refresh *MQe root\DestQM\Local queues\DestQ*. Display its contents in the right hand pane.

Two messages appear; the *Message origin queue manager* property shows that they were sent from *SourceQM*; the *Message creation date* values correspond to the times they were sent.

It is interesting to note that the asynchronous message was sent immediately – that happened because the rules associated with the *SourceQM* queue manager and the definition of the queue *DestQ* on *SourceQM* specified this particular behavior. Alternate rules could have been in place that enforced different behavior, for example that asynchronous messages should only be transmitted immediately if they had a high priority set, that less important messages were transmitted off-peak, or when at least 10 were pending, … etc.

Explore the contents of each message – the *Value* column of the *Message* property identifies which was sent synchronously and which asynchronously.

Messages have now been sent both synchronously and asynchronously, in both cases going indirectly via *ServerQM*. At the same time all admin operations have been through the MQe_Explorer running on *SourceQM*; it has been communicating synchronously and directly to the relevant queue managers.

Asynchronous messaging has behaved identically to synchronous messaging in the script so far because all the queue managers (and hence all the connections) have been available. The power of asynchronous messaging can be illustrated by disabling the intermediate queue manager *ServerQM*.

[40] Another way of synchronously sending a message to *DestQ* is to right click *MQe root\DestQM\Local queues\DestQ* – this will use the remote queue definition for *DestQ* on *SourceQM*.

Restore the window for *ServerQM* and close the queue manager; minimize the window. Now send a further two messages through the *SourceQM* MQe_Explorer. Right click on *MQe root\SourceQM\Connections\DestQM\AsyncDestQ* and *MQe root\SourceQM\Connections\DestQM\SyncDestQ* and send the following:

1.	***Contents*:**	**Second async send from SourceQM**
	***Destination queue manager*:**	**DestQM**
	***Destination queue*:**	**AsyncDestQ**
2.	***Contents*:**	**Second sync send from SourceQM**
	***Destination queue manager*:**	**DestQM**
	***Destination queue*:**	**SyncDestQ**

The asynchronously transmitted message is sent successfully; the synchronously transmitted message cannot be sent – because a channel cannot be established to the destination queue *DestQ*. It is interesting to locate the message that was sent.

On *SourceQM* attempt to refresh the properties of *MQe root\SourceQM\Connections\DestQM\AsyncDestQ.*

The queue's contents cannot be accessed because this async. proxy queue intentionally forbids browse (amongst other operations). However properties of the queue can be inspected.

On *SourceQM* display the properties of *MQe root\SourceQM\Connections\DestQM\AsyncDestQ.*

The queue depth is now shown as one; this is the message waiting to be sent and currently held in the backing store.

Using the *Storage* tab, click in the *Path* value field and then move the cursor to the right to display the entire value (that is, *C:\Program Files\MQe\Java\MQe_Explorer\SourceQM\Queues*). Then (without closing the *Properties* page) using *Windows Explorer*, show the contents of *C:\Program Files\MQe\Java\MQe_Explorer\SourceQM\Queues\DestQM\AsyncDestQ.*

A pair of files of types .*Msg* and .*MQe* represent the message.

The *ServerQM* queue manager can be re-activated so that communication between *SourceQM* and *DestQM* can be restarted.

In the MQe_Explorer instance that was previously used for *ServerQM*, restart the queue manager by opening the file *C:\Program Files\MQe\Java\MQe_Explorer \ServerQM.ini* and then click on *MQe root* in the tree. Minimize this window and then go back to MQe_Explorer on *SourceQM*; refresh *MQe root\DestQM\Local queues\DestQ.* Display its contents in the right hand pane.

No new message has arrived – in fact, if the property pages of the async. proxy queue on *SourceQM* are refreshed, the queue depth on the *Storage* tab is still shown as 1[41].

[41] Be aware that the depth is only displayed when the queue is in the *active* state. For demonstration purposes if you want to see what messages are queued at any time, look in the file system.

Click the *Refresh* button on the *C:\Program Files\MQe\Java\MQe_Explorer\SourceQM\Queues\DestQM\AsyncDestQ* properties page and examine the queue depth.

The message appears to be stuck; in fact MQe is enforcing the queue manager and queue rules on *SourceQM* – the default rules state that messages are triggered for transmission when the queue manager starts up, or when a message arrives on a queue. In this case the queue manager had already started – and the message was already there. If another had been sent – then both would have been transmitted – this behavior is also determined by the rules.

An easy solution in this case is to trigger the transmission manually using the trigger button on the toolbar.

Click the *Trigger* button on the toolbar – then refresh the properties page again – the queue depth is now zero. Click *Close*.

An alternative approach is to changing the queue manager rule. A new rule class *examples.mqe_explorer.QmRule*[42, 43] triggers message transmission every 5 seconds.

On the *SourceQM* queue manager change the queue manager rule (via the queue manager properties) to *examples.mqe_explorer.QmRule*.

The new rule is now in effect but the change in triggering behavior only happens when the queue manager is re-activated[44].

On *SourceQM* display the contents of *MQe root\DestQM\Local queues\DestQ* contents in the right hand pane (three messages).

The script *Advanced messaging* on page 143 can now be used with this same configuration to illustrate some of the more advanced network messaging options.

Resetting the script

Full reset

If the *Advanced messaging* script is not going to be used, then to reset this script to its initial state delete the following files and directories (and their associated contents and sub-directories):

C:\Program Files\MQe\Java\MQe_Explorer\SourceQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\SourceQM

C:\Program Files\MQe\Java\MQe_Explorer\ServerQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\ServerQM

C:\Program Files\MQe\Java\MQe_Explorer\ DestQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\DestQM

To restore the MQe_Explorer option file to its initial state click *Tools→Options→Reset*.

[42] If you are using *MQe_ExplorerV2.exe* ensure that you have installed the new rule class shipped with MQe_Explorer before proceeding (install via *MQe_ExplorerCV2.exe*).
[43] This is a variation on the *examples.rules.ExampleQueueManagerRules* class provided with MQe.
[44] This queue manager rule only takes effect when the queue manager is activated – it would have been possible to write it in such a way that a queue manager restart was not required. In general, a rule change does not require a queue manager restart.

13.3 Advanced messaging

Overview

This script illustrates more advanced of the messaging network configurations available with MQe. It is designed to follow on immediately from the previous script.

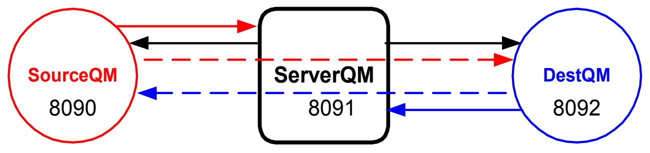
The following features are illustrated:

- Asynchronous messaging:
    - *Message push* (i.e. async. proxy and forward queues)
    - *Message pull* (i.e. home server queues)
    - *Shared intermediate* queues
        - Locally
        - Within the network

Note: The script uses test messages that have built in expiry – hence, if long delays occur between steps, the described behavior may not be seen.

Scenario

The configuration is almost identical to that of the previous script. As before, *SourceQM* is sending messages to *DestQM* via *ServerQM*. In this case however the indirect connections between *SourceQM* and *DestQM* are no longer needed for application messaging, however they are needed if network administration is conducted from either *SourceQM* or *DestQM*[45].



Application connection definitions (SourceQM to DestQM)

ServerQM (direct)	*DestQM (direct)*	*ServerQM (direct)*

Admin connection definitions

ServerQM (direct) *DestQM (via ServerQM)*	*SourceQM (direct)* *DestQM (direct)*	*ServerQM (direct)* *SourceQM (via ServerQM)*

Figure 13-2: Advanced messaging scenario

Note: it is assumed that that the queue managers *SourceQM*, *ServerQM* and *DestQM* exist, along with their associated properties and queues.

[45] For simplicity the connection definitions are left unchanged from script 2. If the script is to be run without the indirect connections then it must be modified so that all admin is either conducted from *ServerQM* or each queue manager is administered directly from its local instance of MQe_Explorer.

Script

Introduction

MQe supports the use of intermediate queues (store, forward and async. proxy queues) so that messages can be queued within the network.

Store and forward queues hold messages bound for one or more queue managers – they are insensitive to the identity of the target queue on a particular queue manager. The first task in this script is to set up a forward queue on *SourceQM* that will hold all messages destined for *DestQ* on *DestQM* – previously they were held in the *SourceQM*'s async. proxy queue *AsyncDestQ* on *DestQM*.

Preparation

The queue managers require minor configuration changes. On *DestQM* all the messages sent previously to the *DestQ* on *DestQM* must be deleted.

Using *DestQM*, select *MQe root\DestQM\Local queues\DestQ* in the tree. In the right hand pane select a message, followed by *Control A*, and followed by the *Delete* key. Accept the deletion of all messages.

No messages must exist elsewhere – for example, on the async. proxy queue *AsyncDestQ* on *SourceQM*.

Using *Windows Explorer* show that the following directory is empty (and if not – delete any files present):

C:\Program Files\MQe\Java\MQe_Explorer\SourceQM\Queues\DestQM\ AsyncDestQ

In order for messages to move when possible the queue manager rules for *DestQM* and *ServerQM* require modification[46] – those for *SourceQM* have already been changed in the script *Basic messaging* on page 136.

Using *DestQM*, change its queue manager rule to *examples.mqe_explorer.QmRule* (the class is available on the drop-down if the default *Tools→Options* are in force – or use the *Reset* button to reset to the defaults). Close the queue manager. Do the same rule change to *ServerQM* and close the queue manager.

The basic configuration changes have been made. The queue managers can now be started.

Bring up *SourceQM* and *DestQM* followed by *ServerQM*. Arrange the windows on the screen so that they are stacked vertically, *SourceQM* at the top, followed by *ServerQM* and then *DestQM*.

Using a forward queue on the source queue manager

A forward queue on *SourceQM* can be defined that will collect messages bound for *DestQM* and send them to *ServerQM*. Forward queues can collect messages for more than one destination but in this case only one is needed.

[46] If you are using *MQe_ExplorerV2.exe* ensure that you have installed the new rule class shipped with MQe_Explorer before proceeding (install via *MQe_ExplorerCV2.exe*).

Define a queue on *SourceQM* with the following properties (right click on *MQe root\SourceQM\Connections\ServerQM*):

1.

***Name*:**	**FQ**
***Description*:**	**Forward queue for DestQM**
***Type*:**	**Forward**
***Local qMgr*:**	**SourceQM**
***Forward to*:**	**ServerQM**
***Destinations*:**	**DestQM**

Now a message can be sent from *SourceQM* to the target queue *DestQ* at *DestQM*. Repeating previous practice the alias name for this queue will be used, that is *AsyncDestQ*.

Using MQe_Explorer on *SourceQM*, send a test message to *AsyncDestQ* at *DestQM* from *SourceQM* (right click on *MQe root\SourceQM\Connections\DestQM\AsyncDestQ*). Then refresh and display the contents of *MQe_root\DestQM\Local queues\DestQ*.

The message arrives as expected. However in order to see how it was routed *DestQM* needs to be taken offline.

Close the *DestQM* queue manager (leave the window available) – then resend the message as previously. Using *Windows Explorer* inspect the contents of:

C:\Program Files\MQe\Java\MQe_Explorer\SourceQM\Queues\DestQM\AsyncDestQ

The new forward queue has had no effect – the message is waiting in the async. proxy queue *AsyncDestQ*. This is because behavior determined by a proxy queue takes precedence over that arising from a forward queue. If the *AsyncDestQ* on *SourceQM* is deleted, the use of the forward queue will become apparent. However the *AsyncDestQ* queue cannot be deleted until it has delivered its message (or until the message has expired).

Using MQe_Explorer on *SourceQM*, attempt to delete the async. proxy queue on *SourceQM* for *AsyncDestQ* at *DestQM* – an error results. In order to do this, bring up *DestQM* and then if necessary, recycle *ServerQM*. Then repeat the deletion attempt (the queue is successfully deleted). Take down *DestQM*.

Send a test message to *AsyncDestQ* at *DestQM* from *SourceQM* – to do this right click on any queue in the path *MQe root\DestQM\Local queues*, or in the path *MQe root\SourceQM\Connections\DestQM*. Then select *New*, *Message* and in the *Destination queue* box type in the queue name "AsyncDestQ"; click *Send*. Using *Windows Explorer*, display the contents of *C:\Program Files\MQe\Java\MQe_Explorer\SourceQM\Queues\ServerQM\FQ*.

The message is now queued for transmission in the forward queue at the source queue manager. Another consequence of this change is that the properties of *FQ* on *SourceQM* can now be used to determine the security attributes associated with transmission – in this case however none have been set.

Using a forward queue in the network

The second task in this script is to arrange for messages to be stored in a forward queue in the network (that is, on *ServerQM*) when *DestQM* is unavailable. The first step is therefore to define the new queue and then experiment sending messages.

Using MQe_Explorer on *SourceQM*, display the remote queue definitions on *ServerQM* at *DestQM*, that is, select *MQe root\ServerQM\Connections\DestQM*. If *AsyncDestQ* exists, delete it (note that it will be defined as a synchronous connection). Then define a queue on *ServerQM* with the following properties:

1.

***Name*:**	**FQ**
***Description*:**	**Forward queue for DestQM**
***Type*:**	**Forward**
***Local qMgr*:**	**ServerQM**
***Forward to*:**	**DestQM**
***Destinations*:**	**DestQM**

Send a test message to *AsyncDestQ* at *DestQM* from *SourceQM* (as previously). Using the Windows Explorer, display the contents of *C:\Program Files\MQe\Java\MQe_Explorer\SourceQM\Queues\ServerQM\FQ* (no messages) and *C:\Program Files\MQe\Java\MQe_Explorer\ServerQM\Queues\DestQM\FQ* (four files, i.e. two messages).

The message is now queued on the server – as is the message sent previously (now moved closer to delivery). If the server had not been available then the message would have been queued at the source, in the *FQ* queue there. As previously, this only works because there is no proxy queue for *AsyncDestQ* on *ServerQM* – if such a queue was present then the *FQ* on *ServerQM* would not be used for any messages destined for *AsyncDestQ* on *DestQM*.

If *DestQM* is now brought up, it will now receive all the messages that are queued for it – this behavior occurs because queue manager rules were changed from the defaults to *examples.mqe_explorer.*QmRule – which requires that the queue manager attempt message delivery every 5 secs.

Re-load *DestQM*. In the right hand pane display the contents of *MQe root\DestQM\Local queues\DestQ*; then delete all the messages.

Using a home server queue on the target queue manager

Home server queues are used in conjunction with store queues – they provide an ability to pull messages from a server. This can be illustrated as follows – first the *FQ* queue on *ServerQM* must be replaced with a new queue *StoreQ* that stores message for *DestQM* but does not forward them.

Using the MQe_Explorer on *SourceQM*, delete the *MQe root\ServerQM\Connections\DestQM\SFQ*. Create a new local queue on *ServerQM* with the following properties (right click on *MQe root\ServerQM\Local queues*):

1.

***Name*:**	**StoreQ**
***Description*:**	**Store queue for DestQM**
***Type*:**	**Store**
***Local qMgr*:**	**ServerQM**
***Destinations*:**	**DestQM**

Now a message can be sent from *SourceQM* to *AsyncDestQ* at *DestQM* – note that all three queue managers: *SourceQM*, *ServerQM* and *DestQM* are now active.

Send a message from *SourceQM* to *AsyncDestQ* at *DestQM*. Using Windows Explorer, show the contents of *C:\Program Files\MQe\Java\MQe_Explorer\ServerQM\Queues\ServerQM\StoreQ* (one message).

The message has reached the store queue – however it has not been forwarded to *DestQM* even though that queue manager is available. However if a home server queue on *DestQM* is defined it can be configured to pull messages from the store queue on *ServerQM*.

Using the MQe_Explorer on *SourceQM*, create a new application queue on *DestQM* with the following properties (right click on *MQe root\DestQM\Connections\ServerQM*):

1.	***Name*:**	**StoreQ**
	***Description*:**	**Home server queue pulling from ServerQM**
	***Type*:**	**Home server**
	***Local qMgr*:**	**DestQM**
	***Get from*:**	**ServerQM**

Display the contents of *MQe root\DestQM\Local queues\DestQ*.

The message has been successfully pulled from the server.

Using proxy queues in the network

One characteristic of store and forward queues is that are queue manager-based, i.e. they route based on queue manager. Routing and/or storing messages at the queue level is enabled through proxy queues.

In the example that follows, only async. proxy queues are used to exploit *ServerQM* as a message server for the target queue *DestQ* on *DestQM*.

First, delete all store and forward queues and the home server queue.

Using MQe_Explorer on *SourceQM*, refresh the views of *ServerQM* and *DestQM*. Then delete the following queues:

MQe root\SourceQM\Connections\ServerQM\FQ

MQe root\ServerQM\Local queues\StoreQ

MQe root\DestQM\Connections\ServerQM\StoreQ

Create the following queues (using *MQe root\SourceQM\Connections\DestQM* and *MQe root\ServerQM\Connections\DestQM*:

1.	***Name*:**	**AsyncDestQ**
	***Description*:**	**Asynchronous access to DestQ**
	***Type*:**	**Async. proxy**
	***Local qMgr*:**	**SourceQM**
	***Queue qMgr*:**	**DestQM**

2.	***Name*:**	**AsyncDestQ**
	***Description*:**	**Asynchronous access to DestQ**
	***Type*:**	**Async. proxy**
	***Local qMgr*:**	**ServerQM**
	***Queue qMgr*:**	**DestQM**

Using MQe_Explorer on *DestQM,* close the queue manager so that it is no longer able to receive messages.

Using MQe_Explorer on *SourceQM*, send a message from *SourceQM* to *AsyncDestQ* by clicking on *MQe root\SourceQM\Connections\DestQM\AsyncDestQ.*

Using Windows Explorer, show the contents of *C:\Program Files\MQe\Java\MQe_Explorer\ServerQM\Queues\DestQM\AsyncDestQ* (one message).

The message just sent is now queued on the server. This technique allows individual target queue-based security to be applied, but requires queue level definitions on the server – thus it is potentially more expensive in terms of network definitions. If *DestQM* is brought back up, the message will be delivered.

Reload *DestQM*. Show the contents of *MQe root\DestQM\Local queues\DestQ.* Using Windows Explorer, show the contents of *C:\Program Files\MQe\Java\MQe_Explorer\ServerQM\Queues\DestQM\AsyncDestQ* (no messages).

The script has demonstrated how MQe can queue messages at the source or within the network, how messages can be pushed, or alternatively pulled. It has also illustrated the implementation of queue manager- and queue-based routing.

Resetting the script

To reset the script to the state existing before the script *Basic messaging*, delete the following files and directories (and their associated contents and sub-directories):

C:\Program Files\MQe\Java\MQe_Explorer\DestQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\DestQM

C:\Program Files\MQe\Java\MQe_Explorer\ServerQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\ ServerQM

C:\Program Files\MQe\Java\MQe_Explorer\ SourceQM.ini

C:\Program Files\MQe\Java\MQe_Explorer\SourceQM

13.4 Gateway configuration and use

Overview

This script demonstrate how to use the WebSphere MQ Explorer management tool on an MQSeries queue manager, together with the MQe_Explorer on MQe, to exchange test messages between a WebSphere MQ queue manager, an MQe gateway and an MQe server.

Scenario

Test messages are sent between all queue managers in a simple, three queue manager network. The topology is a simple one, very similar to that described in the *MQe Programming Guide*. The network comprises three queue managers:

BRANCH000QM – an MQe server.

GATEWAY00QM – an MQe gateway.

CENTRAL00QM – a WebSphere MQ queue manager.

Message traffic is pushed from *BRANCH000QM* through the *GATEWAY00QM* to the *CENTRAL00QM*; message traffic is pulled from *CENTRAL00QM* by the *GATEWAY00QM*, and pushed on to the *BRANCH000QM*.

Note: The script uses upper-case names for the queue managers.

Script

Step 1: Create and configure the WebSphere MQ queue manager

Use the *Start→WebSphere MQ→WebSphere MQ Explorer* to create a WebSphere MQ queue manager called "CENTRAL00QM". Set the queue manager up so it listens on the default port 1414 (tick the *Create listener for TCP/IP* checkbox).

Step 2: Create the sync queue on the WebSphere MQ queue manager

The *GATEWAY00QM* MQe queue manager (to be created later in this script) requires some MQSeries objects configured, in order to allow the MQe queue manager to talk to the WebSphere MQ *CENTRAL00QM* queue manager. One such object is a *sync queue*. This is a local queue that must be persistent.

Using the WebSphere MQ Explorer, create a local queue, called *SYNC.Q.FOR.GATEWAY00QM*. Allow both *put* and *get* of messages. Make the *Default Persistence* have the value "Persist", normal usage.

The sync queue is used by the MQe bridge and holds the persistent state information generated whilst moving messages between the MQe and WebSphere MQ systems (this information is required for assured delivery).

Step 3: Create a server connection channel on the WebSphere MQ queue manager

The bridge to WebSphere MQ also requires a *server connection channel* object, as the MQe bridge may be configured to connect to the WebSphere MQ system using a WebSphere MQ client channel.

Use the WebSphere MQ Explorer to create a server connection channel on the *CENTRAL00QM* queue manager:

***Channel Name*:**	**FOR.GATEWAY00QM**
***Transmission Protocol*:**	**TCP/IP**

Step 4: Create a transmission queue to MQe

In order to route a message from the WebSphere MQ system to the *GATEWAY00QM* MQe queue manager, we also need to create a transmission queue. This transmission queue will not have an associated WebSphere MQ channel; it must however be persistent. You could create more than one such transmission queue, if you wanted to partition the traffic heading to MQe using different routes, but one is enough in this example topology.

Messages bound for the *GATEWAY00QM* or *BRANCH000QM* will be directed to this transmission queue. Later, we will define an MQe bridge object, a *transmission queue listener*, which has the job of pulling the messages from this transmission queue and putting the messages onto the MQe network.

Using the WebSphere MQ Explorer, create the local queue on the *CENTRAL00QM* queue manager as follows:

***Queue Name*:**	**TO.GATEWAY00QM**
***Put Messages*:**	**Allowed**
***Get Messages*:**	**Allowed**
***Default Persistence*:**	**Persist**
***Usage*:**	**Transmission**

Step 5: Create queue manager aliases to refer to the MQe queue managers

Now set up a remote queue manager alias such that the *BRANCH000QM* and *GATEWAY00QM* can both be sent messages, and that such messages are routed to the *TO.GATEWAY00QM* transmission queue.

Using the WebSphere MQ Explorer on the *CENTRAL00QM* queue manager, create the following remote queue definition:

***Queue Name*:**	**GATEWAY00QM**
***Put Messages*:**	**Allowed**
***Default Persistence*:**	**Persist**
***Remote Queue Name*:**	***Leave this blank.***
***Remote Queue Manager Name*:**	**GATEWAY00QM**
***Transmission Queue Name*:**	**TO.GATEWAY00QM**

This special style of remote queue definition is known as a *remote queue manager alias*. It tells WebSphere MQ to put any messages that are destined for the *GATEWAY00QM* queue manager to the *TO.GATEWAY00QM* transmission queue.

For the *BRANCH000QM* MQe queue manager, using the WebSphere MQ Explorer on the *CENRAL00QM* queue manager, create the remote queue definition:

***Queue Name*:**	**BRANCH000QM**
***Put Messages*:**	**Allowed**
***Default Persistence*:**	**Persist**
***Remote Queue Name*:**	***Leave this blank***
***Remote Queue Manager Name*:**	**BRANCH000QM**
***Transmission Queue Name*:**	**TO.GATEWAY00QM**

Repeat for any other MQe queue managers on the MQe network to which you wish to send messages, for example, *BRANCH001QM*, *BRANCH002QM*, … etc.

Using remote queue manager aliases in this way requires only one remote queue definition per MQe queue manager, even if your application puts to six queues on the MQe queue manager. You could alternatively define exact remote queue definitions for every queue on the remote MQe queue manager if you wish, which would need six queue definitions per MQe queue manager. In this script, we use the remote queue manager alias method in order to keep things as simple as possible.

Step 6: Create BRANCH.SALES.QUEUE local queue on CENTRAL00QM

On The *CENTRAL00QM* queue manager, create a normal local queue *BRANCH.SALES.QUEUE*:

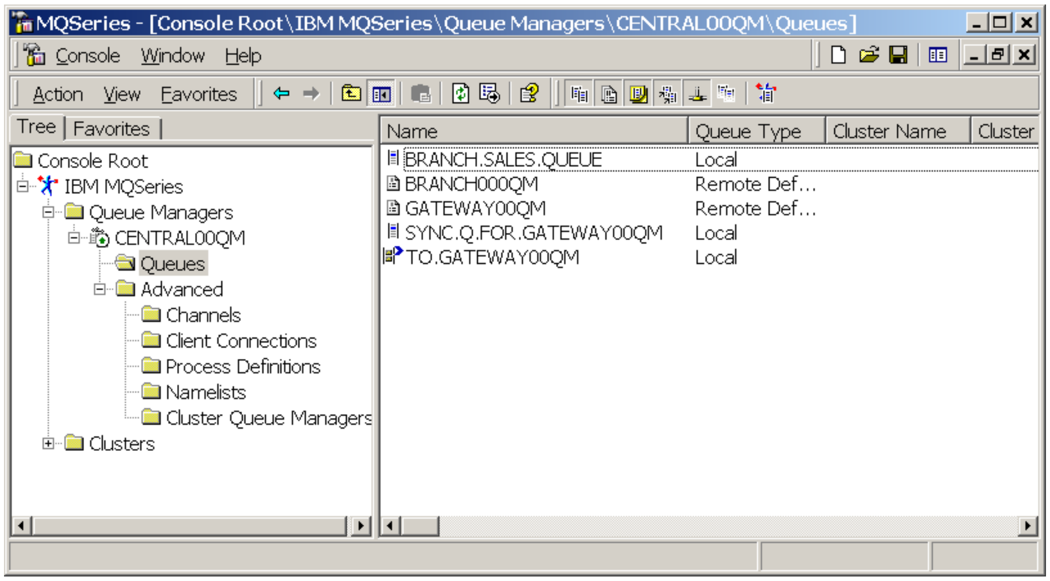
***Queue Name*:**	**BRANCH.SALES.QUEUE**
***Put Messages*:**	**Allowed**
***Get Messages*:**	**Allowed**
***Default Persistence*:**	**Persistent**

Later in this script we will later access this queue from MQe and put test messages to it.

Step 7: Milestone

You should now have a WebSphere MQ queue manager ready for connection to your MQe queue manager. This completes the configuration of the *CENTRAL00QM* WebSphere MQ queue manager.

The WebSphere MQ Explorer should show the following information:



Name	Queue Type	Cluster Name	Cluster
BRANCH.SALES.QUEUE	Local		
BRANCH000QM	Remote Def…		
GATEWAY00QM	Remote Def…		
SYNC.Q.FOR.GATEWAY00QM	Local		
TO.GATEWAY00QM	Local		

Figure 13-3: WebSphere MQ Explorer view

Step 8: Create the *GATEWAY00QM* MQe queue manager

MQe_Explorer can create new local queue managers, which can be subsequently configured to assume the roles of *client*, *server* or *gateway*. Normally when creating such queue managers, the default properties can be taken. First however, check that the MQe_Explorer default options are set such that the necessary local queues will be created. This script requires that the admin-related queues are present, as well as the queue *SYSTEM.DEFAULT.LOCAL.QUEUE*.

Start MQe_Explorer and select the menu item *Tools→Options*. Using the scroll buttons, scroll to the Advanced-1 tab and select it. In the *New configuration – create queues* group box make sure that all four checked boxes are checked. If you have made changes, click the *Apply* button, then in all cases, click *Close*.

The gateway queue manager can now be created.

Using the MQe_Explorer, click on the menu item *File→New→Queue Manager* (or the equivalent toolbar button) and create a new MQe queue manager:

***General tab*:**

***QMgr name*:**	**GATEWAY00QM**
***Description*:**	**Gateway 00 queue manager**

After the queue manager has been created, a comms. listener is required.

Create a comms. listener with the following characteristics and then start it:

***Name*:**	**MyCommsListener1**
***Port*:**	**8082**

Step 9: Create the *BRANCH000QM* MQe queue manager

In a separate instance of MQe_Explorer, create the *BRANCH000QM* MQe queue manager:

***General tab*:**

***QMgr name*:**	**BRANCH000QM**
***Description*:**	**Branch 000 server queue manager**

After the queue manager has been created, its comms. listener is required.

Create a comms. listener with the following characteristics and then start it:

***Name*:**	**MyCommsListener1**
***Port*:**	**8083**

Step 10: Tell the gateway queue manager about the WebSphere MQ *CENTRAL00QM* queue manager

Using the *Gateway00QM* instance – create a new connection (right click on *MQe root\GATEWAY00QM\Connections* and select *New Connection*):

***General tab*:**

***Name*:**	**CENTRAL00QM**
***Local qMgr*:**	**GATEWAY00QM**
***Type*:**	**Alias/MQ**

Step 11: Tell the gateway queue manager about the branch queue manager

Using the *Gateway00QM* instance – create a new connection (right click on *MQe root\GATEWAY00QM\Connections* and select *New Connection*):

***General tab*:**

***Name*:**	**BRANCH000QM**
***Local qMgr*:**	**GATEWAY00QM**
***Type*:**	**Direct**

***Detail tab*:**

***IP address*:**	***The IP address of your machine***
***Port*:**	**8083**

Step 12: Tell the branch queue manager about the gateway queue manager

Using the *BRANCH000QM* instance – create a new connection (right click on *MQe root\BRANCH000QM\Connections* and select *New Connection*):

***General tab*:**

***Name*:**	**GATEWAY00QM**
***Local qMgr*:**	**BRANCH000QM**
***Type*:**	**Direct**

***Detail tab*:**

***IP address*:**	***The IP address of your machine***
***Port*:**	**8082**

Step 13: Tell the branch that it can get to *CENTRAL00QM* via *GATEWAY00QM*

Using the *BRANCH000QM* instance – create a new connection (right click on *MQe root\BRANCH000QM\Connections* and select *New Connection*):

***General tab*:**

***Name*:**	**CENTRAL00QM**
***Local qMgr*:**	**BRANCH000QM**
***Type*:**	**Indirect**

***Detail tab*:**

***Via qMgr*:**	**GATEWAY00QM**

Step 14: Create an bridge object on the gateway MQe queue manager

Using the *Gateway00QM* instance – right click on the *Bridges* node in the tree for the *GATEWAY00QM* queue manager (i.e. *MQe root\GATEWAY00QM\Bridges*) and select *New Bridge*. Using the *New Bridge* panel, create the bridge:

***Name*:**	**MyBridge**
***Local qMgr*:**	**GATEWAY00QM**

Step 15: Create an MQ proxy for the *CENTRAL00QM* queue manager

Note that the name of the proxy object matches the name of the WebSphere MQ queue manager to which it refers.

Using the *Gateway00QM* instance – right click on the *MyBridge* node in the tree for the *GATEWAY00QM* queue manager (i.e. *MQe root\GATEWAY00QM\Bridges\MyBridge*) and select *New Proxy*. Using the *New MQ Proxy* panel, create the proxy:

***Name*:**	**CENTRAL00QM**
***Bridge*:**	**MyBridge**
***Local qMgr*:**	**GATEWAY00QM**

Step 16: Create a client connection

Note that the name of the client connection matches the name of the server connection channel created earlier on the WebSphere MQ queue manager.

Using the *Gateway00QM* instance – right click on the *CENTRAL00QM* proxy node in the tree for the *GATEWAY00QM* queue manager (i.e. *MQe root\GATEWAY00QM\Bridges\MyBridge\CENTRAL00QM*) and select *New Client Conn*. Using the *New Client Connection* panel, create the client connection:

***General tab*:**

***Name*:**	**FOR.GATEWAY00QM**
***MQ proxy*:**	**CENTRAL00QM**
***Bridge*:**	**MyBridge**
***Local qMgr*:**	**GATEWAY00QM**

***Detail tab*:**

***Port*:**	**1414 (matching WebSphere MQ listener).**
***Sync queue*:**	**SYNC.Q.FOR.GATEWAY00QM**

Step 17: Create a listener

This will connect to the transmission queue on WebSphere MQ, pulling messages continuously from that queue and posting them onto the MQe network. Note that one such object is required for each WebSphere MQ transmission queue that is passing messages to the MQe network. Note also that the name of the listener is the same as the transmission queue name on which it acts.

Using the *Gateway00QM* instance – right click on the *FOR.GATEWAY00QM* client connection node in the tree for the *GATEWAY00QM* queue manager (i.e. *MQe root\GATEWAY00QM\Bridges\MyBridge\CENTRAL00QM\FOR.GATEWAY00QM*) and select *New Listener*. Using the *New Listener* panel, create the listener:

***General tab*:**

***Name*:**	**TO.GATEWAY00QM**
***Client conn*:**	**FOR.GATEWAY00QM**
***MQ proxy*:**	**CENTRAL00QM**
***Bridge*:**	**MyBridge**
***Local qMgr*:**	**GATEWAY00QM**

Step 18: Start the bridge, so that message transfer is activated

Using the *Gateway00QM* instance – right click on the *MyBridge* node in the tree for the *GATEWAY00QM* queue manager (i.e. *MQe root\GATEWAY00QM\Bridges\MyBridge*) and select the *Start* action. The state of the bridge will change to *Running*; this can be seen by selecting the *Bridges* node in the tree (*MQe root\GATEWAY00QM\Bridges*); in the right hand pane will be seen the properties of *MyBridge*.

From this time, any messages sent from WebSphere MQ to either of the MQe queue managers should be pulled-down by the running listener.

Step 19: Create a bridge queue that refers to the *BRANCH.SALES.QUEUE* on the WebSphere MQ queue manager

Using the *Gateway00QM* instance – under the *CENTRAL00QM* connection node of the *GATEWAY00QM* queue manager, create a bridge queue, i.e. right click on *MQe root\GATEWAY00QM\Connections\CENTRAL00QM* and select *New Queue*:

***General tab*:**

***Name*:**	**BRANCH.SALES.QUEUE**
***Local qMgr*:**	**GATEWAY00QM**
***Queue qMgr*:**	**CENTRAL00QM**
***Type*:**	**Bridge**

***MQ Bridge tab*:**

***MQ bridge*:**	**MyBridge**
***Proxy*:**	**CENTRAL00QM**
***Client conn*:**	**FOR.GATEWAY00QM**

Step 20: Put a test message from the *GATEWAY00QM* MQe queue manager to the *BRANCH.SALES.QUEUE*

Using the *Gateway00QM* instance – right click on the newly created bridge queue and use the *New Message* menu item:

***Queue mgr*:**	**CENTRAL00QM**
***Queue*:**	**BRANCH.SALES.QUEUE**

You should get the message: "Message sent from 'GATEWAY00QM' … etc." If you get the error: "Unable to send the message – generic MQ error" then check that the properties of the client connection you created – it must have a sync queue which matches that defined on your WebSphere MQ queue manager – in this case *SYNC.QUEUE.FOR.GATEWAY00QM*. Note that if you change this property, you need to stop and then re-start the bridge to make the change take effect.

When you have successfully put a message, look at the message by using the WebSphere MQ Explorer and browsing the *BRANCH.SALES.QUEUE*. Now view the same queue from MQe_Explorer by refreshing the *BRANCH.SALES.QUEUE*. Delete the message using the WebSphere MQ Explorer if you wish.

Step 21: Tell the BRANCH000QM about the BRANCH.SALES.QUEUE on CENTRAL00QM

Using the *BRANCH000QM* instance – add a sync. proxy queue to the *BRANCH001QM* queue manager to refer to the *BRANCH.SALES.QUEUE* on the *CENTRAL00QM* queue manager. You can right click on *MQe root\BRANCH000QM\Connections\CENTRAL00QM* and select *New Queue*.

***Name*:**	**BRANCH.SALES.QUEUE**
***Local qMgr*:**	**BRANCH000QM**
***Queue qMgr*:**	**CENTRAL00QM**
***Type*:**	**Sync. proxy**

Step 22: Put a test message to the *BRANCH.SALES.QUEUE* from the *BRANCH000QM* queue manager

Using the *BRANCH000QM* instance – right click on the proxy queue for *BRANCH.SALES.QUEUE* on the *CENTRAL00QM* queue manager (i.e. *MQe root\BRANCH000QM\Connections\CENTRAL00QM\BRANCH.SALES.QUEUE*) and select *New Message*. Use the defaults to send the message.

You should now be able to view the message, which is sitting on the WebSphere MQ queue, using MQe's synchronous queue browse operation, from *BRANCH000QM's* MQe_Explorer.

You can also view the test message from the WebSphere MQ Explorer, as before.

Step 23: Put a message from WebSphere MQ to the gateway queue manager

Put a number of test messages from the WebSphere MQ queue manager to the MQe branch queue manager. An example program for this purpose (*PutFromMQ*) is shipped with MQe; in a default installation it is available in the *Program Files\MQe\Java\examples\mqbridge\application* directory. Invoke this program from a DOS prompt using the string:

```
jview examples.mqbridge.application.PutFromMQ CENTRAL00QM
    SYSTEM.DEFAULT.LOCAL.QUEUE GATEWAY00QM 1 1 hello
```

The *PutFromMQ* program connects to the WebSphere MQS queue manager and sends a message to the *SYSTEM.DEFAULT.LOCAL.QUEUE* queue on *GATEWAY00QM*. The remote queue manager alias on *CENTRAL00QM* directs the message to the *FOR.GATEWAY00QM* transmission queue. The MQ transmission queue listener on the *GATEWAY00QM's* bridge configuration (called *FOR.GATEWAY00QM)* pulls the message down from the transmission queue, and posts it to the MQe network. The *GATEWAY00QM* queue manager directs the message to its local *SYSTEM.DEFAULT.LOCAL.QUEUE* queue.

Use the MQe_Explorer to view the test messages.

Step 24: Send test messages from WebSphere MQ to the *BRANCH000QM* queue manager

Use the *PutFromMQ* program as above, this time with the command string:

```
jview examples.mqbridge.application.PutFromMQ CENTRAL00QM
    SYSTEM.DEFAULT.LOCAL.QUEUE BRANCH000QM 1 1 hello
```

The route taken by these messages is the same as that used by the previous messages, except that the *GATEWAY00QM* queue manager pushes the messages to the *BRANCH000QM* queue manager, which then deposits them on its local *SYSTEM.DEFAULT.LOCAL.QUEUE* queue.

14 Programming MQe_Explorer applications

MQe_Explorer can be used to launch applications against the local queue manager. In this scenario typically MQe_Explorer is used to create and run the queue manager, then additional applications (i.e. classes) are loaded into the same JVM. The MQe_Explorer and the applications share the same queue manager. These applications can be stopped and started through their own graphical user interfaces; meanwhile the queue manager and MQe_Explorer continue to execute. The MQe_Explorer can be used to manage the various MQe objects and display their status, whilst its trace and console capabilities can be used to debug the applications during development.

Launching classes in this way is described in the section *Load* on page 105. The application *examples.mqe_explorer.PerfTest* can be run as a sample to explore these facilities. In this chapter more information is provided on writing such applications.

14.1 Windows application shell

Applications that require a Windows user interface, implemented through the Windows Foundation Classes, can be created using the Microsoft Windows J++ development environment. Such applications however will only be able to execute on a Windows platform and will require a Microsoft JVM.

The basic shell of such an application is shown in *Figure 14-1: Basic Windows application shell* on page 159. The import statements include the MQe and Windows classes needed for simple applications. The application *MyApp* is shown as being a subclass of *com.ms.wfc.ui.Form* and therefore presents a window on the screen – typically this form will have buttons, menus and other graphical interface elements defined (not shown in the example). The application is initiated by a call to the static *main()* method – and parameters can be passed as strings in that call. An instance of the *MyApp* class is created and this is passed to the *run()* method of the windows *com.ms.wfc.app.Application* class; starting the application in this way initiates the messaging environment needed by Windows such that notification of interesting events, such as mouse clicks etc. will be received.

The constructor has an obligatory *initForm()* call needed by Windows to initialize the form; following that it may be useful to get addressability to the MQe queue manager. From this point, the application logic can be programmed.

The *close()* method shows the minimum processing needed to close the application. The queue manager itself should not be closed since MQe_Explorer will still be using it, likewise the JVM should not be destroyed.

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ibm.mqe.*;
import com.ibm.mqe.administration.*;

public class MyApp extends Form
{
  MQeQueueManager         qMgr;    //local queue manager

  public MyApp()
  {
          // Required for Visual J++ Form Designer support
```

```
                initForm();
                //get addressability to the local queue manager
                qMgr = MQeQueueManager.getReference(null);
                //application continues ……
        }

        //close application
        private void close()
        {
                //dispose of the form and close the application
                this.dispose();
                Application.exitThread();
        }

        //dispose of all resources
        public void dispose()
        {
                super.dispose();
                components.dispose();
                return;
        }

        //main entry point for the application.
        //  - args: array of parameters passed to the application via the command line.
        public static void main(String args[])
        {
                try                             //errors may occur if exit before the form is loaded
                {
                        Application.run(new MyApp());
                }
                catch (Throwable x) {}
        }

      }
```

Figure 14-1: Basic Windows application shell

14.2 Non-windows application shell

Applications without a user-interface, or applications that use the cross-platform Java classes can also be loaded and run by MQe_Explorer. The basic shell is similar and *Figure 14-2: Standard application shell* shows the code needed in the simplest case.

```
import com.ibm.mqe.*;
import com.ibm.mqe.administration.*;

public class MyApp
{
  MQeQueueManager         qMgr;    //local queue manager

  public MyApp()
  {
            //get addressability to the local queue manager
            qMgr = MQeQueueManager.getReference(null);
            //application continues ……

            // …… application finishes
  }

  //main entry point for the application.
  //  - args: array of parameters passed to the application via the command line.
  public static void main(String args[])
  {
            MyApp myApp = new MyApp();
  }

}
```

Figure 14-2: Standard application shell

14.3 Handling console interrupts

The MQe_Explorer console allows threads to be interrupted; this is provided so that applications can be terminated from the console. However, this supports requires the application to be written such that these termination are possible. Very little code is required, as is shown in *Figure 14-3: Console interrupt programming*.

Periodic checks are required to test if the application thread has been interrupted; if so, the application must be closed, (as in the *close()* method in Figure 14-1: Basic Windows application shell) before the thread itself terminates.

```
//application work
while (running)
{
          //application work …
          //…………

          if (Thread.currentThread().isInterrupted())
          {
                    this.close();
                    return;
          }
}
```

Figure 14-3: Console interrupt programming

14.4 Debug

The standard MQe trace facilities can be used for debug; the output is available in the trace window or can be logged. Applications can also access the trace window directly as a *java.io.PrintStream* object; for more details see *Class com.ibm.mqe.mqe_explorer.MQeExplTrace* on page 172.

15 Appendix A: Performance tool

A performance tool is included with MQe_Explorer, not just as an example of a program that can be loaded concurrently into the same JVM and able to share the same local queue manager, but also to support basic performance measurements of MQe functions.

The following operations are supported:

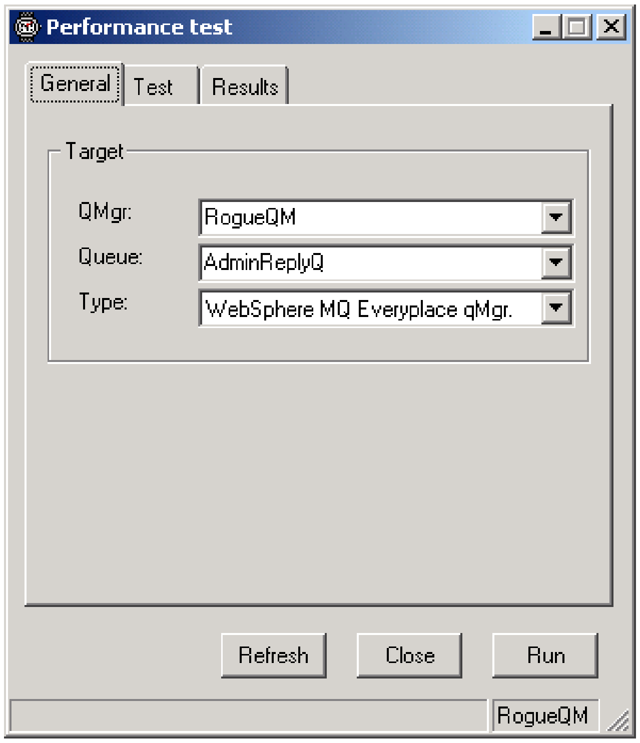
- Put – in a series of put/get pairs (both one- and two-phase operation)
- Put – in a series of consecutive puts
- Get – in a series of put/get pairs (both one- and two-phase operation)
- Put/get pairs (both one- and two-phase operation)
- Browse (retrieve full message or just UID; without locking or lock/unlock pairs)

The target queue may optionally cleared before the test and/or be pre-loaded with messages and directed at be chosen from any local or remote queue manager. Messages are either of the message class *com.ibm.mqe.MQeMsgObject* with a single field containing an array of bytes, of the designated payload size. Alternatively the message class can be *com.ibm.mqe.mqemqmessage.MQeMQMsgObject*. All bytes are randomly generated and all messages are unique. Intentionally, no data compression is possible.

Multiple instances of the performance tool can be run simultaneously; however when using the same target queue care must be taken that simultaneously tests do not conflict. For example, simultaneous use of the clear queue option may destroy a message that another instance is expecting to retrieve.

15.1 Menus and operations

The main window is as follows, showing the *General* tab:



QMgr: RogueQM
Queue: AdminReplyQ
Type: WebSphere MQ Everyplace qMgr.
RogueQM

Figure 15-1: The performance tool – General tab

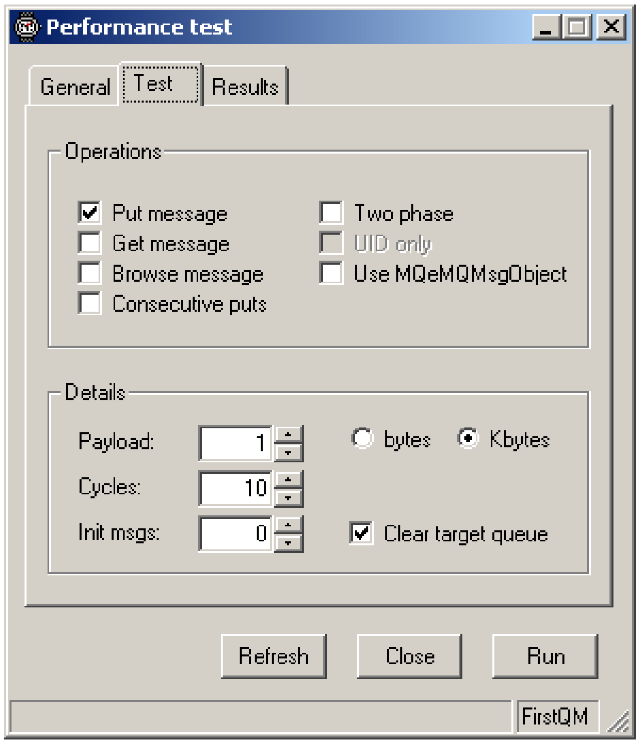
The status bar at the bottom of the window is used to display information messages. On the right hand side is shown the name of the local queue manager.

The following input fields are present on the *General* tab:

- *General* tab:
  - *Queue manager* – the name of the queue manager hosting the target queue to be used.
  - *Queue* – the name of the queue to be used.
  - *Description* – any UNICODE string.
  - *Type* – the type of queue manager hosting the target queue; one of:
    - WebSphere MQ Everyplace qMgr.
    - WebSphere MQ qMgr.

    If WebSphere MQ qMgr. is selected then the performance tool will not attempt to clear the target queue before a test is run; nor can the number of initial messages be set.

The *Test* tab has the following appearance:



Operations

[✓ Put message]
Get message
Browse message
Consecutive puts
Two phase
UID only
Use MQeMQMsgObject

Details

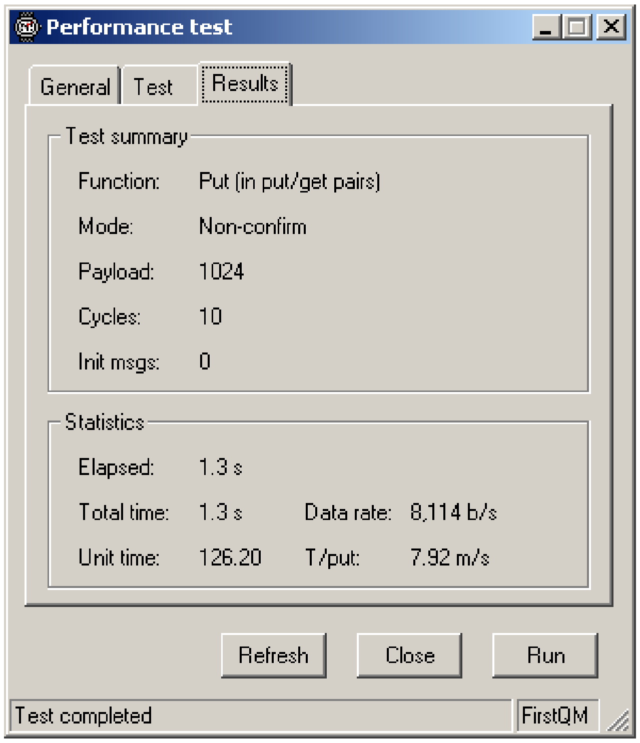
Payload: 1 bytes [✓ Kbytes]
Cycles: 10
Init msgs: 0
[✓ Clear target queue]

Figure 15-2: The performance tool – Test tab

The following input fields are present on the *Test* tab:

- *Test* tab:
  - *Put message* – checking this box causes the put time in a put/get pair to be measured.
  - *Get message* – checking this box causes the get time in a put/get pair to be measured. No filter is present in the get operation.
  - *Browse message* – checking this box causes the browse time to be measured. No filter is present in the browse operation – all available messages on the target queue will be browsed; no messages are removed.
  - *Consecutive puts* – checking this box causes the put time in a series of consecutive puts to be measured.
  - *Two phase* – checking this box causes two-phase operations to be used. For all puts/gets, a two-step put and/or get is used, with non-zero confirm ids; otherwise a one-step operation is used with zero confirm ids. For browse, a browse & lock operation is used, followed by an unlock; otherwise a single browse without lock is used.
  - *UID only* – only applicable to browse; if checked only the UIDs of messages are retrieved in the browse.
  - *Use MQeMQMsgObject* – if checked, the test messages are of class *com.ibm.mqe.mqemqmessage.MQeMQMsgObject*; otherwise they are of class *com.ibm.mqe.MQeMsgObject*.
  - *Payload* – the size of the message payload in either bytes of Kbytes.
  - *Cycles* – the number of times the test is run; the results are averaged across the cycles.
  - *Initial messages* – the number of messages to be placed on the target queue before the test is run – these messages have identical characteristics to the test messages themselves. In this way tests can be run against loaded queues. This parameter is especially important for browse since it controls the number of messages browsed.
  - *Clear target queue* – if checked, the target queue is cleared before the queue is loaded with the initial number of messages. If unchecked, the target queue is not cleared and no initial messages are loaded.

The *Results* tab has the following appearance:


Performance test

Test summary

Function:	Put (in put/get pairs)
Mode:	Non-confirm
Payload:	1024
Cycles:	10
Init msgs:	0

Statistics

Elapsed:	1.3 s		
Total time:	1.3 s	Data rate:	8,114 b/s
Unit time:	126.20	T/put:	7.92 m/s

Test completed | FirstQM

Figure 15-3: The performance tool – Results tab

The following output fields are present on the *Results* tab after the test has been run; changing the input data clears the results:

- *Results* tab:
    - *Function* – the operation tested.
    - *Mode* – details of the way the operation was implemented.
    - *Payload* – the size of the message payload in bytes.
    - *Cycles* – the number of times the test was repeated.
    - *Initial messages* – the number of messages initially loaded on the target queue.
    - *Elapsed* – the elapsed time of the test in seconds (including operations that were not part of the test itself).
    - *Total time* – the total time in seconds for all cycles of the operation to be tested.
    - *Data rate* – the bytes per second measured for the test operation.
    - *Unit time* – the average time in milliseconds for one cycle of the test operation.
    - *Throughput* – the number of messages per second for the test operation.

The *Close* button terminates the performance test application. The *Refresh* button reloads the list boxes with the currently available queue manager and queue names. The *Run* button starts the test. A running test can be aborted by interrupting the class (see *Console* on page 103 for more details); in this case the instance of the tool will also be closed.

16 Appendix B: Reference section

16.1 Icons and buttons

MQe_Explorer uses the following icons and buttons (where two icons are shown the first represents the unselected icon, the second the selected icon).

MQe object icons

Bridge-related icons

Bridges folder (represents the Bridges object)

Bridge

MQ proxy

Client connection

Listener

Communications listeners

Comms. listener

Connections

Alias name of a remote connection to an MQe queue manager, or an alias of an alias-only connection

Alias name of a remote connection to an MQe queue manager (direct or indirect)

Remote connection to an MQ queue manager, or an alias-only connection

Remote connection to an MQe queue manager (direct or indirect)

Fields

ASCII field

Boolean field

Byte field

Double field

Fields field

Float field

Integer field

Long field

Short field

UNICODE field

Object, unknown or untyped field

Miscellaneous

Admin message (with unknown admin id)

Admin request message

Admin request detail

Admin response message (without errors)

Admin response message (with errors)

Admin response detail

WTLS mini-certificate

Folder

Message

MQe root

Queue managers

Queue manager (responding)

Queue manager (not responding)

Queues

Admin queue

Alias name of an admin queue

Alias name of an application queue

Alias name of a queue manager

Alias name of a synchronous or asynchronous proxy queue

Bridge queue

Forward queue

Home server queue

Application queue

Synchronous or asynchronous proxy queue

Threads

Class loaded by user

Main thread

MQe_Explorer thread

Other thread

Buttons

Toolbar – console window

	Auto refresh (≡ *View→Refresh*
	Close file (≡ *File→Close*)
↓	Decrease priority (≡ *Action→Decrease Priority*)
↑	Increase priority (≡ *Action→Increase Priority*)
	Interrupt (≡ *Action→Interrupt*)
	Load a class (≡ *Tools→Load*)

Toolbar – main window

	Close current queue manager (≡ *File→Close)*
	Copy (≡ *Edit→Copy)*
	Create a new queue manager (≡ *File→New Queue Mgr.)*
	Cut (≡ *Edit→Cut)*
	Hide empty columns (≡ *View→Hide Empty)*
	Open an existing queue manager (≡ *File→Open)*
	Paste (≡ *Edit→Paste)*
	Ping (≡ *Action→Ping)*
	Refresh selected object (≡ *View→Refresh)*
▶	Start (≡ *Action→Start)*
■	Stop (≡ *Action→Stop)*
	Translate coded admin field names (≡ *View→Translate Admin)*
	Trigger sending of messages (≡ *Action→Trigger Local)*
	Select parent of the currently selected object
	Use multiple rows for aliases (≡ *View→Use Multiple)*

Toolbar – offline admin window

Close file (≡ *File→Close*)

Delete object (≡ *Edit→Delete*)

Refresh selected object (≡ *View→Refresh*)

Show detail (≡ *View→Detail*)

Show offline request messages (≡ *View→Requests*)

Show offline reply messages (≡ *View→Replies*)

Show other messages (≡ *View→Others*)

Toolbar – trace window

Log trace events a log file (≡ *File→Log Trace Events)*

Copy trace display to a file (≡ *File→Save Display As…)*

Cut selected text to the clipboard (≡ *Edit→Cut)*

Copy selected text to the clipboard (≡ *Edit→Copy)*

Paste text from the clipboard (≡ *Edit→Paste)*

Find text string (≡ *Edit→Find)*

Change font formatting of selected text (≡ *Format→Font)*

Clear display (≡*Edit →Clear)*

Stop/start display of trace events (≡ *View→Display Trace)*

Trace default events (≡ *Action→Default)*

Trace information events (≡ *Action→Information)*

Trace warning events (≡ *Action→Warning)*

Trace error events (≡*Action →Error)*

Trace exception events (≡ *Action→Exception)*

Trace all events (≡ *Action→All)*

16.2 Initialization file sections

MQe_Explorer uses additional information in initialization (*.ini*) files beyond that described in the MQe product publications.

[QueueManager] section

The following entry will be present in an initialization file created by MQe_Explorer:

(ascii)QueueManager=<*queue manager class*>

The <*queue manager class*> identifies the queue manager class that MQe_Explorer will instantiate on starting the queue manager. If the entry is not present, a class that is mapped to the alias "QueueManager" will be used.

The following entries will be actioned if present in the configuration file of an unconfigured queue manager. After configuration, any such entry(ies) will be deleted.

(ascii)ChAttrRule=<*channel attribute rule*>

The <*channel attribute rule*> value will be used to set the queue manager channel attribute rule property. If the entry is not present, a default value will be used.

(long)ChTimeout=<*channel timeout*>

The <*channel timeout*> value will be used to set the queue manager channel timeout property. If the entry is not present, a default value will be used.

(unicode)Description=<*description*>

The <*description*> value will be used to set the queue manager description property. If the entry is not present, a default value will be used.

(ascii)MessageStore=<*message store*>

The <*message store*> value will be used to set the queue manager default message store property. If the entry is not present, a default value will be used.

(ascii)QMgrRule=<*queue manager rule*>

The <*queue manager rule*> value will be used to set the queue manager rule property. If the entry is not present, a default value will be used.

(ascii)QueueAdapter=<*queue adapter*>

The <*queue adapter*> value will be used to set the queue manager default queue adapter property. If the entry is not present, a default value will be used.

(ascii)QueuePath=<*queue path*>

The <*queue path*> value will be used to set the queue manager default queue path property. If the entry is not present, a default value will be used.

[Registry] section

The following entry will be present in an initialization file created by MQe_Explorer:

(boolean)QueueRegistry=<*enabled*>

The parameter <*enabled*> has the value 'true' or 'false' depending on whether queue registries are to be allowed. If the entry is not present, the value 'true' is assumed.

16.3 MQe_Explorer public methods

The following methods give additional control over MQe_Explorer operation:

Class: com.ibm.mqe.mqe_explorer.MQeExplProperty

The following methods provide control over whether error messages will be shown when MQe_Explorer detects errors whilst asynchronously accessing MQe objects:

```
public static boolean getShowClassNotFoundError()
public static void setShowClassNotFoundError(boolean show)
public static boolean getShowExceptionError()
public static void setShowExceptionError(boolean show)
public static boolean getShowIllegalAccessError()
public static void setShowIllegalAccessError(boolean show)
public static boolean getShowInputOutputError()
public static void setShowInputOutputError(boolean show)
public static boolean getShowLinkageError()
public static void setShowLinkageError(boolean show)
public static boolean getShowUnexpectedError()
public static void setShowUnexpectedError(boolean show)
```

Class com.ibm.mqe.mqe_explorer.MQeExplTrace

The following method provides access to the trace window as a *PrintStream* object, allowing debugging information to be directly displayed:

```
public static PrintStream getTracePrintStream()
```

If the trace window has not been displayed (through *View→Trace*) then this method returns *null*. An example of usage is:

```
PrintStream traceOut = MQeExplTrace.getTracePrintStream();
If (traceOut != null)
{
        //write output to trace window
        traceOut.println("My debug information");
}
```

16.4 Accessibility features

The following general keystroke aids to navigation are supported:

Keystroke(s)	Place	Function
↑	Main window: list pane Console window Offline admin window	Go up one list view item
↑	All windows: menu	Go up one menu item
↑	Main window: tree pane	Go up one node in the tree
↓	Main window: list pane Console window Offline admin window	Go down one list view item If nothing is selected, select the first item
↓	All windows: menu	Go down one menu item
↓	Main window: tree pane	Go down one node in the tree
←	Main window: list pane Console window Offline admin window	Scroll left
←	All windows: menu	Go left one menu item
←	Main window: tree pane	Shrink current node in the tree
→	Main window: list pane Console window Offline admin window	Scroll right
→	All windows: menu	Go right one menu item
→	Main window: tree pane	Expand the current node in the tree
Alt	All windows	Move focus to the menu bar *(then press underlined key to activate menu item)*
Alt+space	All windows	Control box
Alt+tab	All	Cycle through windows
AltGr+tab	Trace Property pages	Cycle through tabs
Ctrl +tab	Trace Property pages	Cycle through tabs
Enter	Property pages: buttons	Click selected button
Enter	All windows: menus	Click selected menu item
Esc	Property pages	Cancel
F6	Main window	Cycle through panes
F10	All windows	Move focus to the menu bar *(then press underlined key to activate menu item)*
Shift+Tab	Property pages	Cycle backwards through input fields & buttons
Tab	Property pages	Cycle forwards through input fields & buttons

Figure 16-1: General keystroke aids to navigation

Note: In the above table Property pages is used to also include: Class load, File send/receive, Options, Status, and Test message

The following keystroke aids to navigation are supported in the Trace window text area:

Keystroke(s)	**Function**
↑	Move insertion point up a line
↓	Move insertion point down a line
←	Move insertion point one character to the left
→	Move insertion point one character to the right
Ctrl+↑	Move insertion point one line up
Ctrl+↓	Move insertion point one line down
Ctrl+←	Move insertion point one character to the left
Ctrl+→	Move insertion point one character to the right
Ctrl+Shift+↑	Extend selection to the beginning of the paragraph
Ctrl+Shift+↓	Extend selection to the end of the paragraph
Ctrl+Shift+←	Extend selection to the beginning of the word
Ctrl+Shift+→	Extend selection to the end of the word
Ctrl+A	Extend selection to include all text
Ctrl+End	Move insertion point to the end
Ctrl+Home	Move insertion point to the beginning
End	Move insertion point to the end of the line
Home	Move insertion point to the beginning of the line
Shift+↑	Extend selection one line up
Shift+↓	Extend selection one line down
Shift+←	Extend selection one character to the left
Shift+→	Extend selection one character to the right
Ctrl+Shift+Home	Extend selection to the beginning
Ctrl+Shift+End	Extend selection to the end
Shift+End	Extend selection to the end of the line
Shift+Home	Extend selection to the beginning of the line
Shift+PgDn	Extend selection to end
Shift+PgUp	Extend selection to beginning
Tab	If no insertion point, display insertion point at end If insertion point present, insert a tab

Figure 16-2: Restricted keystroke aids to navigation

Shortcuts

The following keystroke shortcuts are supported:

Keystroke(s)	Window	Name	Menu equivalent	Function
Ctrl+A	Main Offline Admin Trace	Select All	Edit→Select All	Select all
Ctrl+C	Main Trace	Copy	Edit→Copy	Copy to clipboard
Ctrl+D	Console	Decrease priority	Action→Decrease priority	Decrease thread priority
Ctrl+D	Main	Demo mode	Tools→Demo Mode	Demo Mode
Ctrl+Delete	Main Offline Admin Trace	Clear	Edit→Clear	Clear
Ctrl+F	Main	Offline admin	View→Offline Admin	View offline admin window
Ctrl+F	Trace	Find	Edit→Find	Find string/word
Ctrl+G	Main	Ping	Action→Ping	Ping queue manager
Ctrl+I	Console	Increase priority	Action→Increase priority	Increase thread priority
Ctrl+L	Main Console	Load	Tools→Load	Load a class
Ctrl+N	Main	Options	Tools→Options	View options window
Ctrl+O	Main	Open	File→Open	Open an initialization file
Ctrl+P	Main	Stop	Action→Stop	Stop selected object
Ctrl+R	Main	Renew cert.	Action→Renew Credentials	Renew qMgr mini-cert.
Ctrl+R	Offline admin	Resend	File→Resend	Resend message
Ctrl+S	Main	Start	Action→Start	Start selected object
Ctrl+T	Console	Interrupt	Action→Interrupt	Interrupt selected thread
Ctrl+T	Main	Trigger	Action→Trigger Local	Trigger local queue manager
Ctrl+X	Main Trace	Cut	Edit→Cut	Cut to clipboard
Ctrl+V	Main Trace	Paste	Edit→Paste	Paste from clipboard
Ctrl+Z	Main	Console	View→Console	View console window
Delete	Main Offline Admin Trace	Delete	Edit→Delete	Delete selected
F4	Main	Properties	File→Properties	Display object properties
F5	Console Main Offline Admin	Refresh	View→Refresh	Refresh selected object

Figure 16-3: Keystroke shortcuts

17 Appendix C: MQeFileTransferMsg class

The *MQeFileTransferMsg* class is contained in the *com.ibm.mqe.mqe_explorer* package and is used to represent a file (or a segment of a file) being transferred as a message. The class is a subclass of *MQeMsgObject* and adds the following function:

The ability to carry ancillary data associated with the file, such as description, source file specification, segment identification etc.

A validation check on the integrity of the data. When data is stored in the message object a digest is calculated and carried with the message; when retrieval of the data is attempted a new digest is calculated – data is released only if the digests match.

Every instance of an *MQeFileTransferMsg* object represents a segment of a file; an associated file may be mapped into one (or more) such segments. The collection of instances representing one such file are uniquely identified by the combination of:

A unique id (supplied by the programmer).

The originating queue manager name, i.e. that queue manager that instantiated the *MQeFileTransferMsg* objects (and available through the method call *MQeMsgObject.getOriginQMgr*).

Each segment instance carries all the common information associated with the file, such as description, source file specification, etc. Each segment instance uniquely carries its segment identity and the data associated with that segment.

When data is stored in a segment via the *MQeFileTransferMsg.setData* method, a 20 byte SHA digest is generated and carried in the object. When data retrieval is attempted via the *MQeFileTransfer.getData* method, a new digest is calculated; if the digest matches the original, the data is returned, otherwise *null*.

This class has no dependencies on any other classes in the *com.ibm.mqe.mqe_explorer* package.

17.1 Constructor

Syntax

public MQeFileTransferMsg() throws Exception

Description

Creates a new MQeFileTransferMsg object with default values set as follows:

Code:	0
Data:	new byte[0]
Description:	""
File path:	""
Priority:	5
Segment:	0
Segments:	1
Unique id:	0

For details of these properties see the corresponding *get* methods.

Parameters

None.

Return values

None.

Exceptions

Passed from the superclass.

17.2 Methods

MQeFileTransferMsg.getCode

Syntax

public int getCode() throws Exception

Description

Returns the *code* property. The code values are undefined in release 2.0 but may be used to reflect the encoding of the data (e.g. code page) associated with the file.

Parameters

None.

Return values

code – code property.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.getData

Syntax

public byte[] getData() throws Exception

Description

Returns the data associated with the segment. If no data has been set then a new byte[0] is returned; if the data has been corrupted *null* is returned.

Parameters

None.

Return values

data – the data associated with the segment.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.getDescription

Syntax

public String getDescription() throws Exception

Description

Returns a description associated with the file. If no description has been set then a zero length String is returned.

Parameters

None.

Return values

description – the description associated with the file.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.getFilePath

Syntax

public String getFilePath() throws Exception

Description

Returns a file specification associated with the file – this is expected to be the source location of the file, but can be used to indicate the target location.

Parameters

None.

Return values

path – the file specification associated with the file.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.getPriority

Syntax

public byte getPriority() throws Exception

Description

Returns the priority associated with the message; this priority is used by MQe to sequence transmission and retrieval order in queues. It is expected that all segments relating to the file will be given the same priority.

Parameters

None.

Return values

priority – MQe message priority.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.getSegment

Syntax

public int getSegment() throws Exception

Description

Returns the segment identity. Segments associated with the same file must be numbered sequentially, starting from zero.

Parameters

None.

Return values

segment – the segment identity.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.getSegments

Syntax

public int getSegments() throws Exception

Description

Returns the number of segments associated with the file.

Parameters

None.

Return values

segments – the number of segments associated with the file.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.getUniqueId

Syntax

public long getUniqueId() throws Exception

Description

Returns the unique id associated with the file, chosen by the programmer. An easy choice would be to use the current time, from *System.currentTimeMillis*. The combination of this with the originating queue manager name, which is held within the object itself by virtue of the superclass, will ensure that segments associated with a single file share a key that is unique across an MQe network.

Parameters

None.

Return values

id – unique id associated with the file.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setCode

Syntax

public void setCode(int *code*) throws Exception

Description

Sets the *code* property. There is no value checking.

Parameters

code – code property.

Return values

None.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setData

Syntax

public void setData(byte[] *data*) throws Exception

Description

Sets the data associated with the segment. If *null* is passed the data is set to new byte[0].

Parameters

data – the data associated with the segment.

Return values

None.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setDescription

Syntax

public void setDescription(String *descr*) throws Exception

Description

Sets the description associated with the file. If null is passed the description is set to a zero length String. The full range of UNICODE characters is permitted.

Parameters

descr – the description associated with the file.

Return values

None.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setFilePath

Syntax

public void setFilePath(String *path*) throws Exception

Description

Sets the file specification associated with the file. If null is passed the file specification is set to a zero length String. The characters in the specification are limited to the ASCII subset.

Parameters

path – the file specification associated with the file.

Return values

None.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setPriority

Syntax

public void setPriority(byte *priority*) throws Exception

Description

Sets the priority associated with the message. The priority must be in the range 0 – 9; values outside this will be set to the max. or min. as appropriate.

Parameters

priority – the MQe priority associated with the message.

Return values

None.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setSegment

Syntax

public void setSetSegment(int *segment*) throws Exception

Description

Sets the segment identity. The identity must be >= 0; values outside this range will be set to zero.

Parameters

segment – the segment identity.

Return values

None.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setSegments

Syntax

public void setSegments(int *segments*) throws Exception

Description

Sets the number of segments associated with the file. The number of segments must be >= 1; values outside this range will be set to 1.

Parameters

Return values

None.

Exceptions

Passed from the superclass.

MQeFileTransferMsg.setUniqueId

Syntax

public void setUniqueId(long *id*) throws Exception

Description

Sets the unique id associated with the file.

Parameters

id – unique id associated with the file.

Return values

None.

Exceptions

Passed from the superclass.

17.3 Usage notes

Sending files

Files can be sent either in a single message or segmented across multiple messages. If segmented, a segment size is typically used to partition the bytes across messages, with equal numbers of bytes in the first segments and the remaining bytes in the last segment. All the segments are expected to hold all the common file data, such as description etc.; segments associated with a single file should differ only in their segment identifier and data content. The unique id property must be set to a common value for all segments associated with a single file.

In many cases it will be appropriate for file transfer messages to be sent to a dedicated destination queue, but this is not required. If a dedicated queue is not being used, and message class is not to be used to distinguish the messages on arrival, then it may be necessary to add an additional field into all file transfer messages to identify them as such – and of a type that can be used as an MQe filter in *browse* or *getMessage* operations.

If compression and security is required, this can be achieved by using MQe security attributes on that target queue.

If the messages are to be sent to a WebSphere MQ queue then the contents must flow across the bridge in an MQe gateway; this will require appropriate message transformation rules to be set at the gateway.

Receiving files

There are many ways in which files can be retrieved. If a dedicated arrival queue is used then it is known that all messages are expected to contain files. If not, then the messages can be distinguished on message class or through an additional unique identifier. When one (arbitrary) message in a segmented file transfer is received, the other associated messages can be received through the unique id contained in each associated message. It may be necessary to wait for all such messages to arrive. Missing messages are obvious. The integrity of the data is established by reconstructing the original byte array representing the file; any message returning null data is corrupt.

18 Appendix D: Notices

Trademarks

19 Appendix E: International Program License Agreement

Part 1 - General Terms

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE PROGRAM. IBM WILL LICENSE THE PROGRAM TO YOU ONLY IF YOU FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY USING THE PROGRAM YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE UNUSED PROGRAM TO THE PARTY (EITHER IBM OR ITS RESELLER) FROM WHOM YOU ACQUIRED IT TO RECEIVE A REFUND OF THE AMOUNT YOU PAID.

The Program is owned by International Business Machines Corporation or one of its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not sold.

The term "Program" means the original program and all whole or partial copies of it. A Program consists of machine-readable instructions, its components, data, audio-visual content (such as images, text, recordings, or pictures), and related licensed materials.

This Agreement includes Part 1 - General Terms, Part 2 - Country-unique Terms, and "License Information" and is the complete agreement regarding the use of this Program, and replaces any prior oral or written communications between you and IBM. The terms of Part 2 and License Information may replace or modify those of Part 1.

1. License

Use of the Program

IBM grants you a nonexclusive license to use the Program.

You may 1) use the Program to the extent of authorizations you have acquired and 2) make and install copies to support the level of use authorized, providing you reproduce the copyright notice and any other legends of ownership on each copy, or partial copy, of the Program.

If you acquire this Program as a program upgrade, your authorization to use the Program from which you upgraded is terminated.

You will ensure that anyone who uses the Program does so only in compliance with the terms of this Agreement.

You may not 1) use, copy, modify, or distribute the Program except as provided in this Agreement; 2) reverse assemble, reverse compile, or otherwise translate the Program except as specifically permitted by law without the possibility of contractual waiver; or 3) sublicense, rent, or lease the Program.

Transfer of Rights and Obligations

You may transfer all your license rights and obligations under a Proof of Entitlement for the Program to another party by transferring the Proof of Entitlement and a copy of this Agreement and all documentation. The transfer of your license rights and obligations terminates your authorization to use the Program under the Proof of Entitlement.

2. Proof of Entitlement

The Proof of Entitlement for this Program is evidence of your authorization to use this Program and of your eligibility for warranty services, future upgrade program prices (if announced), and potential special or promotional opportunities.

3. Charges and Taxes

IBM defines use for the Program for charging purposes and specifies it in the Proof of Entitlement. Charges are based on extent of use authorized. If you wish to increase the extent of use, notify IBM or its reseller and pay any applicable charges. IBM does not give refunds or credits for charges already due or paid.

If any authority imposes a duty, tax, levy or fee, excluding those based on IBM's net income, upon the Program supplied by IBM under this Agreement, then you agree to pay that amount as IBM specifies or supply exemption documentation.

4. Limited Warranty

IBM warrants that when the Program is used in the specified operating environment it will conform to its specifications. IBM does not warrant uninterrupted or error-free operation of the Program or that we will correct all Program defects. You are responsible for the results obtained from the use of the Program. The warranty period for the Program expires when its Program services are no longer available. The License Information specifies the duration of Program services.

During the warranty period warranty service is provided without charge for the unmodified portion of the Program through defect-related Program services. Program services are available for at least one year following the Program's general availability. Therefore, the duration of warranty service depends on when you obtain your license. If the Program does not function as warranted during the first year after you obtain your license and IBM is unable to resolve the problem by providing a correction, restriction, or bypass, you may return the Program to the party (either IBM or its reseller) from whom you acquired it and receive a refund in the amount you paid for it. To be eligible, you must have acquired the Program while Program services (regardless of the remaining duration) were available for it.

THESE WARRANTIES ARE YOUR EXCLUSIVE WARRANTIES AND REPLACE ALL OTHER WARRANTIES OR CONDITIONS, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

These warranties give you specific legal rights, and you may also have other rights which vary from jurisdiction to jurisdiction. Some jurisdictions do not allow the exclusion or limitation of implied warranties, so the above exclusion or limitation may not apply to you. In that event such warranties are limited in duration to the warranty period. No warranties apply after that period.

5. Limitation of Liability

Circumstances may arise where, because of a default on IBM's part or other liability, you are entitled to recover damages from IBM. In each such instance, regardless of the basis on which you may be entitled to claim damages from IBM, (including fundamental breach, negligence, misrepresentation, or other contract or tort claim), IBM is liable for no more than 1) damages for bodily injury (including death) and damage to real property and tangible personal property and 2) the amount of any other actual direct damages up to the greater of U.S. $100,000 (or equivalent in your local currency) or the charges for the Program that is the subject of the claim.

IBM WILL NOT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS), EVEN IF IBM, OR ITS RESELLER, HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

IBM will not be liable for 1) loss of, or damage to, your records or data or 2) any damages claimed by you based on any third party claim.

This limitation of liability also applies to any developer of a Program supplied to IBM. It is the maximum for which IBM and its suppliers are collectively responsible.

6. General

Nothing in this Agreement affects any statutory rights of consumers that cannot be waived or limited by contract.

IBM may terminate your license if you fail to comply with the terms of this Agreement. If IBM does so, your authorization to use the Program is also terminated.

You agree to comply with applicable export laws and regulations.

Neither you nor IBM will bring a legal action under this Agreement more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation.

Neither you nor IBM is responsible for failure to fulfill any obligations due to causes beyond its control.

The laws of the country in which you acquire the Program govern this Agreement, except 1) in Australia, the laws of the State or Territory in which the transaction is performed govern this Agreement; 2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia (FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this Agreement; 3) in the United Kingdom, all disputes relating to this Agreement will be governed by English Law and will be submitted to the exclusive jurisdiction of the English courts; 4) in Canada, the laws in the Province of Ontario govern this Agreement; and 5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this Agreement.

Part 2 - Country-unique Terms

AUSTRALIA:

Limited Warranty (Section 4):

The following paragraph is added to this Section:

The warranties specified in this Section are in addition to any rights you may have under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

Limitation of Liability (Section 5):

The following paragraph is added to this Section:

Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

EGYPT:

Limitation of Liability (Section 5):

The following replaces item 2 in the first paragraph of this Section:

2) as to any other actual direct damages, IBM's liability will be limited to the total amount you paid for the Program that is the subject of the claim.

FRANCE :

Limitation of Liability (Section 5):

The following replaces the second sentence in the first paragraph of this Section:

In such instances, regardless of the basis on which you are entitled to claim damages from IBM, IBM is liable for no more than 1) damages for bodily injury (including death) and damage to real property and tangible personal property; and 2) the amount of any other actual direct damages up to the greater of a) U.S. $100,000 (or equivalent in local currency) or b) the charges for the Program which is the subject of the claim.

GERMANY:

Limited Warranty (Section 4):

The following paragraphs are added to this Section:

The minimum warranty period for Programs is six months.

In case a Program is delivered without Specifications, we will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. You have to check the usability according to the Program information within the "money-back guaranty" period.

The following replaces the first sentence of the first paragraph of this Section:

The warranty for an IBM Program covers the functionality of the Program for its normal use and the Program's conformity to its Specifications.

Limitation of Liability (Section 5):

The following paragraph is added to the Section:

The limitations and exclusions specified in the Agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

In item 2, replace "U.S. $100,000" with "DEM 1.000.000".

The following sentence is added to the end of item 2 of the first paragraph:

IBM's liability under this item is limited to the violation of essential contractual terms in cases of ordinary negligence.

INDIA:

Limitation of Liability (Section 5):

The following replaces items 1 and 2 in the first paragraph:

1) liability for bodily injury (including death) or damage to real property and tangible personal property will be limited to that caused by IBM's negligence; and 2) as to any other actual damage arising in any situation involving nonperformance by IBM pursuant to, or in any way related to the subject of this Agreement, IBM's liability will be limited to the charge paid by you for the individual Program that is the subject of the claim.

General (Section 6):

The following replaces the fourth paragraph of this Section:

If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party may have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

IRELAND:

Limited Warranty (Section 4):

The following paragraph is added to this Section:

Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing, all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

Limitation of Liability (Section 5):

The following replaces items 1 and 2 in the first paragraph of this Section:

1) death or personal injury or physical damage to your real property solely caused by IBM's negligence; and 2) the amount of any other actual direct damages, up to the greater of Irish Pounds 75,000 in respect of Programs or 125 percent of the charges for the Program that is the subject of the claim or which otherwise gives rise to the claim.

The following paragraph is added at the end of this Section:

IBM's entire liability and your sole remedy, whether in contract or in tort, in respect of any default will be limited to damages.

ITALY:

Limitation of Liability (Section 5):

The following replaces the second sentence in the first paragraph:

In each such instance unless otherwise provided by mandatory law, IBM is liable for no more than damages for bodily injury (including death) and damage to real property and tangible personal property and 2) as to any other actual damage arising in all situations involving non-performance by IBM pursuant to, or in any way related to the subject matter of this Agreement, IBM's liability, will be limited to the total amount you paid for the Program that is the subject of the claim.

NEW ZEALAND:

Limited Warranty (Section 4):

The following paragraph is added to this Section:

The warranties specified in this Section are in addition to any rights you may have under the Consumer Guarantees Act 1993 or other legislation which cannot be excluded or limited. The Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if you require the goods or services for the purposes of a business as defined in that Act.

Limitation of Liability (Section 5):

The following paragraph is added to this Section:

Where Programs are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

PEOPLE'S REPUBLIC OF CHINA:

Charges (Section 3):

The following paragraph is added to the Section:

All banking charges incurred in the People's Republic of China will be borne by you and those incurred outside the People's Republic of China will be borne by IBM.

UNITED KINGDOM:

Limitation of Liability (Section 5):

The following replaces items 1 and 2 in the first paragraph of this Section:

1) death or personal injury or physical damage to your real property solely caused by IBM's negligence; 2) the amount of any other actual direct damages, up to the greater of Pounds Sterling 75,000 in respect of Programs or 125 percent of the charges for the Program that is the subject of the claim or which otherwise gives rise to the claim.

The following item is added:

3) breach of IBM's obligations implied by Section 12 of the Sale of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

The following paragraph is added at the end of this Section:

IBM's entire liability and your sole remedy, whether in contract or in tort, in respect of any default will be limited to damages.

Z125-3301-10 (10/97)

End of Document