MQSeries link LotusScript Extension
User's Guide

Release 1.3

Copyright

Original: Original book produced for IBM MQSeries link LotusScript Extension.

Preface

The MQSeries link LotusScript Extension User's Guide describes the IBM MQSeries link LotusScript Extension, and shows you how you can use it in your Lotus Notes applications.

Information in this book includes

- How to install the MQLSX
- Guidance on how to design and program your applications using the MQLSX
- Code samples and how you can use them in your own applications
- Where to find more information about MQSeries™, Lotus Notes™ and LotusScript™
- How to use trace
- Hints and tips for programmers
- What are some of the common pitfalls
- Reason codes
- A full reference guide to the MQLSX Classes and their use

This book covers the MQSeries link LotusScript Extension (MQLSX) and complements both Lotus and MQSeries publications. An on-line version is also available as a Notes database.

The typographical conventions used in this document are the same as those used in the LotusScript Programmer's Guide

Who this book is for

This book is for designers and programmers wanting to develop Lotus Notes applications that need to interoperate with other, non-Notes applications, using LotusScript.

This book is for you if:

- You are an experienced Lotus Notes developer who may or may not be experienced in using LotusScript.
- You or others within your enterprise have some experience or knowledge of MQSeries.

Contents

Chapter 1 Introduction

This book:

- explains how to install the MQSeries link LotusScript Extension (MQLSX)
- provides you with help when using the MQLSX
- describes each of the MQLSX classes with their properties and methods.

If you are not very familiar with the Message Queue Interface (MQI), you will find it useful to have a copy of the *MQSeries Application Programming Reference* manual.

MQLSX overview

The MQSeries link LotusScript Extension (MQLSX) enables your Notes™ application to interact with other, non-Notes, applications throughout your enterprise.

It gives your Notes LotusScript™ application the ability to run transactions and access data on any of your enterprise systems that you can access through MQSeries. It gives you integration between Lotus Notes and MQSeries software, extending the scope of Notes to include data and transactions that are part of other environments.

The MQLSX is an Application Programming Interface that you call from LotusScript to access the MQI. It requires an MQSeries environment with an MQSeries application to process the messages that your Notes application generates.

The MQLSX code does not make any calls to Notes. Your applications handle what is updated in Notes, splitting the messages received from MQSeries into fields, and adding them to new or existing Notes documents.

Two samples are provided:

- MQLSX Starter sample. You are recommended to use this initially to check that your installation of the MQLSX is successful and that you have the basic MQSeries environment in place. The sample is also provided to demonstrate how LotusScript and the MQLSX can be used.
- MQLSX Link sample application. This provides similar functionality to the MQSeries Link, and is implemented using the MQLSX. It demonstrates how you can use the MQLSX and the Agent function within Notes to interact with an MQSeries application. For purposes of this sample, the MQSeries sample program, amqslnk0, is the MQSeries application.

Note This release of MQLSX can only be used with Notes Release 4.5.1 or later, although no testing has been carried out with Notes 4.6.

MQSeries environment support

To run the MQLSX in an MQSeries server environment you need at least one of the following installed on your system:

- MQSeries for AIX Version 2.2.1 or later
- MQSeries for OS/2 Version 2.0.1 or later
- MQSeries for HP-UX Version 2.2.1 or later
- MQSeries for Sun Solaris Version 2.2 or later
- MQSeries for Windows Version 2.0 or later
- MQSeries for Windows NT Version 2.0 or later

Note MQSeries for Windows Version 2.0 does not support MQSeries clients.

To run the MQLSX in an MQSeries client environment you need at least one of the following installed on your system:

- MQSeries client on AIX™
- MQSeries client on OS/2™
- MQSeries client on Sun Solaris™
- MQSeries client on HP-UX™
- MQSeries client on Windows™ 3.1
- MQSeries client on Windows NT™

Note The MQSeries client requires access to at least one supporting MQSeries server .

MQSeries link LotusScript Extension or MQSeries Enterprise Integrator for Lotus Notes?

This section describes the strengths of each product to help you decide whether you should be using the MQLSX or the MQEI to connect to your enterprise.

MQSeries link LotusScript Extension (MQLSX)

- Incorporates the full power of the MQI.
- MQSeries object model conformance.
  Useful if you are already familiar with the MQSeries object model.
- Better performance.
  MQLSX performs slightly better because there is no database lookup at runtime.

  Note This is dependent on the speed of your network and systems where the databases are stored.
- No Notes dependency.
  MQLSX has no Notes dependency, just a LotusScript dependency. This allows you to use it from SmartSuite products in a Notes free environment. MQEI can be used from SmartSuite but requires Notes to be present to access the MQEI Definition and MQEI Security databases. (This function was not implemented in the Beta.)

MQSeries Enterprise Integrator for Lotus Notes (MQEI)

- Common API for accessing enterprise services regardless of the nature of the enterprise system.
  The API has a common set of verbs that abstract away from the details of each enterprise system. The programmer only needs to learn this single API.
- LotusScript program independence from network configuration.
  For example, names of queue managers and queues are not coded into the LotusScript but into MQEI Service definitions within the MQEI Definition database.
- LotusScript program independence from message formats.
  Similarly, the exact format of messages are not coded into the LotusScript but into MQEI Message definitions within the MQEI Definition database. If you want to use an MQEI Message definition in several places, you only need a single definition that can be shared.
- MQSeries or CICS can be used as network transport.
- MQSeries IMS and CICS bridge headers are automatically built by the MQEI when sending a message.
- Integrated security features through the MQEI Security database allow you to seamlessly sign on to your enterprise systems.

Where to find more information about MQSeries

A variety of MQSeries publications are available to help you use the MQLSX. The following books are a selection that you may find particularly useful:

- *MQSeries: An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Planning Guide*, GC33-1349
- *MQSeries Command Reference*, SC33-1369
- *MQSeries for AIX Version 2.2.1 System Management Guide*, SC33-1373
- *MQSeries for HP-UX Version 2.2.1 System Management Guide*, SC33-1633
- *MQSeries for OS/2 Version 2.0.1 System Management Guide*, SC33-1371
- *MQSeries for Sun Solaris Version 2.2 System Management Guide*, SC33-1800
- *MQSeries for Windows NT Version 2.0 System Management Guide*, SC33-1643
- *MQSeries System Administration*, SC33-1873
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Clients*, SC33-1632
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Application Programming Reference Summary*, SX33-6095
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Distributed Queuing Guide*, SC33-1139
- *MQSeries for Windows Version 2.0 User's Guide*, *GC33-1822-00*

Each of these publications includes a complete list of the MQSeries publications available.

A further source of information is the MQSeries home page on the Internet, located at:

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

Where to find more information about LotusScript

Lotus provide the following documentation for LotusScript:

- *The LotusScript Programmers Guide* Part No. 312106
- *The LotusScript Language Reference* Part No. 12382

A further source of information is the Lotus home page on the Internet, located at:

http://www.lotus.com/

Chapter 2 Installation and Configuration

This chapter explains what software you need on your system before you install the MQLSX package, the steps you take to install the package, and how to check that the installation has been successful.

Pre-installation considerations

This release of the MQLSX runs only with Lotus Notes Release 4.5.1 or later, (although no testing has been carried out with Notes 4.6) in either the Notes client or Domino Server environment.

Note It is strongly recommended that you read the ReadMe file provided, or the printed Release Notes, before commencing installation.

MQLSX MQSeries requirements

The MQLSX requires access to either an MQSeries client or an MQSeries server (from the following list) that is installed in the same environment:

- MQSeries client for AIX
- MQSeries client for HP-UX
- MQSeries client for OS/2
- MQSeries client for Sun Solaris
- MQSeries client for Windows 3.1
- MQSeries client for Windows 95
- MQSeries client for Windows NT
- MQSeries for AIX Version 2.2.1 (for the server)
- MQSeries for HP-UX Version 2.2.1 (for the server)
- MQSeries for OS/2 Version 2.0.1 (for the server)
- MQSeries for Sun Solaris Version 2.2 (for the server)
- MQSeries for Windows NT Version 2.0 (for the server)
- MQSeries for Windows Version 2.0 (for the leaf-node server)

If you choose to use one of the MQSeries client environments, connect it to an MQSeries server that supports it. This can be any MQSeries server that supports the MQSeries client, and does not have to be a server capable of running Notes.

Note MQSeries for Windows is different from the other MQSeries family of products. It is designed to run on a workstation with Microsoft Windows 3.1, Windows for Workgroups, Windows 95 or WIN-OS/2.
For more information see the *MQSeries for Windows User Guide.*

To check your installation is successful, you are recommended to run the MQLSX Starter sample.

Disk space requirements

Disk space requirements for the MQLSX executable code and samples depend on the platform you are running:

AIX	13.0 MB
HP-UX	12.5 MB
OS/2	15.0 MB
Sun Solaris	13.0 MB
Windows 3.1	10.0 MB
Windows 95	12.0 MB
Windows NT	12.0 MB
Windows for Workgroups	10.0 MB
WIN-OS/2	10.0 MB

- 1.6 MB is required to hold the Encapsulated PostScript file version of the *MQSeries link LotusScript Extension User's Guide*
- 0.9 MB is required to hold the Portable Document Format (PDF) file version of the *MQSeries link LotusScript Extension User's Guide*
- 1.6 MB is required for the on-line *MQSeries link LotusScript Extension User's Guide*, the MQLSX Starter sample, Conversion Tables, and the readme text

The MQLSX package

The MQLSX package is usually provided on a CD. For more information about installing the MQLSX package on your system from CD, see "Installing MQLSX" later in this chapter.

The installation process creates a root directory with subdirectories docs, samples, bin, and conv, with contents as shown in the following table.

The MQLSX package may also be provided as a **.**tar**.**Z file for UNIX systems and as a **.**zip file for INTEL systems.

Directory	File name	What it is...
root	readme.txt	A file containing any product and information updates that have become available since this documentation was produced.
docs	gmqlhelp.nsf	A Notes database containing the online version of the *MQSeries link LotusScript Extension User's Guide*.
	gmqlhelp.ps	The *MQSeries link LotusScript Extension User's Guide* as an Encapsulated PostScript file. This will print on both A4 and letter stationery.
	gmqlhelp.pdf	The *MQSeries link LotusScript Extension User's Guide* in Portable Document Format that can be printed and read online using the Adobe™ Acrobat™ reader.
samples	gmqlsamp.nsf	A Notes database containing the MQLSX Starter sample that demonstrates how you can use the MQLSX. You can also use this sample to check that the MQSeries queue manager is installed and running properly.
	gmqlclnt.nsf	A Notes database containing the MQLSX Link client sample. See MQSeries link LotusScript Extension sample application for details.
	gmqlagnt.nsf	A Notes database containing the MQLSX Link agent sample. See MQSeries link LotusScript Extension sample application for details.
	gmqlagnt.txt	A commented version of the MQLSX Agent program.

Directory	File name	What it is...
	mqlink.nsf	A Notes database containing a sample document holding control information for processing a message when running the MQLSX Link sample application. This is the same database that is provided by many MQSeries products.
	gmqlxtra.nsf	A Notes database containing the MQLSX link extra agent sample.
	mqlinkx.nsf	A Notes database containing a sample document holding control information for processing a message when running the MQLSX link extra agent sample application. This is a modified version of the database with the same name that is provided by several MQSeries SupportPacs.
	mqlinkx.ntf	A Notes design template for mqlinkx.nsf
	test1.nsf and test2.nsf	Notes databases containing demonstration applications.
	mqlxdemo.exe mqlxdemo.dat mqlxdemo.c	An MQSeries sample application and a supporting data file. A sample C program that performs an MQPut to a message queue.
	amqslnk0.c	An MQSeries sample program needed as part of the MQLSX sample application, also provided by many MQSeries products.
	amqslnk0.tst	An MQSeries script that creates the queues needed for the MQLSX link sample application.
bin (AIX)	libmqlsx.a	A directory containing the AIX 4.1 version of the MQLSX (MQSeries API Library to Lotus Notes). This is supported when used with the AIX Notes client or server code in the following environments: • MQSeries AIX client running under AIX 4.1.4 or later • MQSeries for AIX server running under AIX 4.1.4 or later
	mqlsxmqm	Dynamic load library for MQSeries server
	mqlsxmqic	Dynamic load library for MQSeries client
	gmqlevel	Code Level Service Utility

Directory	File name	What it is...
	amqslnk0	MQSeries sample program
bin (HP-UX)	libmqlsx.sl	A directory containing the HP-UX version of the MQLSX (MQSeries API Library to Lotus Notes). This is supported when used with the HP-UX Notes client or server code in the following environments: • MQSeries HP-UX client running under HP-UX Version 10.01 or later Version 10 • MQSeries HP-UX server running under HP-UX Version 10.01 or later Version 10
	mqlsxmqm	Dynamic load library for MQSeries server
	mqlsxmqic	Dynamic load library for MQSeries client
	gmqlevel	Code Level Service Utility
	amqslnk0	MQSeries sample program
bin (OS/2)	MQLSX.DLL	A directory containing the OS/2 version of the MQLSX (MQSeries API Library to Lotus Notes). This is supported when used with the OS/2 Notes client or server code in the following environments: • MQSeries OS/2 client running under OS/2 V2.1, OS/2 Warp, Warp Connect, or OS/2 Warp server • MQSeries OS/2 server running under OS/2 V2.1, OS/2 Warp, Warp Connect, or OS/2 Warp server
	GMQLEVEL.EXE	Code Level Service Utility
	amqslnk0	MQSeries sample program
bin (Sun Solaris)	libmqlsx.so	A directory containing the Sun Solaris version of the MQLSX (MQSeries API Library to Lotus Notes). This is supported when used with the Sun Solaris Notes client or server code in the following environments: • MQSeries Sun Solaris client running under Sun Solaris 2.4 or later • MQSeries Sun Solaris server running under Sun Solaris 2.4 or later
	mqlsxmqm	Dynamic load library for MQSeries server
	mqlsxmqic	Dynamic load library for MQSeries client
	gmqlevel	Code Level Service Utility
	amqslnk0	MQSeries sample program

Directory	File name	What it is...
bin (Windows 3.1, Windows for Workgroups and WIN-OS/2)	MQLSX.DLL	A directory containing the Windows 3.1, the Windows for Workgroups and WIN-OS/2 version of the MQLSX (MQSeries API Library to Lotus Notes). This is supported when used with the Win16 Notes client code in the following environments: • MQSeries client on Windows 3.1 running under Windows 3.1 • MQSeries client on Windows 3.1 running under OS/2 Warp and Warp Connect • MQSeries for Windows leaf-node server running under Windows 3.1 • MQSeries for Windows leaf-node server running under Windows for Workgroups • MQSeries for Windows leaf-node server running under WIN-OS/2
	GMQLEVEL.EXE	Code Level Service Utility
	amqslnk0	MQSeries sample program
bin (Windows 95 and Windows NT)	MQLSX.DLL	A directory containing the Windows NT and the Windows 95 version of the MQLSX (MQSeries API Library for LotusScript). This is supported when used with the win32 Notes client or server code in the following environments: • MQSeries client on Windows NT running under Windows NT server 3.51 or the Windows NT workstation 3.51 • MQSeries for Windows NT server running under Windows NT server 3.51 or the Windows NT workstation 3.51 • MQSeries for Windows leaf-node server running under Windows 95
	GMQLEVEL.EXE	Code Level Service Utility
	amqslnk0.exe	MQSeries sample program
conv		A directory containing the files required to support the LotusScript **ReadString** and **WriteString** methods (when character conversion is requested). The file *readme.ccs*, in this directory, details the supported conversions:
	README.CCS	Details the supplied conversions
	GMQLCCS.TBL	Used by MQLSX to establish allowed conversions

Directory	File name	What it is...
	nnnnmmmm.TBL	Table for supported conversions, where nnnn is hex value of the coded character set identifier (CCSID) for the 'from' codepage, mmmm is the hex value of the ccsid for the 'to' codepage

Before installing the MQLSX

IMPORTANT:

- Setting of the GMQ_XLAT_PATH environment variable is mandatory. This automatically set as part of the installation process. Do not change it. If it is not set correctly data conversion will **not** occur.
- The translation tables provided are essential if you need any data conversion to take place.
- If you have an earlier release of the MQLSX, the MQLSX shared library must not be in use at the time of installation.
- You do not need to recompile any of your scripts.

Updating your MQLSX installation

If MQLSX is re-installed over an existing installation, and any of the sample Notes database files shipped with MQLSX are found in the SAMPLES directory, they are automatically backed up in case you had modified them.

In all other respects follow the same steps as if installing for the first time.

Installing MQLSX

This section describes how to install MQLSX on your operating system.

Installing on AIX

Logged on as root:

1. Insert the MQEI / MQLSX CD-ROM into your CD-ROM drive, unless your are installing from a server machine.
2. From the shell type:

```
smit
```

   You can use the alternative fastpath command instead:

```
smitty install_latest
```

At this point, you need to follow the instructions that relate to the level of AIX you are running.

If you are running AIX 4.1.n:

1. Select the device appropriate for your installation using this sequence of windows:

   Software Installation and Maintenance

   Install and Update Software

   Install/Update Selectable Software (Custom Install)

   Install Software Products at Latest Level

   Install New Software Products at Latest Level

2. On panel displayed, you need to enter the device name attached to your CD-ROM reader and the directory in which the new software resides (on the CD or on your server). Press PF4 for a list of input devices known to your system. Enter the CD-ROM device name or the server device name. Press enter.
3. Select 'SOFTWARE to install', press enter
4. Press List to get a list of all available software. To install all of the MQLSX components, select the line showing '1.3.0.0 mqlsx' , and press enter. If you want to install specific components move the cursor to the component line select it. Only press enter after you have selected the components you want.
5. There is no need to change any of the defaults displayed. Press enter.
6. 'Are you Sure?', press enter.
7. A summary installation panel is displayed, followed by Command OK.

If you are using AIX 4.2

1. Select the device appropriate for your installation using this sequence of windows:

   `Software Installation and Maintenance`

   `Install and Update Software`

   `Install and Update from LATEST Available Software`

On the 'INPUT device/directory for software panel:

2. Select the directory the software is to be installed from, on the CD or on your server. Press OK.
3. Select 'SOFTWARE to install', press enter
4. Press List to get a list of all available software. To install all of the MQLSX components, select the line showing '1.3.0.0 mqlsx' , and press enter. If you want to install specific components move the cursor to the component line select it. Only press enter after you have selected the components you want.
5. Set the options you want on the 'Installation Options' panel. Press OK.
6. Press OK on the confirmation panel.

The MQLSX components are now installed on your system in:

- usr/lpp/mqm/mqlsx
  readme - A readme file. Text that MUST be read before using the MQEI.
  All other files are needed by the uninstall option.
- usr/lpp/mqm/mqlsx/bin
  A directory containing the MQLSX executables.
- usr/lpp/mqm/mqlsx/conv
  A directory containing the files needed to support character conversion.
- usr/lpp/mqm/mqlsx/docs
  A directory containing the MQLSX User Guide in PostScript format (gmqlhelp.ps), Portable Document Format (gmqlhelp.pdf), and as a Notes database (gmqlhelp.nsf).
- usr/lpp/mqm/mqlsx/lib
  A directory containing the AIX version of the MQLSX.
- usr/lpp/mqm/mqlsx/samples
  A directory containing components needed to run the MQLSX samples. This directory include Notes databases.

Installing on HP-UX

Use the HP-UX **swinstall** program, to install the MQLSX :

Logged on as root:

1. If you are installing from the CD, insert the MQEI / MQLSX CD-ROM into your CD-ROM drive.
2. Type:

   ```
   swinstall -s/<cd_mount_point>/hp/mqlsx.fpkg
   ```

substituting <cd_mount_point> with the name of your CD-ROM device or the directory on which the MQLSX package is available to you.

3. From the Software Selection panel displayed:
   - press the spacebar to highlight the line for the complete package
   - press enter to expand the package into its components, highlight the ones you want to install
   Type m to mark for install, or click Actions ..... Mark for Install, from the menu bar.
   Press OK on error selection box if displayed, it's for information only.
   Click Actions ..... Install (analysis) from the menu bar.
4. When status is Ready, check there are no errors or warnings listed in the log file.
5. Press OK to continue install.
6. Press Yes to start install.
7. Check there are no errors or warnings listed in the log file.
8. Press Done to exit install window.
9. Select File ..... Exit from the menu bar.

The MQLSX components are now installed on your system in:

- opt/mqm/mqlsx
  readme - A readme file. Text that MUST be read before using the MQEI.
  All other files are needed by the uninstall option.
- opt/mqm/mqlsx/bin
  A directory containing the MQLSX executables.
- opt/mqm/mqlsx/conv
  A directory containing the files needed to support character conversion.
- opt/mqm/mqlsx/docs
  A directory containing the MQLSX User Guide in PostScript format (gmqlhelp.ps), Portable Document Format (gmqlhelp.pdf), and as a Notes database (gmqlhelp.nsf).

- opt/mqm/mqlsx/lib
  A directory containing the HP-UX version of the MQLSX.
- opt/mqm/mqlsx/samples
  A directory containing components needed to run the MQLSX samples. This directory include Notes databases.

Installing on OS/2

If you are installing from the CD, place it in the CD-ROM drive. If you are installing from a server, make sure you are connected to it.

1. From an OS/2 window or a full-screen session:
   - Exit any other OS/2 applications and processes that you may have running.
   - Change to the drive from which you want to install the MQLSX
   - Change to the \OS2\MQLSX\EN_US directory
   - At the command prompt, type INSTALL and press the enter key.
2. The MQLSX Welcome logo is displayed, overlaid with the instructions window. Select the Continue button.
3. The Install window is displayed. If you select the Update CONFIG.SYS check box, the CONFIG.SYS file is updated automatically as part of the installation process. Your original CONFIG.SYS file is renamed to CONFIG.BAK and is stored in the same directory. If you do not select this check box, a CONFIG.ADD file is generated. This file is a copy of CONFIG.SYS file with the necessary updates to the LIBPATH and PATH statement. You can rename the CONFIG.ADD file to CONFIG.SYS.
4. Select the OK button to continue.
5. The Install - directories window is displayed.
   - The list box shows the components that you can choose to install. When you select one or more components (the component line is highlighted), the Bytes needed field shows the amount of disk space required for installation.
   - The File directory entry field allow you to specify the drive and directory into which the components are to be installed. The default is C:\MQM\MQLSX.
     Select the Disk space button to show how much disk space is free on each drive, and to select another drive for installation. Select the Install button to continue.
6. The Install - progress window is displayed. This window shows:
   - The file currently being installed (source) and the drive and directory into which it is being installed (target).
   - A progress bar, indicating the percentage of files already unpacked and installed.
   - The elapsed time.
   - The status, for example, unpacking, processing or transferring.

If you select the Stop button, you are asked whether you want to delete the partial system you have installed. Select Yes. You are returned to the introductory window. Select File ..... Start Install from the menu bar to start the installation.

7. When the installation is complete, the Installation and Maintenance window is displayed. Select OK. The Introductory window is displayed. Leave the installation program by selecting the Exit button.

The MQLSX components are installed on your system in the following directories, unless you changed the default directory or chose not to install all the components:

- MQM\MQLSX
  readme.txt - A readme file. Text that MUST be read before using the MQEI.
  All other files are needed by the uninstall option.
- MQM\MQLSX\BIN
  A directory containing the MQLSX executables and the OS/2 version of the MQLSX.
- MQM\MQLSX\CONV
  A directory containing the files needed to support character conversion.
- MQM\MQLSX\DOCS
  A directory containing the MQLSX User Guide in PostScript format (gmqlhelp.ps), Portable Document Format (gmqlhelp.pdf), and as a Notes database (gmqlhelp.nsf).
- MQM\MQLSX\SAMPLES
  A directory containing components needed to run the MQLSX samples. This directory include Notes databases.

8. Shut your system down and restart.

Installing on Sun Solaris

1. If you are installing from the CD, check to see if the Volume Manager is running on your system by typing the following command:

   ```
   /usr/bin/ps -ef ¦ /bin/grep vold
   ```

If it is running, the CD is mounted on /cdrom/domino_mq automatically. If it is not running, mount the CD by typing the following commands:

```
mkdir -p /cdrom/domino_mq

mount -F hsfs -r /dev/dsk/cntndnsn
/cdrom/domino_mq
```

substituting cntndnsn with the name of your CD-ROM device.

2. Use the Solaris **pkgadd** program, to install the MQLSX type:

   ```
   pkgadd -d /<cd_mount_point>/solaris/mqlsx.img
   ```

substituting <cd_mount_point> with cdrom/domino_mq or the directory on which the MQLSX package is available to you.

3. You are prompted for a list of packages to install. Press enter to accept the default, or select 1 or all and press enter. As there is only one component all these actions have the same result.
4. You are prompted for installable options. Select those you wish to install.

   Note Remember, if you do not choose all the options, and you want to install a further option later, the pkgadd program requires you to uninstall the original options followed by a reinstall of all the options you require.
5. Press the Enter key
6. Enter Y and press the Enter key to:

   ```
   This package contains scripts which will be executed with
   super-user permission during the process of installing
   this package.

   Do you want to continue with the installation of <mqlsx>
   [y,n?]
   ```

"Installation of <mqlsx> was successful is displayed on completion.

The MQLSX components are now installed on your system in:

- opt/mqm/mqlsx
  readme - A ReadMe file. Text that MUST be read before using the MQLSX.
  All other files are needed by the uninstall option.
- opt/mqm/mqlsx/bin
  A directory containing the MQLSX executables.

- opt/mqm/mqlsx/conv
  A directory containing the files needed to support character conversion.
- opt/mqm/mqlsx/docs
  A directory containing the MQLSX User Guide in PostScript format (gmqlhelp.ps), Portable Document Format (gmqlhelp.pdf), and as a Notes database (gmqlhelp.nsf).
- opt/mqm/mqlsx/lib
  A directory containing the Sun Solaris version of the MQLSX.
- opt/mqm/mqlsx/samples
  A directory containing components needed to run the MQLSX samples. This directory include Notes databases.

Installing on Windows 3.1, Windows for Workgroups, WIN OS/2

If you are installing from the CD, place it in the CD-ROM drive. If you are installing from a server, ensure you are connected to it.

1. From Program Manager:
   - Select File ..... Run
   - Type drive:\win16\mqlsx\setup
     substituting drive with the name of the drive from which you want to install the MQLSX
   - Press enter
2. The MQLSX Welcome panel is displayed, introducing the installation process. Select Next to continue.
3. Select the destination path panel, is displayed. If you want to use the default drive and directory, select the Next button to continue. Alternatively, change the drive and directory as required and select the Next button to continue.

   Note For the uninstall option to work, you must not move the files after installation.
4. Select Components panel is displayed, showing a list of components that you can install. All components are selected by default. To deselect any component, click on the tick-mark preceding it. When you have selected the components you want, select the Next button to continue.
5. Select Program Folder panel is displayed. The default name is MQSeries link LSX. Choose a name and select the Next button to continue.
6. Start Copying Files panel is displayed, summarizing the selections you have made so far. If any amendments are necessary, use the Back button to return to the relevant window and make any changes. Otherwise, select the Next button to continue.
7. The next panel displayed shows the progress of the installation process.

   Note Select the Cancel button if you have a need to stop the install, in which case the Exit Setup window is displayed. Select the Exit Setup button to stop the install, otherwise select the Resume button to continue with the install.
8. Setup Complete panel is displayed. Uncheck the box if you do not want to view the ReadMe file at this point, it is available in the folder you chose earlier. Select the Finish button.
9. Installation is now complete. If you have chosen to view the ReadMe file, the Notepad application runs to display the file.
10. Reboot your system.
    Your AUTOEXEC.BAT file is updated with the following statements:

* SET PATH=C:\MQM\MQLSX\BIN
* SET GMQ_XLAT_PATH=C:\MQM\MQLSX\CONV

Check your PATH statement has been updated correctly. If it has been truncated, see your DOS documentation for more information.

The MQLSX components are now installed on your system in the following directories, unless you changed the default directory or chose not to install all the components:

- mqm\mqlsx
  readme - A ReadMe file. Text that MUST be read before using the MQLSX.
  All other files are needed by the uninstall option.
- mqm\mqlsx\bin
  A directory containing the MQLSX executables and the Windows 16 bit version of the MQLSX.
- mqm\mqlsx\conv
  A directory containing the files needed to support character conversion.
- mqm\mqlsx\docs
  A directory containing the MQLSX User Guide in PostScript format (gmqlhelp.ps), Portable Document Format (gmqlhelp.pdf) and as a Notes database (gmqlhelp.nsf).
- mqm\mqlsx\samples
  A directory containing components needed to run the MQLSX samples. This directory include Notes databases.

Installing on Windows NT and Windows 95

If you are installing from the CD, place it in the CD-ROM drive. If you are installing from a server, ensure you are connected to it.

If you are installing on Windows NT, follow the instructions for the version of Windows NT you are using.

Tip You are recommended to exit any other Windows applications that you may have running before you start to install the MQLSX.

Note For the uninstall option to work, you must not move the files after installation.

Windows NT version 4 and Windows 95

From your Windows desktop:

- Click Start - Run...
- Type
  drive:\win32\mqlsx\setup
  where drive: is the drive letter you are installing from.
- Click OK.
- Go to step 1.

Windows NT version 3.51

From the Windows Program Manager:

- Choose File - Run... from the Windows Program Manager menu bar.
- Type
  drive:\win32\mqlsx\setup
  where drive: is the drive letter you are installing from.
- Click OK.
- Go to step 1.

1. The MQLSX Welcome window is displayed, introducing the installation process. Select Next to continue.
2. Select the destination path panel is displayed. If you want to use the default drive and directory, select the Next button to continue. Alternatively, change the drive and directory (using the Browse... button) as required and select the Next button to continue.
3. Select Components panel is displayed, showing a list of components that you can install. All components are selected by default. To deselect any component, click on the tick-mark preceding it. When you have selected the components you want, select the Next button to continue.

4. Select Program Folder panel is displayed. The default name is MQSeries link LSX. Choose a name for the Program Group folder you want to add the MQLSX icons to and select the Next button to continue.
5. Start Copying Files panel is displayed, summarizing the selections you have made so far. If any amendments are necessary, use the Back button to return to the relevant window and make any changes. Otherwise, select the Next button to begin copying files onto your system.
6. The next window displayed shows the progress of the installation process.

   Note Select the Cancel button if you have a need to stop the install, in which case the Exit Setup window is displayed. Select the Exit Setup button to stop the install, otherwise select the Resume button to continue with the installation.
7. Setup Complete window is displayed. Uncheck the box if you do not want to view the ReadMe file at this point. Select the Finish button.
8. Installation is now complete. If you have chosen to view the ReadMe file, the Notepad application runs to display the file.
9. When the Restart Windows panel is displayed, select a check box. Select:
   Yes - to restart now, or
   No - to restart later
10. Click OK when you have made your selection.

On Windows 95 your AUTOEXEC.BAT file is updated with the following statements:

* SET PATH="%PATH%;C:\MQM\MQLSX\BIN"

* SET GMQ_XLAT_PATH=C:\MQM\MQLSX\CONV

Note During the installation process, your Microsoft Visual C++ 4.0 Runtime library file (MSVCRT40.DLL) file may be updated.

The MQLSX components are now installed on your system in the following directories, unless you changed the default directory or chose not to install all the components:

- mqm\mqlsx
  readme - A ReadMe file. Text that MUST be read before using the MQLSX.
  All other files are needed by the uninstall option.
- mqm\mqlsx\bin
  A directory containing the MQLSX executables and the Windows 32 bit version of the MQLSX.
- mqm\mqlsx\conv
  A directory containing the files needed to support character conversion.
- mqm\mqlsx\docs
  A directory containing the MQLSX User Guide in PostScript format (gmqlhelp.ps), Portable Document Format (gmqlhelp.pdf), and as a Notes database (gmqlhelp.nsf).
- mqm\mqlsx\samples
  A directory containing components needed to run the MQLSX samples. This directory include Notes databases.

Post Installation

Copying Notes databases on your Notes client

1. Copy the file gmqlhelp.nsf to the directory where your Notes database files are kept.
2. Start Notes and select the workspace where you want to keep the database
3. From the Notes main panel select File - Database - Open
4. Enter the File Name gmqlhelp.nsf or select it with the browse button.
5. Select Open. The database will be opened on your screen.
6. Select File - Close. The MQLSX Help database icon will now appear in your Notes workspace.

To install gmqlsamp.nsf, gmqlclnt.nsf, gmqlagnt.nsf, gmqlxtra.nsf, mqlink.nsf, or mqlinkx.nsf repeat the steps 1 to 6, but substitute gmqlhelp.nsf with the appropriate filename.

Copying a database to a Domino Server

If you are installing the online help database on multiple Notes workstations, consider storing the database on a server so that several people can access the database from just one copy. Work with your Notes administrator to determine on which server to place a database, since the administrator is aware of server resources, topology, and network protocols. The server where you store the database should be one that database users can access and that has sufficient memory and disk space to support the database.

Copy gmqlhelp.nsf to your server in the following way:

1. Start Notes and select the workspace where you have installed the database (as described in *Installing Notes databases on a Notes client*).
2. On your workspace, select the icon for the database you want to copy.
3. Select File - Database - New Copy
4. Next to server, click the list arrow to display the list of servers. Select the server where you want to place the copy.
5. In the Title box, enter a title for the database, if you want a different title.
6. In the File Name box, enter the file name, if you want a different file name. Limit the file name to eight characters plus the **.**nsf extension.
7. Click OK

To install any other MQLSX notes database on a Domino Server, follow the steps 1 to 7 above, changing the file name accordingly.

MQLSX shared library and data conversion files

The MQLSX provides the MQSeries API to LotusScript. There are separate installation directories for the different platforms.

The MQLSX shared library must reside on the same workstation that the LotusScript is being executed from. You can only run the MQLSX from a Domino Server if Notes agents are executing the LotusScript.

The MQLSX uses data conversion tables to perform Character Set conversion.

Linking to shared library files on Intel platforms

On Intel platforms, the search uses standard system services to check for the shared library in your current working directory before searching your normal shared library search path.

OS/2 and Windows NT

The MQLSX dynamically detects and uses either the MQSeries server DLL (mqm.dll) or the MQSeries client DLL (mqic.dll). The MQLSX searches for and uses these in the order of the server shared library followed by that for the MQSeries client.

Windows 3.1

The MQLSX dynamically detects and uses either the MQSeries server DLL (mqm16.dll as supplied with MQSeries for Windows) or the MQSeries client DLL (mqic.dll). The MQLSX searches for, and uses these, in the order mqm16.dll followed by mqic.dll.

Windows for Workgroups, Windows 95, and WIN-OS/2

The MQLSX uses mqm16.dll as there is no MQSeries client support for these platforms.

Linking to shared library files on UNIX systems

When the libmqmlsx shared library file is used from LotusScript, it dynamically loads mqlsxmqm (for the MQSeries server) or, if it can't be found, the mqlsxmqic (for the MQSeries client) in order to resolve MQI calls.

AIX

The current directory and directories /usr/lpp/mqm/mqlsx/lib, /usr/lib, and /lib are automatically searched for the mqlsxmqm and mqlsxmqic objects (which can be soft links). Alternatively you can override this search path, or point to the object in a different directory using the GMQ_MQ_LIB environment variable.

HP-UX

Use the GMQ_MQ_LIB environment variable to enable libmqlsx.sl to find either mqlsxmqm or mqlsxmqic.

Domino multi-threaded agents

If you are running multi-threaded agents on a Domino server that use the MQLSX, you **must** have an MQSeries queue manager running on the same machine as the Domino server.

MQLSX environment variables

There are five environment variables that you need to know about when setting up the MQLSX on your local system. Since some of them are purely for diagnostic purposes, you do not have to set them all.

- GMQ_TRACE
- GMQ_TRACE_LEVEL
- GMQ_TRACE_PATH
- GMQ_XLAT_PATH
- GMQ_MQ_LIB

GMQ_TRACE

If you want to use the trace facility to help you solve any problems you may be having, switch it on or off using the GMQ_TRACE environment variable. Unless you are having a problem, you are recommended to run with tracing set off to avoid any unnecessary overheads on your system resources.

For more information, see "Using trace" in Chapter 4.

GMQ_TRACE_LEVEL

Use the GMQ_TRACE_LEVEL environment variable to set the level of detail you want recorded in your trace file.

For more information, see "Using trace" in Chapter 4.

GMQ_TRACE_PATH

If you have switched the trace facility on (using the GMQ_TRACE environment variable), you can specify the directory where you want the trace files to be stored. You do not have to give a filename for the trace file - these are created at run time. If you do not specify a directory in the GMQ_TRACE_PATH environment variable, the trace files are written to the current working directory.

You can identify a trace file by the gmqnnnnn.trc file name (where nnnnn is a string of five numbers).

For more information, see "Using trace" in Chapter 4.

Note This was called GMQ_PATH in earlier releases.

GMQ_XLAT_PATH

You **must** set the GMQ_XLAT_PATH environment variable to locate the data conversion tables that are used by the MQLSX.

For more information, see "Data Conversion" in Chapter 3.

GMQ_MQ_LIB

You only need to set the GMQ_MQ_LIB environment variable if you want to override the inbuilt mechanism for picking up MQSeries libraries.

For example, set this to specify the MQSeries client library when both the MQSeries client and local queue manager libraries are available locally and you want to force your program to run as an MQSeries client application, even though it is local to the queue manager.

Under normal circumstances, you should not need to set this value.

Caution Do not set this environment variable to point to the MQLSX. It must be used only to point to the MQSeries libraries on INTEL platforms. On UNIX platforms it must only be used to point to the appropriate MQLSX "stub" objects i.e. mqlsxmqm or mqlsxmqic.

Setting up an environment to run the MQLSX

Hardware requirements

There are no additional hardware requirements above those listed for MQSeries and Lotus Notes.

Software requirements

- Lotus Notes Release 4.5.1 (Domino Server or Notes client) or later.
- The MQLSX package
- MQSeries server or MQSeries client

Setting up your MQSeries environment

Before you run a script using the MQLSX, check that you can run the queue manager that your script will connect to, and that the necessary queues are in place. You can do this by using the command DISPLAY QMGR; the command fails if the queue manager is not running. How to use the DISPLAY QMGR command is explained in *the MQSeries Command Reference*.

Tip If you are in doubt about which shared library the MQLSX is using, you can run the MQLSX with trace on and look for the entry under EstablishEPS.

For more information on how to create and start a queue manager, see the *MQSeries System Management Guide* for your platform.

For more information on how to define a queue, see the *MQSeries Command Reference* manual.

Running an installation verification test

To verify that you have installed the software successfully, run the MQLSX Starter sample provided in the MQLSX package. For details of the Starter sample, see "About the MQLSX Starter sample ", and "Running the MQLSX Starter sample ".

About the MQLSX Starter sample

What is demonstrated in the sample

The sample demonstrates how to use the MQLSX in LotusScript to:

- Connect to a queue manager
- Access a queue
- Put messages on a queue
- Get messages from a queue

Preparing to run the sample

To run the sample you need to have:

- Lotus Notes Release 4.5 (Domino Server or Notes Client) or later.
- An MQSeries Server or an MQSeries Server and an MQSeries Client
- An MQSeries queue manager running
- An MQSeries queue already defined
- The MQLSX installed on the machine where the sample will be run
- The gmqlsamp.nsf database installed on your Notes Client or a Domino server

Running the MQLSX Starter sample

Before you run the MQLSX Starter sample check that you have a queue manager running and you have defined the queue. For details of creating and running a queue manager and defining a queue, refer to the *MQSeries System Management Guide* for your platform.

Starting the sample

1. On your Notes workspace, select the icon for the MQLSX sample
2. From the main menu, select Create - MQLSX sample. The MQLSX sample form is displayed on the screen.

The MQLSX sample form has the following fields and buttons:

- MQSeries Queue Manager name (field)
- MQSeries Queue name (field)
- Data to be sent (field)
- Data received (field)
- Put Msg on Queue (button)
- Get Msg from Queue (button)

Access to the named queue is automatic within the sample.

Putting a message on the queue

1. In the field *MQSeries Queue Manager name*, enter the name of the queue manager that you have running. If you leave this field blank, it will **not** connect to the default queue manager.
2. In the field *MQSeries Queue name*, enter the name of a queue you have defined.
3. In the field *Data to be sent*, enter the message to be put on the queue.
4. Press the button *Put Msg on Queue* to send the message.

Note The first time you Put a message on the queue there may be a delay while the MQLSX connects to the queue manager.

When the message has been successfully Put on the queue, the data in the field *Data to be sent* is cleared.

Getting a message from the queue

1. In the field *MQSeries Queue Manager name*, enter the name of the queue manager that is running.
2. In the field *MQSeries Queue name*, enter the name of the queue you have defined.
3. Press the button *Get Msg from Queue* to get the message.

The sample gets a message from the named queue and displays it in the *Data received* field. If there is no message to Get, the *Data received* field remains blank.

Exception handling

There are some common problems that you may have while running the sample:

- The queue manager name is incorrect
- The queue manager is not running
- The queue name is incorrect
- The queue does not exist

For common problems the MQLSX Starter sample attaches a diagnostic text to the error message Any other error messages include a Reason Code that you can look up in the *MQSeries Application Programming Reference* manual.

The MQLSX Starter sample script

The MQLSX Starter sample database, GMQLSAMP.NSF, contains sample code written in LotusScript and attached to a Notes form. The sample uses the MQLSX to enable you to create and retrieve messages from an MQSeries queue.

The sample is not intended to demonstrate general programming techniques, so some error checking that you may want to include in a production program has been omitted. However, this sample is suitable to use as a base for your own message queuing programs.

Viewing the MQLSX Starter sample code

You have full access to the source code of the MQLSX sample if have a full Notes licence, which you can look at in the following way:

1. In your Notes workspace, click on the MQLSX Starter sample icon to select the sample database.
2. From the main menu select View - Design.
3. From the navigation pane, select Forms. The MQLSX sample form will appear in the view pane.
4. Double click on the MQLSX Starter sample form listed in the view pane. This opens the Form Builder window.
5. From the programming panel at the bottom of the Form Builder window you can select Define and Event items to look at the various elements of the sample form.

For example, if you select Define "Put Msg on Queue (Button)" and Event "Click", you will be able to look at the script that will be run when the "Put Msg on Queue" button is pressed.

Select Define "(Globals) MQLSX Sample" to look at Events including:

- Scripts for various subs that handle the calls to the MQLSX.
- Declarations of Object variables.
- Declarations of MQSeries variables.

Options:

```
Uselsx "mqlsx"
```

is used to load the MQLSX.

Chapter 3 Design and Programming using the MQLSX

This chapter complements the information provided by the MQSeries, Lotus Notes, and LotusScript documentation.

It includes:

- How to access the MQLSX
- Programming hints and tips
- Using the IMS bridge
- Help when you are dealing with applications that run in environments with different code pages (data conversion)
- Error handling

Designing applications that access non-Notes applications

Advantages

The MQLSX brings you the benefits from using both Notes and MQSeries. It extends the scope of Notes to include data and transactions that are part of other environments. Enhancing the information management capabilities of Notes, MQSeries provides commercial messaging for many platforms and time-independent, once-only assured delivery of messages.

Typical applications

- To avoid duplicating or re-typing information; for instance names and addresses need only be held in one place.
- To provide reporting information on the total enterprise situation, from information held on Notes and other applications.
- To process financial transactions such as payment of an insurance premium, without being dependent on someone else entering the information into another system. This can be done from your Notes application using MQSeries to pass on the information, ensuring it is processed when the necessary transaction system is available.

Using the MQLSX

When designing a LotusScript application that uses the MQLSX, the most important item of information is the actual *message* that will be sent or received from the remote MQSeries system. Therefore you must know the format of the items that will be inserted into the message.

You should also know:

- The code page that the remote system runs in
- The encoding that the remote system requires

To help you to keep your code portable it is good practice always to set the code page and encoding even if these are currently the same in both the sending and receiving systems.

Domino Server or Notes client

When considering how to structure the implementation of the system you design, remember that your MQLSX scripts **mus**t run on the same machine that you have MQSeries installed on. If you do not have MQSeries installed locally, you can make use of Lotus Notes agents. An action initiated on a remote Notes client can cause the triggering of an agent on the Domino server where MQSeries is installed. This mechanism is used in the 'MQLSX link sample application'. You will need to do this explicitly, as Notes by default will run a script locally.

When your MQLSX script runs, it will need an MQSeries application that picks up the message your script has sent or one that puts a message on a queue for your script to get. For this to work, both your script and the MQSeries application need to know the structure of the message they are dealing with.

Accessing the MQLSX

In the LotusScript editor, under (options) event put the following:

```
Uselsx "mqlsx"
```

For information on the **Uselsx** statement, see the *LotusScript Language Reference*.

Programming hints and tips

The following hints and tips are in no significant order. They are subjects that, if relevant to the work you are doing, will hopefully save you time.

Using large messages

The MQLSX supports messages up to 4MB long if memory is available. However, when using large messages Notes restricts:

- Plain text to 64K.
- Fields added to the summary buffer to 32K.
- LotusScript string length to 32000 characters.

Tip To overcome this limitation consider appending strings into a rich text field in Notes, which has no size restrictions. The MQLSX has been designed to take advantage of this feature. Copy the data 32000 characters at a time from an MQLSX message into LotusScript strings. From the LotusScript strings, append the data into a rich text field.

For example, where the message data is greater than 32K (the maximum length of a string in LotusScript), read the data in multiple parts. This code fragment assumes that the message, MyMsg, has already been taken from the queue using the get method of the MQQueue class and is less than 64K in length:

```
Dim MessagePartA As String
Dim MessagepartB As String
...
...
MessagePartA = MyMsg.ReadString(32000)
MessagePartB = MyMsg.ReadString(MyMsg.DataLength)
```

Writing large scripts

The maximum storage size of a script is 32K. If you need to write a script larger than this, consider subdividing your script code into functions or use the %include function within Notes.

For a full list of LotusScript limits, see the appendix in the *LotusScript Language Reference* manual.

Embedded nulls in a string

The MQSeries constants, used for the initialization of three MQMessage properties:

MQMI_NONE (24 NULL characters)

MQCI_NONE (24 NULL characters)

MQACT_NONE (32 NULL characters)

are not supported by the MQLSX, but the LotusScript String call allows you to do the same thing.

To set the MessageId of an MQMessage to MQMI_NONE:

```
mymessage.MessageId = String(24,0)
```

To set the CorrelationId of an MQMessage to MQCI_NONE:

```
mymessage.CorrelationId = String(24,0)
```

To set the AccountingToken property of an MQMessage to MQACT_NONE:

```
mymessage.AccountingToken = String(32,0)
```

Message Descriptor properties

Where an MQSeries application is the originator of a message and MQSeries generates the

- AccountingToken
- CorrelationId
- MessageId

you are recommended to use the AccountingTokenHex, CorrelationIdHex, and MessageIdHex properties if you want to look at their values, or manipulate them in any way - including passing them back in a message to MQSeries. The reason for this is that MQSeries generated values are strings of bytes that have any value from zero through to 255 inclusive, they are not strings of printable characters.

Where your MQLSX script is the originator of a message and you generate the

- AccountingToken
- CorrelationId
- MessageId

you are recommended to use the AccountingToken, CorrelationId, and MessageId properties.

Object out of scope

It is good programming practice to delete an object when it goes out of scope.

For example:

```
Dim qms As MQSession
Dim qm As MQQueueManager
...
...
Set qm As qms.AccessQueueManager(MQ_queue_manager)
qm.Connect
...
...
qm.Disconnect
Delete qm
```

The **Delete** call deletes the storage allocated during the **AccessQueueManager** method.

Receiving a message from MQSeries

There are several ways of receiving a message from MQSeries:

- Polling by issuing a **GET** followed by a wait, using the LotusScript TIMER function
- Issuing a **GET** with the **Wait** option; you specify the wait duration by setting the **WaitInterval** property. This is recommended when, even though you set your system up to run in a multithreaded environment, the software running at the time may only run singlethreaded. This avoids your system locking up indefinitely.

Caution Issuing a **GET** with the **Wait** option and setting the **WaitInterval** to MQWI_UNLIMITED will cause your system to lockup until the **GET** call completes, if the process is single threaded.

- Issuing a **GET** without the **Wait** option. In this case, once your script has issued the call, control is passed to the next script waiting to run. This second script, and any other scripts that may run before the original script regains control, must not affect any of the objects that the original script will expect to be the same as at the time it lost control.

Automatic buffer management

The MQLSX MQMessage object controls the size of a buffer, dynamically changing (within the limitations of the system) the buffer size to accomodate the data in a message.

The default size of a buffer is 2K bytes. If you want to restrict how much of a message your application gets, and you want it to get more than 2K bytes, you must use the ResizeBuffer method before getting the message with the MQGMO_ACCEPT_TRUNCATED option. However, if you do not wish to use the MQGMO_ACCEPT_TRUNCATED option, the MQLSX will handle all automatic buffer management.

Using the IMS Bridge

For applications that put or get messages that involve access through the MQSeries IMS bridge for MVS, there are a few points to bear in mind.

- When you put a message destined for an IMS system, use the IMS bridge header (MQIIH). Set the MQIIH_Format to "MQIMSVS" and the MQMessage Format property to "MQIMS".
- When you get a message back from the IMS bridge, either unpack the header within your application, or offset the message by 84 bytes to ignore the header and point to the start of the message from IMS. An offset of 89 bytes takes you to the first data byte of the message.

Further information about the MQIIH header can be found in the *MQSeries Application Programming Reference* manual.

For more information about designing and writing applications to use the services MQSeries provides, see the *MQSeries Application Programming Guide*.

You may find the following code samples useful when using the IMS Bridge. However, these are code fragments and should not be taken as full programming solutions.

```
' Declare additional IMS Items.  These fields are needed
to submit an IMS transaction.

Dim II As Integer

Dim zz As String

Dim trancode As String

Dim trandata As String

' Additional MQSeries items
```

```
Dim MQITII_NONE As String
        ' The default value for TraninstanceID, not
supplied in MQLSX

MQITII_NONE=string(16,0)
        ' 16 nulls

Dim MQIIH_Encoding As Long

Dim MQIIH_CodedCharSetId As Long

Dim MQIIH_Format As String

Dim MQIIH_Flags As Long

Dim MQIIH_LTermOverride As String

Dim MQIIH_MFSMapName As String

Dim MQIIH_ReplyToFormat As String

Dim MQIIH_Authenticator As String

Dim MQIIH_TranInstanceId As String

Dim MQIIH_TranState As String

Dim MQIIH_CommitMode As String

Dim MQIIH_SecurityScope As String

Dim MQIIH_Reserved As String

' Set additional MQIIH items.  This is to put default
values into the MQIIH fields.

' Note that MQIIH_Format and MQIIH_ReplyToFormat must be
set to "MQIMSVS".

MQIIH_Reserved As String

MQIIH_Encoding = Mqenc_native

MQIIH_CodedCharSetId = Mqccsi_q_mgr

MQIIH_Format = "MQIMSVS "

MQIIH_Flags = MQIIH_NONE

MQIIH_LTermOverride = "        "

MQIIH_MFSMapName = "        "

MQIIH_ReplyToFormat = ″MQIMSVS "

MQIIH_Authenticator = MQIAUT_NONE

MQIIH_TranInstanceId = MQITII_NONE

MQIIH_TranState = MQITS_NOT_IN_CONVERSATION
```

```
MQIIH_CommitMode = MQICM_SEND_THEN_COMMIT
MQIIH_SecurityScope = MQISS_CHECK
MQIIH_Reserved = " "
' Write IIH to MQ message MQmsg.
MQmsg.writestring(Mqiih_struc_id)
MQMsg.writelong(Mqiih_version_1)
MQMsg.writelong(Mqiih_length_1)
MQMsg.writelong(MQIIH_Encoding)
MQMsg.writelong(MQIIH_CodedCharSetId)
MQMsg.writestring(MQIIH_Format)
MQMsg.writelong(MQIIH_Flags)
MQMsg.writestring(MQIIH_LTermoverride)
MQMsg.writestring(MQIIH_MFSMapname)
MQMsg.writestring(MQIIH_ReplyToFormat)
MQMsg.writestring(MQIIH_Authenticator)
MQMsg.writestring(MQIIH_TranInstanceId)
MQMsg.writestring(MQIIH_TranState)
MQMsg.writestring(MQIIH_CommitMode)
MQMsg.writestring(MQIIH_SecurityScope)
MQMsg.writestring(MQIIH_Reserved)
' Write IMS message to MQ message MQmsg.
ll = 4 + Cint(Len(trancode)) + Cint(Len(trandata))    '
total length of message
zz = "  "
       ' reserved for IMS
MQMsg.writeshort(ll)
MQMsg.writestring(zz)
MQMsg.writestring(trancode)
MQMsg.writestring(trandata)
MQmsg.messagetype = 8
'Set message as DATAGRAM
MQmsg.format = "MQIMS   "
'Set Data Convert Format for IMS Bridge.
```

```
MQmsg.ReplyToQueueManagerName = "VM03"    ' MQM for reply
message

MQmsg.ReplyToQueueName = "EF.VM03.SDRC.REMOTE"   ' Reply
Queue on above MQM

MQmsg.UserId = "MYUSER"
' 12 byte userid - default is blanks

' Unpack the reply message.

MQMsg.DataOffset = 89
        ' Jump past the IIH, ll, zz, & attribute byte

replymsg = MQMsg.ReadString(MQMsg.DataLength)    ' get the
reply from IMS
```

Disconnecting from MQSeries

Within your LotusScript program you are recommended to use the Disconnect method before the program ends.

If you don't explicitly disconnect from the MQSeries queue manager, the results may be unpredictable. For example, any messages placed on a queue under syncpoint may not be committed if the program does not explicity call the Disconnect method.

Data conversion

Data conversion is necessary when a message is created on one system and processed by another system where the character set and encoding are different.

There are several occasions when and where this can take place:

- By the queue manager. This can be by the queue manager receiving the message, prior to an application issuing an MQGET call, or the queue manager sending the message to another queue manager. A queue manager is restricted to a set of 'built-in' formats.
- By writing your own data-conversion exit. This is invoked when an application gets a message from a queue.
- By using the MQLSX read and write methods (excluding the ReadUnsignedByte and WriteUnsignedByte methods).

When considering which of these are the most appropriate for your application, note:

- MQSeries does not support data conversion on all platforms.
- Data conversion by a queue manager using the 'built-in' formats converts the whole message.
- The MQLSX read and write methods convert the individual field within a message.

Data conversion by MQSeries

If you want the whole message converted when you retrieve or put a message on a queue, you can request that MQSeries does it rather than doing the conversion within the MQLSX.

When you retrieve a message, use the MQGMO_CONVERT option on the Get method.

When you put a message on a queue, if you haven't set the **CharacterSet (**which corresponds to the MQSeries **CodedCharSetId** attribute**)** and **Encoding** properties they are, by default, set to the values of the system that constructed the message. Use the MQGMO_CONVERT on the Put method within your script, or on the MQGET call within your MQSeries application.

If you set the MQGMO_CONVERT option on the MQGET call within your MQSeries application, MQSeries will attempt to convert the message, unaware of any conversion that has already taken place within your MQLSX application. If you have set the **CharacterSet (CodedCharSetId)** and **Encoding** properties to match the contents of the message, data conversion will only be attempted when necessary.

The *MQSeries Application Programming Guide* explains how data conversion works within MQSeries. It covers the rules a queue manager follows to determine if it is to do the conversion, as well as how to write and invoke your own data-conversion exit.

Data conversion by the MQLSX

The Get method defined in the MQQueue class retrieves a message from an MQSeries queue. This call copies the message from the queue into an internal MQLSX message object.

The Put method defined in the MQQueue class takes the message from the internal MQLSX message object and places it on an MQSeries queue.

In a message, the **CharacterSet (CodedCharSetId)** and **Encoding** properties define the code page and the numeric encodings used within your data part of the message.

Benefits of data conversion by MQSeries

You don't need to know the destination of a message.

If your application is dealing with large messages, or a large number of small messages, the server running MQSeries may have more resources available to handle the volume of data conversion, rather than possibly stretching your workstation to its limits by the conversion taking place within the MQLSX.

You can avoid the situation whereby a message has been removed from the MQSeries queue, however data conversion within the MQLSX fails because either there are missing entries in the gmqlccs.tbl or the conversion file is not available.

Benefits of data conversion using the MQLSX

There are read and write methods provided for the different data types, therefore the conversion is more specific to the contents of the individual field.

If your application only uses a small portion of a large message, the amount of data conversion is considerably reduced using the read methods.

You do not have to understand and write any data-conversion exit programs.

Using the MQLSX methods

Read methods

The read methods use the values held in the **CharacterSet (CodedCharSetId)** and the **Encoding** properties to carry out any necessary data conversion, so that the data is correctly presented to your LotusScript application.

Write methods

When constructing a message, your application should set the **CharacterSet (CodedCharSetId)** and the **Encoding** properties to match the requirements of the receiving system. (If you don't specify any values, they take those for the platform your application is running on.) These values are used by each subsequent write method as your application builds the message, as well as by the MQSeries application when it processes the message.

You should not modify the **CharacterSet (CodedCharSetId)** and **Encoding** properties after you have invoked the Get method or after you have started constructing a message, that will later be put on a queue. If you do change these properties when you are constructing a message, such that they do not match your message data, you must not use the MQGMO_CONVERT on the MQGET call unless you want your message data converted.

Two forms of data conversion are supported by the MQLSX.

Numeric Encoding

If you set the Encoding Property, the following methods will convert between different numeric encoding systems:

- ReadLong Method
- WriteLong Method
- ReadShort Method
- WriteShort Method

MQSeries provides a header file, cmqc.h, that defines the data encoding for the platform you are running on.

Looking at the OS/2 version of this file:

```
#define MQENC_NATIVE 0x00000222L
// MQENC_NATIVE is further broken down into individual encodings for binary integers.....
/* Encodings for Binary Integers */
#define MQENC_INTEGER_NORMAL  0x00000001L
#define MQENC_INTEGER_REVERSED  0x00000002L
```

These definitions can be used when you need to send a 'number' either from a little (for example Intel) to a big endian system, or the other way round.

Example

To send an integer from an Intel system to a System 370 operating system:

```
Dim msg As New Mqmessage              ' Define an MQSeries
message for our use...

myenc = msg.Encoding                  ' Currently 546 (or 222
hex.)

Print myenc
           ' Print the current Encoding property value for
information

msg.Encoding=273
' Set the encoding property to 273 (or 111 in hex.)

myenc = msg.Encoding                  ' Get a copy

Print myenc
           ' Print it to see the change

Dim local_num As long                 ' Define a long integer

local_num = 1234
' Set it

msg.WriteLong(local_num)
'Write the number into the message
```

The WriteLong method reads the encoding property and if it's different from the native setting, it converts the integer accordingly.

Character Set Conversion

Character Set conversion is necessary when you send a message from one system to another system where the code pages are different. Character Set conversion is included in the methods:

- ReadString Method
- WriteString Method

Note You must set the CharacterSet property to a supported character set value (ccsid). If the CharacterSet property is set to an invalid value, no error is reported until a ReadString or WriteString method is called.

The MQLSX uses conversion tables, delivered in the conv directory, to perform Character Set conversion.

Check that the GMQ_XLAT_PATH environment variable has been set to point to the directory on your system to which you copied the contents of the conv directory.

If you don't set the environment variable, the MQLSX will look for the conversion files in your current working directory (for instance, the NOTES directory when you are running NOTES).

Example

To convert strings automatically to Code Page 437:

```
Dim msg As New Mqmessage
          'Define an MQSeries message

msg.CharacterSet = 437
          'Set code page required

mymsg.Writestring "A character string"
'Put the character string in the message
```

The WriteString method converts the Unicode passed in from LotusScript to the characterset associated with the message. This occurs before the string is put in the buffer that is sent to the MQSeries server when you invoke the put method.

Similarly with the ReadString method, the incoming MQSeries message (using the Get method) has a code page associated with it (in the MQMD). The data is converted from this code page to Unicode before being passed to LotusScript.

Establishing a characterset for an environment

As part of the initialization of the MQLSX, the default characterset is established for the Notes instance under which the MQLSX initialization is taking place as follows:

Platform	How a characterset is established
AIX	Using the nl_langinfo function, which returns a value in the form "IBM-850" for example. The system function ccstoccsid converts the majority of values returned from nl_langinfo to a characterset (CCSID), those it cannot convert are converted using a table. The default value 850 is used when a CCSID cannot be established.
HP-UX	First, the local language is established using the nl_function. Typically this is value in the form "roman8". This is converted to a CCSID value using the table: 88591= CCSID 819 88592 = CCSID 912 88595 = CCSID 915 88596 = CCSID 1089 88597 = CCSID 813 88598 = CCSID 916 88599 = CCSID 920 roman8 = CCSID 1051 If the language is not in the table (i.e. not supported), the default value 819 is used.
OS/2	Using the DosQueryCp call. If a value is not returned, the default value 850 is used.

Platform	How a characterset is established
Sun Solaris	First, the language that the session is running under is established using the setlocale function. Typically this returns a value in the form "en_US". This is converted to a CCSID value using the table: 88591= CCSID 819 88592 = CCSID 912 88595 = CCSID 915 88596 = CCSID 1089 88597 = CCSID 813 88598 = CCSID 916 88599 = CCSID 920 roman8 = CCSID 1051 If the language is not in the table (i.e. not supported), the default value 819 is used.
Windows 3.1	By an intdos call using 0x66. If the CCSID cannot be established using this, the default value of 850 is used.
Windows 95	As Windows 3.1 if you are using the 16 bit MQLSX. As Windows NT if you are using the 32 bit MQLSX.
Windows NT	By a call to the system routine GetConsoleCP. The value returned is used as the CCSID, unless no value is returned, when the registry value for OEMCP from SYSTEM\CURRENTCONTROLSET\NLS\CODE PAGE is used. If a value cannot be found, the default value of 850 is used.
WIN-OS/2	As Windows 3.1

The role of the gmqlccs.tbl and NNNNMMMM.tbl files

The gmqlccs.tbl file holds information concerning the relationship between charactersets and code pages. The GMQ_XLAT_PATH environment variable enables the MQLSX to locate the gmqlccs.tbl file during the initialization of the MQLSX. The contents of this file are loaded into memory. When character conversion is required (when a ReadString or WriteString method is called), the information held in memory (from the gmqlccs.tbl file) is used to establish which conversion tables are required.

In this MQLSX release (where Unicode conversion is used), these tables are named 34B0nnn.tbl and nnn34B0.tbl. For example, where the characterset is set to 437, a WriteString method needs 34B001B5.tbl, where 01B5 is the hexidecimal value of 437. The first time the MQLSX uses a table, the contents of the table are loaded into memory ready for when it is needed again.

Note On UNIX systems the names of the gmqlccs.tbl file and the conversion tables, must be all upper case or all lowercase. On Intel platforms the case of the filenames is not critical.

MQLSX character data conversion in detail

Strings created under Lotus Notes are stored in LMBCS (Lotus MultiByte Character Set, which is closely aligned with Unicode) format. When you issue the Uselsx "mqlsx" command within your LotusScript program, the MQLSX classes (MQSession, MQMessage etc.) register themselves together with their properties and methods.

In particular the MQMessage class (which contains the WriteString and ReadString methods) registers itself as using Unicode™. This means is that when LotusScript invokes a method or property of the MQMessage class that involves an input or output string parameter, this is passed in or passed back using Unicode.

WriteString method

The WriteString method requires a string parameter, which is passed by Notes to the MQLSX as Unicode.

For example:

If you enter a dollar sign ($) from your keyboard into a Notes field and subsequently pass this character in a Notes string to the MQLSX using the WriteString method, the MQLSX sees x'0024' - this being the Unicode assigned code-point for dollar. Similarly, if you have a keyboard that supports the pound sign (£), and pass this character in a Notes string to the MQLSX using the WriteString method, the MQLSX sees x'00A3'.

On receiving the method call, the MQLSX uses the gmqlccs.tbl together with the appropriate conversion table to convert from Unicode to the local code page or to another code page that you have specified using the CharacterSet property of the message.

For example:

If the local code page is 437 (MS-DOS Latin US) or the CharacterSet property is set to 437 explicitly, the Unicode value for pound sign (x'00A3') is converted into the 437 code point for the pound sign, which is x'9C' using the conversion table 34B001B5.tbl. This is the data passed that is in the MQSeries message. The MQMD in the resultant MQSeries message indicates that the code page that the message was generated under is 437.

ReadString method

The ReadString method returns a string and uses the reverse mechanism to WriteString.

For example:

If you receive an MQSeries message (using the Get method), the MQMD contains the code page that the message was generated under. If you get the message that was put out in the WriteString method example, any incoming pound signs (£) are present in the message as x'9C'. The ReadString method on the data containing these pound signs converts x'9C' to x'00A3' using the table 01B534B0.tbl and passes it back to Notes.

Losing data when using WriteString.

If the data you enter into your string from Notes contains characters not supported in the code page you are converting to, in the process of converting from Unicode to that code page they are converted to the substitute character (normally x'7F' for ASCII code pages), losing the original data.

In most cases this is not a problem as the characters you can enter from your keyboard into a notes field are all supported by the local code page. However, if you use the CharacterSet property to specify a different code page, some characters may be lost depending on the match between the local code page and the one you are converting to. For ASCII code pages the characters between x '20' and x'7F' should all be converted.

Another area where you risk losing information is if you attempt to coerce Strings to contain data by using the Chr or UChr LotusScript functions.

One way of preventing this loss is to set the CharacterSet property to 1200, which is Unicode. This prevents any conversion taking place; the Unicode string is passed to the MQLSX unconverted.

One implication of this is that any receiving MQSeries application needs to be able to support Unicode. Another implication is that the data takes two bytes for every character.

When data conversion fails

Data conversion fails if you:

- Specify an invalid character set or one for which you do not have the conversion table.
- Have not set the GMQ_XLAT_PATH environment variable
- The gmqlccs.tbl file was not found in the directory specified in GMQ_XLAT_PATH.

If the MQLSX WriteString method fails to convert the data in a field, because the conversion tables are not available, you are likely to get the MQSeries return code MQRC_NOT_CONVERTED (2119). No data is written to the message.

If the MQLSX ReadString method fails to convert the data in a message, the message will no longer be on the MQSeries queue (after the success of the GET method). The message will be in the buffer used by your program, so you can use the PUT method to place it on an MQSeries queue if you wish to exit the program and resolve the problem later. Alternatively you could consider changing your application and do the conversion within MQSeries prior to using the get method.

If the CCSID entry is missing from the gmqlccs.tbl, you are likely to get the MQSeries return code MQRC_TARGET_CCSID_ERROR (2115). The supported conversions are listed in the readme.ccs file provided in the conv directory.

Error handling

Each MQLSX object includes properties to hold error information and a method to reset / clear them. The properties are:

- CompletionCode
- ReasonCode

The method is:

- ClearErrorCodes

Each object also raises events:

- Mqwarning
- Mqerror

How it works

Your MQLSX script/application invokes an MQLSX object's method, or accesses / updates a property of the MQLSX object:

1. The ReasonCode and CompletionCode in the object concerned are updated.
2. The ReasonCode and CompletionCode in the MQSession object are also updated with the same information.

If the CompletionCode is **not** equal to MQCC_OK:

3. The MQLSX issues an Mqwarning or Mqerror event against the object concerned.
4. The event passes to an event handler for the object, if there is one available. If your event handler has cleared the problem, use the ClearErrorCodes method within the error handler which resets the ReasonCode to MQRC_NONE and the CompletionCode to MQCC_OK.
5. On return from the event handler processing, if any, the MQLSX copies the object ReasonCode and CompletionCode once again to the MQSession object.

If the CompletionCode is **not** equal to MQCC_OK:

6. The MQLSX issues an Mqwarning or Mqerror event against the MQSession object.
7. The event passes to an event handler for the MQSession object, if there is one available. If your event handler has cleared the problem, use the ClearErrorCodes method within the error handler which resets the ReasonCode to MQRC_NONE and the CompletionCode to MQCC_OK.

If the MQSession object CompletionCode is equal to MQCC_ERROR:

8. The MQLSX generates a LotusScript error, number 32000. Use this within your script using the On Error statement to process it.
9. Use the Error$ function to retrieve the associated error string. This is in the form:

   MQLSX: ReasonCode=nnnn

   where nnnn is the latest MQSession object ReasonCode.

For more information on how to use the On Event and On Error statements, see the *LotusScript Language Reference* manual.

Getting a property

This is a special case as the CompletionCode and ReasonCode are not always updated:

- If a property get succeeds, the object and MQSession object ReasonCode and CompletionCode remains unchanged.
- If a property get fails with a CompletionCode of warning, the ReasonCode and CompletionCode remain unchanged and no Mqwarning event is raised.
- If a property get fails with a CompletionCode of error, the ReasonCode and CompletionCode are updated to reflect the true values, and error processing proceeds as described.

Using Events and Error handlers

In general, MQLSX errors (unlike other LotusScript errors) can be handled using an Event or Error handler or a combination of these.

Error handling using Event Handlers

Error and warning event handlers for a specific MQLSX object (an MQQueue object for example) can registered by the in-line LotusScript code as shown:

```
          Set MQq =
MQQmgr.AccessQueue(Queue_name.Text,OpenOptions,"","","")

          On Event Mqwarning From MQq Call WarningFromMQq

          On Event Mqerror From MQq Call ErrorFromMQq
```

If the CompletionCode from an MQLSX object method (or update) is not MQCC_OK, then the MQLSX issues an Mqwarning or Mqerror event (depending on severity) for the object concerned, giving control to the appropriate event handler. An event handler might typically perform all required processing for an Mqwarning event allowing the in-line LotusScript code to continue just as if no warning had occurred. For an Mqerror event however, the in-line code may require access to the reason code, or some other indication that the operation was unsuccessful. MQRC_NO_MSG_AVAILABLE, for example, indicates that no message was returned from a 'Get' but in many cases this will not be an error. The following examples illustrate how this can be accomplished using event handlers.

Examples of Mqwarning and Mqerror event handlers

```
'* Mqwarning event handler

Sub WarningFromMQq(MQq As MQQueue)

     Messagebox "Warning From MQq, reason code: "
&MQq.ReasonCode

     MQq.ClearErrorCodes  '* clears reason code, completion
code, and event

End Sub


'* Mqerror event handler

Sub ErrorFromMQq (MQq As MQQueue)

     If MQq.ReasonCode = MQRC_UNKNOWN_OBJECT_NAME Then

          Messagebox "Error From MQq, invalid queue name"

          uidoc.FieldClear ("MQqueue_name")

     Elseif    MQq.ReasonCode = MQRC_NO_MSG_AVAILABLE Then
```

```
          Messagebox "No message available"
          uidoc.FieldClear ("MsgRecvd")
     Else
          Messagebox "Error From MQq, reason code: "
&MQq.ReasonCode
          uidoc.FieldClear ("MQqueue_name")
     End If

     GlobalReasonCode = MQq.ReasonCode
     GlobalCompletionCode = MQq.CompletionCode
     MQq.ClearErrorCodes  '* clears reason code, completion
code, and event

End Sub
```

Note The ClearErrorCodes method must be issued by the Mqerror event handler in order to prevent a LotusScript error being raised when the event completes. Thus status information can only be passed from the event handler to the in-line LotusScript code using global variables. In this example the reason and completion codes are preserved while other examples might use a fatal error indication (such as the MQLSX sample programs for example that use MQFatalError).

If Mqwarning and Mqerror event handlers are not registered for the object concerned, or the completion code is not cleared using ClearErrorCodes, control will be given to the appropriate MQSession event Handler. Mqwarning and Mqerror event handlers for the MQSession object are constructed in a similar way to those described and can be registered as shown in the example:

```
Set MQqms = New MQSession
On Event Mqwarning From MQqms Call WarningFromMQqms
On Event Mqerror From MQqms Call ErrorFromMQqms
```

Error handling using Error Handlers

If an Mqerror event handler is not registered for the MQSession object concerned or the completion code is not cleared by the event handler issuing ClearErrorCodes, the MQLSX generates a LotusScript error 32000. If an Mqwarning event handler is not registered for the MQSession object concerned or the completion code is not cleared by the event handler issuing ClearErrorCodes, then no LotusScript error is generated. MQLSX warnings can thus be handled using an event handler as described above, or, if no Mqwarning event handler is registered, the completion and reason codes associated with the warning will be preserved for handling by the in-line LotusScript code.

To handle MQLSX errors exclusively using error handlers, no Mqerror event handler should be registered. This approach offers the advantages handling both MQLSX and non-MQLSX LotusScript errors in a similar way and making completion and reason code information available to the in-line LotusScript code without the use of global variables.

Error handlers can be registered in the top-level or lower-level procedures as follows:

```
     On Error GoTo HandleError
```

'HandleError' processing might typically be positioned at the end of the procedure as shown (for an objected-oriented implementation):

```
'*************************************************************************
'*  Handle errors
'*************************************************************************
     Exit Sub
HandleError:
     Dim ErrRcd As ErrorRecord
     Set  ErrRcd = New ErrorRecord
     End
End Sub
```

The associated class definition might appear as follows (this can clearly be modified to log the error or perform whatever action is most appropriate).

```
Const MQLSX_ERROR = 32000
Class ErrorRecord
     MQqms As MQSession
     Sub New
          If Err = MQLSX_ERROR Then
               Set MQqms = New MQSession
               If MQqms.ReasonCode <> MQRC_NO_MSG_AVAILABLE
Then
                    Print "MQLSX Error ", Error(), Err(),
Erl()
               End If
               MQqms.ClearErrorCodes
          Else
              Print "LotusScript Error ", Error(), Err(),
Erl()
          End If
     End Sub
End Class
```

For a procedural implementation, the body of the sub 'New' above would replace the 'dim' and 'set' statements below the 'HandleError' label.

Note that an error in a called sub that contains no 'On Error' statement will be caught by an 'On Error' statement in a higher or top-level procedure.

To ignore an MQSeries warning (such as MQRC_NO_MSG_AVAILABLE) each affected procedure must issue a statement of the form:

```
On Error MQLSX_ERROR resume next
```

The MQLSX reason code can then be examined by the in-line LotusScript code and handled (i.e. ignored or otherwise) or returned to the caller for processing as appropriate. The following example is an extract from a class method that attempts to get a message using the MQQueue 'Get' method). RCode& and CCode& are properties of the same (current) class definition as the method containing the code fragment shows, and are used here to preserve the completion and reason codes from the 'Get' for subsequent access by the user of the object. A second On Error MQLSX statement is included to reinstate error handling for subsequent MQLSX methods.

```
On Error MQLSX_ERROR Resume Next
.
MQreplyq.Get MQMsg, MQgmo
.
CCode& = MQreplyq.CompletionCode
RCode& = MQreplyq.ReasonCode
.
On Error MQLSX_ERROR GoTo HandleError
```

Chapter 4 Troubleshooting

This chapter explains the trace facility provided and common pitfalls with help to avoid them.

Code level tool

You may be asked by the IBM/Lotus Service personnel what level of code you have installed.

To do this, run the 'gmqlevel' utility program.

From the command prompt, change to the directory containing the mqlsx.dll or add the full path name and enter:

```
gmqlevel yyyyy > xxxxx.xxx
```

where yyyyy is the name of the shared library (eg. mqlsx.dll)

and xxxxx.xxx is the name of the output file.

If you do not specify an output file, the detail is displayed on the screen.

Using trace

The MQLSX includes a trace facility to help the service organisation identify what is happening when you have a problem. It shows the paths taken when you run your MQLSX script. Unless you have a problem, you are recommended to run with tracing set off to avoid any unnecessary overheads on your system resources.

There are three environment variables that you set to control trace:

- GMQ_TRACE
- GMQ_TRACE_PATH
- GMQ_TRACE_LEVEL

You set these variables in one of two ways.

1. From a command prompt, from which you must subsequently start Notes, as this is only effective locally.
2. By putting the information into your system startup file. This is effective globally.
   - select Main - Control Panel on Windows NT and Windows 95
   - edit your autoexec.bat file on Windows 3.1, Windows for Workgroups, and WIn-OS/2
   - edit your config.sys file on OS/2
   - edit your .profile file on UNIX systems

Tip When deciding where you want the trace files written, ensure that the user has sufficient authority to write to, not just read from, the disk. (This is particularly relevant on UNIX and Windows NT.)

If you have tracing switched on, it will slow down the running of the MQLSX, but it will not affect the performance of your Notes or MQSeries environments. When you no longer need a trace file, it is your responsibility to delete it.

You must stop Notes running to change the status of the GMQ_TRACE variable.

Note The MQLSX trace environment variable is different to the trace environment variable used within the MQSeries range of products. Within the MQSeries range of products, the trace environment variable is used to specify the name of the trace file. Within the MQLSX, the trace environment variable turns tracing on. If you set the variable to a string of characters, any string of characters, tracing will remain switched on. It is not until you set the variable to null that tracing is turned off.

Trace filename and directory

The trace file name takes the form GMQnnnnn.trc, where nnnnn is the id of the Notes process running at the time.

Commands on OS/2, Win-OS/2, Windows 3.1 and Windows NT:

Command	Effect
SET GMQ_TRACE_PATH=drive:\directory	Sets the trace directory where the trace file will be written
SET GMQ_TRACE_PATH=	Removes the GMQ_TRACE_PATH environment variable, the trace file is written to the current working director (when Notes is started)
SET GMQ_TRACE_PATH	Displays the current setting of the trace directory path on OS/2, Windows for WorkGroups, and Windows 3.1
ECHO %GMQ_TRACE_PATH%	Displays the current setting of the trace directory path on Windows NT
SET GMQ_TRACE=xxxxxxxx	This sets tracing ON. You switch traci on by putting one or more characters after the '=' sign For example: SET GMQ_TRACE=yes or SET GMQ_TRACE=no In both of these examples, tracing will set ON
SET GMQ_TRACE=	Sets tracing OFF

Command	Effect
SET GMQ_TRACE	Displays the contents of the environme variable on OS/2, Windows 3.1 and Windows for WorkGroups
ECHO %GMQ_TRACE%	Displays the contents of the environme variable on Windows NT
SET	Displays the contents of all the environment variables on OS/2, Windows 3.1,Windows for WorkGroup and Windows NT

Commands on AIX, HP-UX and Sun Solaris

Command	Effect
export GMQ_TRACE_PATH=/directory	Sets the trace directory where the trace file
unset GMQ_TRACE_PATH	Removes the GMQ_TRACE_PATH enviro the trace file is written to the current work (when Notes is started)
echo $GMQ_TRACE_PATH	Displays the current setting of the trace di
export GMQ_TRACE=xxxxxxxx	This sets tracing ON. You switch tracing or more characters after the '=' sign For example: export GMQ_TRACE=yes or export GMQ_TRACE=no In both of these examples, tracing will be
unset GMQ_TRACE	Sets tracing off
echo $GMQ_TRACE	Displays the contents of the environment
echo	Displays all the settings for all the environ for the session

Trace level

The environment variable GMQ_TRACE_LEVEL allows you to control how much detail is recorded in the trace file. It can be set to any numeric value greater than zero, although any value above nine does not provide any more information.

In addition, you can suffix the value with a + (plus) or - (minus) sign. Using the plus sign, the trace includes all control block dump information and all informational messages. Using the minus sign includes only the entry and exit points in the trace, i.e. no control block information or text is output to the trace file.

The default value of GMQ_TRACE_LEVEL is 2.

Example trace

The example trace below shows 'typical' trace output. It has been annotated and edited in order to illustrate the key features you might want to look for.

```
Trace for program d:\notes\NLNOTES.EXE(MQSeries MQLSX)
   started at Thu Jul 17 06:58:30 1997

@(!)     ***** Code Level is 1.3.0 *****
  !(00144)BuildDate Jul 15 1997

 ************************************************************
 The head of the trace contains details of the code level and
 when the trace was built - these can be important in problem
 resolution.
 ************************************************************

(00144)@06:58:30.159
  -->xxxInitialize
 ************************************************************
 The number in brackets two lines above is the thread-id, whilst
 the time is displayed following the @. These times are based on the
 system clock and cannot be relied on for performance measurements.
 Any lines starting with --> are entry into a function, and these
 will have corresponding exit points with the same number of dashes.
 The number of dashes corresponds to the depth within the code.
 The depth traced can be controlled via use of the GMQ_TRACE_LEVEL
 environment variable.
 ************************************************************

  ---->ObtainSystemCP
  !(00144)Code page is 437
************************************************************
```

```
  One of the first things that happens is that the code page is
  determined from the operating system. In this case its 437,
  which is 01B5 in hex.  The GMQLCCS.TBL file is located
  using the GMQ_XLAT_PATH environment variable. For
  this release (1.2.0) of the product which uses Unicode with
  the MqMessage class its important that this file is found.
 ************************************************************

   <----ObtainSystemCP (rc= OK)

  !(00144)XLAT_PATH is e:\etm\source\samples\dc\generic

  !(00144)Successfully opened GMQLCCS.TBL under path -
e:\etm\source\samples\dc\generic\GMQLCCS.TBL -

  !(00144)Just about to close gmqlccs.tbl


  <--xxxInitialize (rc= OK)


  -->LSX: MainEntryPoint

  !(00144)LSX: Version 1.1


  <--LSX: MainEntryPoint (rc= OK)


  -->LSX: MQLSX_MessageProc

  !(00144)LSX: LSX_MSG_SETPATH received; library loaded from
d:\notes\mqlsx.DLL


  <--LSX: MQLSX_MessageProc (rc= OK)


  -->LSX: MQLSX_MessageProc

  !(00144)LSX: LSX_MSG_INITIALIZE received

  !(00144)LSX: SUCCESS on ClassRegistration of MqQueueManager

  !(00144)LSX: SUCCESS on ClassRegistration of MqGetMessageOptions

  !(00144)LSX: Class MqMessage is using Unicode
```

```
!(00144)LSX: SUCCESS on ClassRegistration of MqMessage
!(00144)LSX: SUCCESS on ClassRegistration of MqPutMessageOptions
!(00144)LSX: SUCCESS on ClassRegistration of MqQueue
!(00144)LSX: SUCCESS on ClassRegistration of MqSession
!(00144)LSX: SUCCESS on ClassRegistration of MqProcess

************************************************************
The above entries show the registration of the varios classes
defined in the MQ LSX. Notice that the registration (with Notes)
of the MqMessage class uses Unicode.
************************************************************
 ---->LSX: RegisterCMQC

 <----LSX: RegisterCMQC (rc= OK)

 ---->LSX: RegisterCMQCFC

 <----LSX: RegisterCMQCFC (rc= OK)

 ---->LSX: RegisterIMQTYPE

 <----LSX: RegisterIMQTYPE (rc= OK)

 <--LSX: MQLSX_MessageProc (rc= OK)

 -->LSX: Class entry point
 !(00144)LSX: LSI_ADTMSG_CREATE received for class:MqSession
 !(00144)LSX: Could not find MQSession for threadID = [144]
 !(00144)LSX: >>> MEM >>> new: 0x1540ad4
 !(00144)LSX: Adding MQSession for threadID = [144]
```

!(00144)LSX: LotusScript >>> Set [X] = new MqSession [0x1540ad4]

<--LSX: Class entry point (rc= OK)

-->LSX: ClassControl

!(00144)LSX: LSI_ADTMSG_ADDREF received for class:MqSession refCount = 1

<--LSX: ClassControl (rc= OK)

-->LSX: ClassControl

!(00144)LSX: LSI_ADTMSG_EVENT_REG received for class:MqSession event = MQWARNING

!(00144)LSX: LotusScript >>> On Event MQWARNING From MqSession Call [X]

<--LSX: ClassControl (rc= OK)

-->LSX: ClassControl

!(00144)LSX: LSI_ADTMSG_EVENT_REG received for class:MqSession event = MQERROR

!(00144)LSX: LotusScript >>> On Event MQERROR From MqSession Call [X]

<--LSX: ClassControl (rc= OK)

****

The following entries show a typical set of trace entries for the method AccessQueueManager against the MqSession. This results in an MQCONN taking place and since this is the first MQSeries API call it dynamically loads the MQSeries dll (in this case mqm.dll). The data passing between the MQLSX and MQSeries is detailed as an aid to problem determination. Each LotusScript invocation of

```
 a method against a MQLSX object results in a set of trace entries
 bounded by a pair of LSX: ClassControl entries.
 ************************************************************

  -->LSX: ClassControl
  !(00144)LSX: LSI_ADTMSG_METHOD received for class:MqSession;
method:AccessQueueManager[3]
  !(00144)LSX: LotusScript >>>
MQSession.accessQueueManager("freddy")
  !(00144)LSX: >>> MEM >>> new: 0x1540b04


  ---->ImqQueueManager::connect


  ------>gmqdyn0a:MQCONN
  !(00144)>>>Queue Manager Name...
      0000  66 72 65 64 64 79 00 00 00 00 00 00 00 00 00 00  : freddy..........
      0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
      --- 1 lines identical to above ---
  !(00144)GMQDYN0A : About to try and find a dynamic library


  -------->EstablishEPs
  !(00144)Using mqm.dll


  <--------EstablishEPs (rc= 1)
  !(00144)GMQDYN0A : About to go off to real MQCONN
  !(00144)GMQDYN0A: Back from real MQCONN
  !(00144)<<<Queue Manager Name...
      0000  66 72 65 64 64 79 00 00 00 00 00 00 00 00 00 00  : freddy..........
      0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
      --- 1 lines identical to above ---
  !(00144)<<<HConn...
      0000  10 2A CC 01                                      : .*..
```

```
!(00144)<<<Completion Code...
    0000  00 00 00 00                          : ....
!(00144)<<<Reason Code...
    0000  00 00 00 00                          : ....

<------gmqdyn0a:MQCONN (rc= OK)

<----ImqQueueManager::connect (rc= OK)

<--LSX: ClassControl (rc= OK)
************************************************************
   : : :
  Various lines missing for clarity
   : : :
************************************************************
-->LSX: ClassControl
!(00144)LSX: LSI_ADTMSG_ADDREF received for class:MqMessage
refCount = 1

<--LSX: ClassControl (rc= OK)

-->LSX: ClassControl
!(00144)LSX: LSI_ADTMSG_METHOD received for class:MqMessage;
method:WriteString[9]
!(00144)LSX: LotusScript >>>       MQMessage.writeString( ... )
    0000  54 00 68 00 69 00 73 00 20 00 69 00 73 00 20 00  : T.h.i.s. .i.s. .
    0010  61 00 20 00 73 00 61 00 6D 00 70 00 6C 00 65 00  : a. .s.a.m.p.l.e.
!(00144)LSX: Notes has passed us 16 (Unicode characters)
!(00144)LSX: WriteString instrlen = 32, outstrlen = 48
************************************************************
```

Here you can see a hex dump of the Unicode data

passed by notes to the LSX via WriteString. This is converted
from Unicode into code page 437 that involves
the loading of the conversion table 34B001B5.tbl. The 34B0 is
Unicode, whilst the 01B5 is the local code page (437) in hex.

```
*****************************************************************
 ---->xcsConvertString
 !(00144)fromCCSID:1200 toCCSID:437
 !(00144)in length:32, out length:48

 ------>OpenConversion

 -------->xxxOpenConv

 ---------->xxxGetTable
 !(00144)About to open file 34B001B5

 <----------xxxGetTable  (rc= OK)

 <--------xxxOpenConv (rc= OK)

 <------OpenConversion (rc= OK)
 !(00144)translated out length:16

 <----xcsConvertString (rc= OK)
 !(00144)LSX: CharacterSet conversion from 1200 to 437, rc = 0

 <--LSX: ClassControl (rc= OK)
 *****************************************************************
   : : :
  Various lines missing for clarity
```

```
  : : :
************************************************************
************************************************************
 The final entry is the result of Notes shutting down
************************************************************

-->LSX: MQLSX_MessageProc
!(00144)LSX: LSX_MSG_TERMINATE received; clearing MQSession list

<--LSX: MQLSX_MessageProc (rc= OK)
```

When your MQLSX script fails

First Failure Symptom Report

Independently of the trace facility, for unexpected and internal errors, a First Failure Symptom report is produced.

This report is found in a file named GMQnnnnn.fdc, where nnnnn is the id of the Notes process that is running at the time. You find this file in the working directory from which you started Notes or the name of the path specified in the GMQ_TRACE_PATH environment variable.

Other sources of information

MQSeries provides various error logs and trace information, depending on the platform involved. See either the *MQSeries Problem Determination Guide* for MVS or the *MQSeries System Management Guide* for other platforms for more information.

Common pitfalls

There are some features that may catch you out when you least expect it, depending on your experience of using LotusScript and MQSeries. The following are those identified so far.

LotusScript events

LotusScript treats events differently within the Notes Client and the Notes Server environments.

In the Notes Client environment events are posted immediately, however within the Notes Server environment there is a delay in an event being posted. When you are using remote agents within Notes, this can give the impression that events are not always posted when you would expect them.

Windows NT and Windows 95

Check you are running all 16 bit versions or all 32 bit versions of Lotus Notes, MQSeries and the MQLSX. If not, reinstall the appropriate product.

Data conversion

Your program can fail whilst it is trying to convert data on a read or write. See Using Data Conversion for more information.

Reason codes

The following reason codes can occur in addition to those documented for the MQSeries MQI. For further codes, refer to the MQSeries documentation.

Reason code	Explanation
MQRC_LIBRARY_LOAD_ERROR (6000)	One or more of the MQSeries libraries could not be loaded. Check that all MQSeries libraries are in the correct search path on the system you are using. For example, on OS/2, make sure that the directories containing the MQSeries libraries are in LIBPATH.
MQRC_CLASS_LIBRARY_ERROR (6001)	One of the MQSeries class library calls returned an unexpected ReasonCode / CompletionCode. Check the First Failure Symptom Report for details. Take note of the last method/property and class being used and inform IBM Support of the problem.
MQRC_STRING_LENGTH_TOO_BIG (6002)	A **ReadString or WriteString** call tried to read a string of more than 32000 characters (16000 characters if you are running the Win16 version). Find the ReadString call in your application and correct the call.
MQRC_WRITE_VALUE_ERROR (6003)	A write method has detected a data overflow. Correct the data passed to the Write method to ensure it is within the acceptable boundaries for the data type in question.

Reason code	Explanation
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	An open object does not have the correct **OpenOptions** and requires one or more additional options. An implicit re-open is required but closure has been prevented. Set the **OpenOptions** explicitly to cover all eventualities so that implicit re-opening is not required. Closure has been prevented because the queue is open for exclusive input and closure would present a window of opportunity for others to potentially gain access to the queue.
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	An open object does not have the correct **OpenOptions** and requires one or more additional options. An implicit re-open is required but closure has been prevented. Set the **OpenOptions** explicitly to cover all eventualities so that implicit re-opening is not required. Closure has been prevented because the queue is a local queue of the definition type MQQDT_TEMPORARY_DYNAMIC , that would be destroyed by closure.
MQRC_ATTRIBUTE_LOCKED (6104)	An attempt has been made to change the value of an attribute of an object whilst that object is open. Certain attributes, such as **AlternateUserId**, cannot be changed whilst an object is open.
MQRC_CURSOR_NOT_VALID (6105)	The browse cursor for an open queue has been invalidated since it was last used by an implicit re-open. Set the **OpenOptions** explicitly to cover all eventualities so that implicit re-opening is not required.

Reason code	Explanation
MQRC_NULL_POINTER (6108)	A null pointer has been supplied where a non-null pointer is either required or implied. This denotes an internal consistency that should not occur.
MQRC_NO_CONNECTION_REFERENCE (6109)	The **MQQueue** object has lost its connection to the **MQQueueManager**. This will occur if the **MQQueueManager** is disconnected. Delete the **MQQueue** object.
MQRC_NO_BUFFER (6110)	No buffer is available. For an **MQMessage** object, one cannot be allocated, denoting an internal inconsistency in the object state that should not occur.
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	The length of the binary data is inconsistent with the length of the target attribute. Zero is a correct length for all attributes. 24 is the correct length for a **CorrelationId** and for a **MessageId**. 32 is the correct length for an **AccountingToken**. Some properties must be supplied in full, such as **CorrelationId**.
MQRC_INSUFFICIENT_BUFFER (6113)	There is insufficient buffer space available after the data pointer to accommodate the request. This could be because the buffer cannot be resized.
MQRC_INSUFFICIENT_DATA (6114)	There is insufficient data after the data pointer to accommodate the read request. Reduce the buffer to the correct size and read the data again.

Reason code	Explanation
MQRC_DATA_TRUNCATED (6115)	Data has been truncated when copying from one buffer to another. This could be because the target buffer cannot be resized, or because there is a problem addressing one or other buffer, or because a buffer is being downsized with a smaller replacement.
MQRC_ZERO_LENGTH (6116)	A zero length has been supplied where a positive length is either required or implied.
MQRC_NEGATIVE_LENGTH (6117)	A negative length has been supplied where a zero or positive length is required.
MQRC_NEGATIVE_OFFSET (6118)	A negative offset has been supplied where a zero or positive offset is required.
MQRC_INCONSISTENT_OBJECT_STATE (6120)	There is an inconsistency between this object, which is open, and the referenced **MQQueueManager** object, which is not connected.

Chapter 5 MQLSX Reference

This chapter describes the classes of the MQSeries link LotusScript Extension (MQLSX), developed for Lotus Notes Release 4. The classes enable you to write Notes applications that can access other applications running in your non-Notes environments, using MQSeries.

MQLSX objectives

The MQSeries LotusScript Extension (MQLSX) is designed to:

- Provide an infrastructure to enable you to develop applications that integrate your Lotus Notes environment with your traditional transaction system applications and their data.
- Give you access to all the functions and features of the MQSeries API, permitting full interconnectivity to other MQSeries platforms.
- Conform to the normal conventions expected of a LotusScript extension.
- Enable you to take advantage of the following benefits provided by MQSeries:
  - Access to enterprise application logic, not just the data
  - Access to a wide variety of platforms
  - Queued Entry into high throughput asynchronous messaging environments

LotusScript/MQI interface

This is supplied as a LotusScript Extension Module (LSX) that provides the following classes:

- MQGetMessageOptions Class
- MQMessage Class
- MQProcess Class
- MQPutMessageOptions Class
- MQQueue Class
- MQQueueManager Class
- MQSession Class

In addition the MQLSX provides:

- Predefined LotusScript constants (such as MQFMT_NONE) needed to use the classes. The constants are a subset of those defined in the MQSeries C header files (cmqc*.h) with some additional MQLSX Reason codes.

About MQLSX classes

This information should be read in conjunction with the *MQSeries Application Programming Reference* manual.

There are MQLSX classes called MQGetMessageOptions Class , MQMessage Class , MQProcess Class , MQPutMessageOptions Class , MQQueue Class , MQQueueManager Class , and MQSession Class .

The MQSession Class provides a root object that contains the status of the last action performed on any of the MQLSX objects, see Error handling for more information.

The MQQueueManager, MQQueue, and MQProcess classes provide access to the underlying MQSeries objects. Methods or property accesses against these classes will in general result in calls being made across the MQSeries API.

The MQMessage, MQPutMessageOptions, and MQGetMessageOptions classes encapsulate the MQMD, MQPMO, and MQGMO data structures respectively, and are used to help you put messages to queues and retrieve messages from them.

Parameter passing

Parameters on method invocations are all passed by value, except where that parameter is an object, in which case it is a reference that is passed.

The class definitions provided list the Data Type for each parameter or property. If the LotusScript variable used is not of the required type, then the value will be automatically converted to/from the required type - providing such a conversion is possible. This follows standard LotusScript conversion rules.

Many of the methods take fixed length string parameters, or return a fixed length character string. The conversion rules are as follows:

- If the user supplies a fixed length string of the wrong length, as an input parameter or a return value, then the value is truncated or padded with trailing spaces as required.
- If the user supplies a variable length string of the wrong length as an input parameter, then the value is truncated or padded with trailing spaces.
- If the user supplies a variable length string of the wrong length as a return value, then the string is adjusted to the required length (since returning a value destroys the previous value in the string anyway).

Strings provided as input parameters may contain embedded Nulls.

Object access methods

These methods do not relate directly to any single MQSeries call. Each of these methods create an object in which reference information is then held, followed by connecting or opening an MQSeries object:

- when a connection is made to a queue manager or a process object, it holds the 'object handle' generated by MQSeries.
- when a queue is opened, it holds the 'connection handle' generated by MQSeries.

These MQSeries attributes are explained in the *MQSeries Application Programming Reference* manual.

Errors

Syntactic errors on parameter passing are detected by LotusScript at compile time and runtime errors can be trapped using On Error.

The MQSeries LotusScript classes all contain two special read-only properties - ReasonCode and CompletionCode. These can be read at any time.

An attempt to access any other property, or to issue any method call could potentially generate an error.

If a property set or method invocation succeeds, then the owning object's ReasonCode and CompletionCode fields are set to MQRC_NONE and MQCC_OK respectively.

If the property access or method invocation does not succeed then appropriate error or warning codes are set in these fields.

MQSession Class

This is the root class for the MQSeries link LotusScript Extension.

There is always one and only one MQSession object per LotusScript instance.

An attempt to create a second object will create a second reference to the original object.

Properties:

- CompletionCode Property
- ReasonCode Property

Methods:

- AccessQueueManager Method
- ClearErrorCodes Method

LotusScript Events:

- Mqerror
- Mqwarning

Creation:

New - creates a new MQSession object reference.

Syntax:

Dim *mqsess* **As New MQSession** or

Set *mqsess* = **New MQSession**

CompletionCode Property

Read-only. Returns the MQSeries completion code set by the most recent method or property access issued against any MQSeries object.

It is reset to MQCC_OK when a call, other than a property Get, is made successfully against any MQLSX object.

An error event handler can inspect this property to diagnose the error, without having to know which object was involved.

Defined in
MQSession Class

Data Type:
Long

Legal Values:
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
To get: *completioncode& = MQSession*.**CompletionCode**

ReasonCode Property

Read-only. Returns the reason code set by the most recent method or property access issued against any MQSeries object.

An error event handler can inspect this property to diagnose the error, without having to know which object was involved.

Defined in:
MQSession Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual and the additional MQLSX values listed under Reason codes.

Syntax:
To get: *reasoncode& = MQSession*.**ReasonCode**

AccessQueueManager Method

Creates a new MQQueueManager object and connects it to a real Queue Manager via the MQSeries client or server.

If successful it sets the MQQueueManager's ConnectionStatus to TRUE.

A Queue Manager can be connected to by at most one MQQueueManager object per LotusScript instance.

If the connection to the object fails, an error event is raised, the object's ReasonCode and CompletionCode are set, and the MQSession object's ReasonCode and CompletionCode are set.

Defined in:
MQSession Class

Syntax:
Set *qm* = *MQSession*.**AccessQueueManager**(*Name$*)

Parameter:
Name$

String.Name of Queue Manager to be connected to.

ClearErrorCodes Method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE.

Defined in:
MQSession Class

Syntax:
Call *MQSession*.**ClearErrorCodes**

MQQueueManager Class

This represents a connection to a queue manager. The queue manager may be running locally (an MQSeries server) or remotely with access provided by the MQSeries client. An application must create an object of this class and connect it to a queue manager. When an object of this class is destroyed it is automatically disconnected from its queue manager.

Containment:
MQProcess and MQQueue objects are associated with this class.

Properties:

- AlternateUserId Property
- AuthorityEvent Property
- CharacterSet Property
- CommandInputQueueName Property
- CommandLevel Property
- CompletionCode Property
- ConnectionStatus Property
- DeadLetterQueueName Property
- DefaultTransmissionQueueName Property
- Description Property
- InhibitEvent Property
- LocalEvent Property
- MaximumHandles Property
- MaximumMessageLength Property
- MaximumPriority Property
- MaximumUncommittedMessages Property
- Name Property
- PerformanceEvent Property
- Platform Property
- ReasonCode Property
- RemoteEvent Property
- StartStopEvent Property
- SyncPointAvailability Property
- TriggerInterval Property

Methods:

- AccessProcess Method
- AccessQueue Method
- Backout Method
- ClearErrorCodes Method
- Commit Method
- Connect Method
- Disconnect Method

LotusScript Events:

- Mqerror
- Mqwarning

Creation:

New - creates a new MQQueueManager object. If you do not want to connect to the default queue manager, you must set the name of the newly created queue manager before accessing any of the properties, other than those listed under Property Access. If you do **not** name the queue manager and access one of the properties outside the list, for example the Description property, the MQLSX will attempt to implicitly connect to the default queue manager.

A new MQQueueManager object can also be created by using the AccessQueueManager method on the MQSession object.

Syntax:

Dim *qmname* **As New MQQueueManager** or

Set *qmname* = **New MQQueueManager**

Property Access:

The following properties can be accessed at any time

- AlternateUserId
- CompletionCode
- ConnectionStatus
- ReasonCode

The remaining properties can be accessed only if the object is connected to a queue manager, and the userid is authorised for inquire against that queue manager. If an alternate userid is set and the current userid is authorised to use it, then the alternate userid is checked for authorisation for inquire instead.

If these conditions do not apply, the MQLSX will attempt to connect to the Queue Manager and open it for inquire automatically. If this is unsuccessful the call will set a CompletionCode of MQCC_FAILED and one of the following ReasonCodes:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_QMGR_NAME_ERROR
- MQRC_QMGR_NOT_AVAILABLE

The Backout, Commit, Connect, and Disconnect methods set errors matching those set by the MQI calls MQBACK, MQCMIT, MQCONN, and MQDISC respectively.

AlternateUserId Property

Read-Write. The alternate userid to be used to validate access to the Queue Manager attributes.

This property may not be set if ConnectionStatus is TRUE.

Defined in:
MQQueueManager Class

Data Type:
String of 12 characters

Syntax:
To get: *altuser$ = MQQueueManager*.**AlternateUserId**

To set: *MQQueueManager*.**AlternateUserId** = *altuser$*

AuthorityEvent Property

Read-only. The MQI AuthorityEvent attribute. Controls whether authorization events are generated. If the value is set to MQEVR_ENABLED, an event is generated when unauthorized access to the queue manager is attempted.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax:
To get: *authevent& = MQQueueManager*.**AuthorityEvent**

CharacterSet Property

Read-only. The MQI CodedCharSetId attribute. This identifies the character set used by the queue manager for all character strings defined in the MQI. This includes the names of objects, and queue creation date and time. It does not include application data carried in the message.

Defined in:
MQQueueManager Class

Data Type:
Long

Syntax:
To get: *characterset& = MQQueueManager*.**CharacterSet**

CommandInputQueueName Property

Read-only. The MQI CommandInputQName attribute. This is the name of the queue to which applications, if authorized, can send commands.

Defined in:
MQQueueManager Class

Data Type:
String of 48 characters

Syntax:
To get: *commandinputqname$ = MQQueueManager*.**CommandInputQueueName**

CommandLevel Property

Read-only. Returns the version and level of the MQSeries queue manager implementation (MQI CommandLevel attribute)

Defined in:
MQQueueManager Class

Data Type:
Long

Syntax:
To get: *level& = MQQueueManager*.**CommandLevel**

CompletionCode Property

Read-only. Returns the completion code set by the last method or property access issued against the object.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
To get: *completioncode& = MQQueueManager*.**CompletionCode**

ConnectionStatus Property

Read-only. Indicates if the object is connected to its queue manager or not.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- TRUE (-1)
- FALSE

Syntax:
To get: *status& = MQQueueManager*.**ConnectionStatus**

DeadLetterQueueName Property

Read-only. The MQI DeadLetterQName attribute. This is the name of a queue defined on the local queue manager. Messages are sent to this queue if they cannot be routed to their correct destination.

Defined in:
MQQueueManager Class

Data Type:
String of 48 characters

Syntax:
To get: *dlqname$ = MQQueueManager*.**DeadLetterQueueName**

DefaultTransmissionQueueName Property

Read-only. The MQI DefXmitQName attribute. This is the name of the transmission queue, that holds messages prior to them being sent to another queue manager, if no other transmission queue is specified.

Defined in:
MQQueueManager Class

Data Type:
String of 48 characters

Syntax:
To get: *defxmitqname$ = MQQueueManager*.**DefaultTransmissionQueueName**

Description Property

Read-only. The MQI QMgrDesc attribute. Use this property to hold a descriptive commentary. The content of this property has no significance to the queue manager. It must **not** contain any null characters - it is padded with blanks when necessary.

Defined in:
MQQueueManager Class

Data Type:
String of 64 characters

Syntax:
To get: *description$ = MQQueueManager*.**Description**

InhibitEvent Property

Read-only. The MQI InhibitEvent attribute. This determines whether or not Inhibit Get and Inhibit Put events are generated.

If it is set to MQEVR_ENABLED, an event is generated when the Get method is used against a queue that prevents any messages being removed from it. Similarly, an event is generated when the Put method is used against a queue that prevents any messages being placed on it.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax:
To get: *inhibevent& = MQQueueManager*.**InhibitEvent**

IsConnected Property

Read-only. Indicates if the object is connected to its queue manager or not. This property is exactly the same as the ConnectStatus property.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- TRUE (-1)
- FALSE

Syntax:
To get: *status& = MQQueueManager*.**IsConnected**

LocalEvent Property

Read-only. The MQI LocalEvent attribute. This determines whether or not local events are generated. A local event is generated when an application is unable to access a local queue. For more information see the *MQSeries Programmable System Management* manual.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax:
To get: *localevent& = MQQueueManager*.**LocalEvent**

MaximumHandles Property

Read-only. The MQI MaxHandles attribute. This is the maximum number of open handles that any one task can have at the same time.

Defined in:
MQQueueManager Class

Data Type:
Long

Syntax:
To get: *maxhandles& = MQQueueManager*.**MaximumHandles**

MaximumMessageLength Property

Read-only. The MQI MaxMsgLength Queue Manager attribute. This is the maximum length of a message, in bytes, that the queue manager can handle.

Defined in:
MQQueueManager Class

Data Type:
Long

Syntax:
To get: *maxmessagelength& = MQQueueManager*.**MaximumMessageLength**

MaximumPriority Property

Read-only. The MQI MaxPriority attribute. This is the maximum message priority supported by the queue manager, zero being the lowest.

Defined in:
MQQueueManager Class

Data Type:
Long

Syntax:
To get: *maxpriority& = MQQueueManager*.**MaximumPriority**

MaximumUncommittedMessages Property

Read-only. The MQI MaxUncommittedMsgs attribute. This is the maximum number of uncommitted messages that can exist within a unit of work.

Defined in:
MQQueueManager Class

Data Type:
Long

Syntax:
To get: *maxuncommitted& = MQQueueManager*.**MaximumUncommittedMessages**

Name Property

Read-write. The MQI QMgrName attribute. This is the name of the queue manager to which an application is connected.

This property cannot be written once the MQQueueManager is Connected.

Defined in:
MQQueueManager Class

Data Type:
String of 48 characters

Syntax:
To get: *name$ = MQQueueManager*.**Name**

To set: *MQQueueManager*.**Name** = *name$*

PerformanceEvent Property

Read-only. The MQI PerformanceEvent attribute. This determines whether or not performance events are generated. For more information see the *MQSeries Programmable System Management* manual.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax:
To get: *perfevent& = MQQueueManager*.**PerformanceEvent**

Platform Property

Read-only. The MQI Platform attribute. This is the platform that the queue manager is running on.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQPL_AIX
- MQPL_MVS
- MQPL_OS2
- MQPL_OS400
- MQPL_UNIX
- MQPL_WINDOWS_NT

Syntax:
To get: *platform& = MQQueueManager*.**Platform**

ReasonCode Property

Read-only. Returns the reason code set by the last method or property access issued against the object.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual.

Syntax:
To get: *reasoncode& = MQQueueManager*.**ReasonCode**

RemoteEvent Property

Read-only. The MQI RemoteEvent attribute. This controls whether or not remote events are generated. A remote event is raised when an application cannot access a queue on another queue manager. For more information see the *MQSeries Programmable System Management* manual.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax:
To get: *remoteevent& = MQQueueManager*.**RemoteEvent**

StartStopEvent Property

Read-only. The MQI StartStopEvent attribute. This controls whether or not start and stop events are raised. A start event is raised when a queue manager is started. A stop event is raised when a request is made for a queue manager to stop or quiesce. For more information see the *MQSeries Programmable System Management* manual.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax:
To get: *strstpevent& = MQQueueManager*.**StartStopEvent**

SyncPointAvailability Property

Read-only. The MQI SyncPoint attribute. This indicates whether or not the queue manager supports units or work and syncpointing when you use the Put and Get methods in the MQQueue Class.

Defined in:
MQQueueManager Class

Data Type:
Long

Legal Values:
- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

Syntax:
To get: *syncpointavailability& = MQQueueManager*.**SyncPointAvailability**

TriggerInterval Property

Read-only. The MQI TriggerInterval attribute. This is a time interval (in milliseconds) used to restrict the number of trigger messages that are generated. This is only relevant when the value of the MQQueue Class property, TriggerType, is MQTT_FIRST.

Defined in:
MQQueueManager Class

Data Type:
Long

Syntax:
To get: *trigint& = MQQueueManager*.**TriggerInterval**

AccessProcess Method

Creates a new MQProcess object and associates it with this MQQueueManager object. It sets the name property and the alternate userid of the MQProcess object, and attempts to open it for inquire.

If you do not wish to use alternate userid validation, set this parameter to "".

Defined in:
MQQueueManager Class

Syntax:
Set *process = MQQueueManager*.**AccessProcess**(*Name$*, *AlternateUserId$*)

Parameters:
Name$

String. Name of MQSeries Process definition.

AlternateUserId$

String. The alternate userid to validate access to the process object.

AccessQueue Method

Creates a new MQQueue object and associates it with this MQQueueManager object. It sets the Name, OpenOptions, DynamicQueueName and AlternateUserId properties of the MQQueue object to the values provided, and attempts to open it.

If the open is unsuccessful the call fails. An error event is raised against the object, the object's ReasonCode and CompletionCode are set, and the MQSession ReasonCode and CompletionCode are set.

All parameters are mandatory, but DynamicQueueName, QueueManagerName, and AlternateUserId may be set to the default of "" if they are not needed.

OpenOptions control the operations that can be performed on a queue. The OpenOption MQOO_INQUIRE is optional, it is automatically added to the options your application supplies. The options are listed in the *MQSeries Application Programming Reference* manual, under the MQOPEN call. They include:

MQOO_INPUT_SHARED - allowing more than one application to get messages from this queue, where they also open the queue using the MQOO_INPUT_SHARED option.

MQOO_INPUT_EXCLUSIVE - gives your application exclusive access to a queue

MQOO_SET - enables you to change the attributes of the queue after you have opened it.

Set the QueueManagerName to "" if the queue to be opened is local. Otherwise it should be set to the name of the remote queue manager that owns the queue, and an attempt is made to open a local definition of the remote queue.

See the *MQSeries Application Programming Guide* for more information on remote queue name resolution and queue manager aliasing.

If the Name property is set to a model queue name, specify the name of the dynamic queue to be created in the *DynamicQueueName$* parameter. If the value provided in the *DynamicQueueName$* parameter is "", the value set into the queue object and used on the open call is "AMQ.*".

See the *MQSeries Application Programming Guide* for more information on naming dynamic queues.

Defined in:

MQQueueManager Class

Syntax:
Set *queue* = *MQQueueManager*.**AccessQueue**(*Name$*, *OpenOptions&*, *QueueManagerName$*, *DynamicQueueName$*, *AlternateUserId$*)

Parameter:
Name$

String. Name of MQSeries queue.

OpenOptions&

Long. Options to be used when queue is opened.

See the *MQSeries Application Programming Referen*ce manual for more information.

QueueManagerName$

String. Name of the queue manager that owns the queue to be opened. A value of "" implies the queue manager is local.

DynamicQueueName$

String. The name assigned to the dynamic queue at the time the queue is opened when the Name$ parameter is specified as a model queue.

AlternateUserId$

String. The alternate userid used to validate access when opening the queue.

Backout Method

Backs out any uncommitted message puts and gets that have occurred as part of a unit of work since the last syncpoint.

Defined in:
MQQueueManager Class

Syntax:
Call *MQQueueManager*.**Backout**

ClearErrorCodes Method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQQueueManager Class and the MQSession Class.

Defined in:
MQQueueManager Class

Syntax:
Call *MQQueueManager*.**ClearErrorCodes**

Commit Method

Commits any message puts and gets that have occurred as part of a unit of work since the last syncpoint.

Defined in:
MQQueueManager Class

Syntax:
Call *MQQueueManager*.**Commit**

Connect Method

Connects the MQQueueManager object to a real Queue Manager via the MQSeries client or server.

Sets ConnectionStatus to TRUE.

A maximum of one MQQueueManager object per LotusScript instance is allowed to connect to a Queue Manager.

Defined in:
MQQueueManager Class

Syntax:
Call *MQQueueManager*.**Connect**

Disconnect Method

Disconnects the MQQueueManager object from the Queue Manager.

Sets ConnectionStatus to FALSE.

All Queue objects associated withthe MQQueueManager object are made unusable and cannot be re-opened.

Any uncommitted changes (message puts and gets) are committed.

Defined in:
MQQueueManager Class

Syntax:
Call *MQQueueManager*.**Disconnect**

MQQueue Class

This represents a connection to an MQSeries Queue. This connection is provided by an associated MQQueueManager object. When an object of this class is destroyed it is automatically closed.

Containment:
Contained by the MQQueueManager Class .

Properties:

- AlternateUserId Property
- BackoutRequeueName Property
- BackoutThreshold Property
- BaseQueueName Property
- CloseOptions Property
- CompletionCode Property
- CreationDateTime Property
- CurrentDepth Property
- DefaultInputOpenOption Property
- DefaultPersistence Property
- DefaultPriority Property
- DefinitionType Property
- DepthHighEvent Property
- DepthHighLimit Property
- DepthLowEvent Property
- DepthLowLimit Property
- DepthMaximumEvent Property
- Description Property
- HardenGetBackout Property
- InhibitGet Property
- InhibitPut Property
- InitiationQueueName Property
- MaximumDepth Property
- MaximumMessageLength Property
- MessageDeliverySequence Property
- Name Property

- OpenInputCount Property
- OpenOptions Property
- OpenOutputCount Property
- OpenStatus Property
- ProcessName Property
- QueueType Property
- ReasonCode Property
- RemoteQueueManagerName Property
- RemoteQueueName Property
- RetentionInterval Property
- Scope Property
- ServiceInterval Property
- ServiceIntervalEvent Property
- Shareability Property
- TransmissionQueueName Property
- TriggerControl Property
- TriggerData Property
- TriggerDepth Property
- TriggerMessagePriority Property
- TriggerType Property
- Usage Property

Methods:

- ClearErrorCodes Method
- Get Method
- Put Method

LotusScript Events:

- Mqerror
- Mqwarning

Creation:

Use the AccessQueue Method from the MQQueueManager Class.

Property Access:
If the queue object is **not** connected to a queue manager, you can read the following properties:

- AlternateUserId
- CompletionCode
- Name
- OpenOptions
- OpenStatus
- ReasonCode

and you can write to:

- CloseOptions

If the queue object **is** connected to a queue manager, you can read all the properties.

Note Reading a property not listed above, such as TriggerControl, will cause an implicit connection to the underlying queue manager.

Queue Attribute properties:
Properties not listed in the previous section are all attributes of the underlying MQSeries Queue. They can be accessed only if the object is connected to a queue manager, and the user's userid is authorised for Inquire or Set against that queue. If an alternate userid is set and the current userid is authorised to use it, then the alternate userid is checked for authorisation instead.

The property must be an appropriate property for the given QueueType (see the *MQSeries Application Programming Reference* manual).

If these conditions do not apply, then the property access will set a CompletionCode of MQCC_FAILED and one of the following ReasonCodes

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_QUEUE_MGR_NAME_ERROR
- MQRC_QUEUE_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode is MQCC_WARNING)

Opening a queue:
The only way to create an MQQueue object is by using the MQQueueManager AccessQueue method, unless an implicit connection has taken place. The MQQueue object remains open (OpenStatus=TRUE) until it is deleted. The value of the MQQueue CloseOptions property controls the behaviour of the close operation that takes place when the MQQueue object is deleted.

The MQQueueManager AccessQueue method opens the queue using the OpenOptions parameter with the automatic addition of the MQOO_INQUIRE value. MQSeries validates the OpenOptions against the user authorisation as part of the open queue process.

Tip Check that the OpenOptions are appropriate for the actions to be performed on a queue (put, get, etc.). In certain circumstances, if the OpenOptions are insufficient, the MQLSX attempts to close and reopen the queue with additional OpenOptions:

- A property get is always allowed, as the OpenOptions always include MQOO_INQUIRE.
- If a queue property set is attempted and the OpenOptions do not include MQOO_SET, the queue is closed and reopened with MQOO_SET added to the OpenOptions.
- If a put is attempted and the OpenOptions do not include MQOO_OUTPUT, the queue is closed and reopened with MQOO_OUTPUT added to the OpenOptions.
- If a put is attempted and the MQPMO options include MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT, but the OpenOptions do not permit this, the queue is closed and reopened with MQOO_SET_IDENTITY_CONTEXT or MQOO_SET_ALL_CONTEXT, as appropriate.
- If a get is attempted and the OpenOptions do not include any of the MQOO_INPUT_* options, the queue is closed and reopened with MQOO_INPUT_AS_Q_DEF added to the OpenOptions.
- If a get is attempted and the MQGetMessageOptions request MQGMO_BROWSE_*, but the OpenOptions do not include MQOO_BROWSE, the queue is closed and reopened with MQOO_BROWSE added to the OpenOptions.

Note A queue will not be reopened if it is a temporary dynamic queue, or the queue is already open for exclusive input. In these cases, the method call to reopen the queue will fail with either MQRC_REOPEN_TEMPORARY_Q_ERROR or MQRC_REOPEN_EXCL_INPUT_ERROR, as appropriate.

If a queue is opened for Browse and a reopen occurred for one of the reasons listed, subsequent attempts to do a get with one of the following:

- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MSG_UNDER_CURSOR

will be rejected with MQRC_CURSOR_NOT_VALID.

AlternateUserId Property

Read-only. The alternate userid used to validate access to the Queue when it was opened.

Defined in:
MQQueue Class

Data Type:
String of 12 characters

Syntax:
To get: *altuser$ = MQQueue.***AlternateUserId**

BackoutRequeueName Property

Read-only. The MQI BackOutRequeueQName attribute. This is the name of a queue that an application can use to put a message that is causing a unit of work to fail.

The message concerned is identified by the value of its BackoutCount property, which is incremented each time it causes a unit of work to fail. The application can test for the value held in the BackoutCount property to be greater than the value held in the BackoutThreshold property, and move the message to the queue named in the BackoutRequeueName property.

Defined in:
MQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *backoutrequeuename$ = MQQueue.***BackoutRequeueName**

BackoutThreshold Property

Read-only. The MQI BackoutThreshold attribute. This property is provided for an application to use. It is used in conjunction with the BackoutRequeueName and BackoutCount (this is a property of the MQSeries message). The value held in this property is available for comparison with the BackoutCount value, by an application, to enable the application to remove a problem message that is part of a unit of work.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
See the *MQSeries Programming Reference* manual

Syntax:
To get: *backoutthreshold& =MQQueue.***BackoutThreshold**

BaseQueueName Property

Read-only. The queue name to which the alias resolves.

Valid only for alias queues.

Defined in:
MQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *baseqname$ = MQQueue.***BaseQueueName**

CloseOptions Property

Read-Write. Options used to control what happens when the Queue is closed.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

MQCO_DELETE and MQCO_DELETE_PURGE are valid only for dynamic queues.

Syntax:
To get: *closeopt&* = *MQQueue*.**CloseOptions**

To set: *MQQueue*.**CloseOptions** = *closeopt&*

CompletionCode Property

Read-only. Returns the completion code set by the last method or property access issued against the object.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
To get: *completioncode&* = *MQQueue*.**CompletionCode**

CreationDateTime Property

Read-only. Date and time this queue was created.

Defined in:
MQQueue Class

Data Type:
Variant of type 7 (date/time) or EMPTY

Syntax:
To get: *datetime = MQQueue*.**CreationDateTime**

CurrentDepth Property

Read-only. The MQI CurrentQDepth attribute. The number of messages currently on the queue.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *currentdepth& = MQQueue*.**CurrentDepth**

DefaultInputOpenOption Property

Read-only. The MQI DefInputOpenOption attribute. This controls the way that the queue is opened if the OpenOption specifies MQOO_INPUT_AS_Q_DEF.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQOO_INPUT_EXCLUSIVE

MQOO_INPUT_SHARED

Syntax:
To get: *defaultinop& = MQQueue*.**DefaultInputOpenOption**

DefaultPersistence Property

Read-only. The MQI DefPersistence attribute. The default persistence for messages on a queue.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *defpersistence& = MQQueue.***DefaultPersistence**

DefaultPriority Property

Read-only. The MQI DefPriority attribute. The default priority for messages on a queue.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *defpriority& = MQQueue.***DefaultPriority**

DefinitionType Property

Read-only. The MQI DefinitionType attribute. This describes the type of queue. A model queue is needed to create a dynamic queue. A queue of type MQQDT_PREDEFINED is a permanent queue.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:

MQQDT_PREDEFINED

MQQDT_PERMANENT_DYNAMIC

MQQDT_TEMPORARY_DYNAMIC

Syntax:
To get: *deftype& = MQQueue.***DefinitionType**

DepthHighEvent Property

Read-only. The MQI QDepthHighEvent attribute.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQEVR_DISABLED

MQEVR_ENABLED

Syntax:
To get: *depthhighevent& = MQQueue.***DepthHighEvent**

DepthHighLimit Property

Read-only. The MQI QDepthHighLimit attribute.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *depthhighlimit& = MQQueue.***DepthHighLimit**

DepthLowEvent Property

Read-only. The MQI QDepthLowEvent attribute.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQEVR_DISABLED

MQEVR_ENABLED

Syntax:
To get: *depthlowevent& = MQQueue.***DepthLowEvent**

DepthLowLimit Property

Read-only. The MQI QDepthLowLimit attribute.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *depthlowlimit& = MQQueue.***DepthLowLimit**

DepthMaximumEvent Property

Read-only. The MQI QDepthMaxEvent attribute.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQEVR_DISABLED

MQEVR_ENABLED

Syntax:
To get: *depthmaximumevent& = MQQueue.***DepthMaximumEvent**

Description Property

Read-only. A description of the queue.

Defined in:
MQQueue Class

Data Type:
String of 64 characters

Syntax:
To get: *description$ = MQQueue.***Description**

HardenGetBackout Property

Read-only. Whether to maintain an accurate backout count.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQQA_BACKOUT_HARDENED

MQQA_BACKOUT_NOT HARDENED

Syntax:
To get: *hardengetback& = MQQueue*.**HardenGetBackout**

InhibitGet Property

Read-write. The MQI InhibitGet attribute.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQQA_GET_INHIBITED

MQQA_GET_ALLOWED

Syntax:
To get: *getstatus& = MQQueue*.**InhibitGet**

To set: *MQQueue*.**InhibitGet** = *getstatus&*

InhibitPut Property

Read-write. The MQI InhibitPut attribute.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQQA_PUT_INHIBITED

MQQA_PUT_ALLOWED

Syntax:
To get: *putstatus&* = *MQQueue*.**InhibitPut**

To set: *MQQueue*.**InhibitPut** = *putstatus&*

InitiationQueueName Property

Read-only. Name of initiation queue.

Defined in:
MQQueue ClassMQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *initqname$* = *MQQueue*.**InitiationQueueName**

IsOpen Property

Read-only. Indicates if the queue is Opened or not. Initial value is TRUE after AccessQueue method. This property is exactly the same as the OpenStatus property.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
- TRUE (-1)
- FALSE

Syntax:
To get: *status&* = *MQQueue*.**IsOpen**

MaximumDepth Property

Read-only. Maximum queue depth.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *maxdepth&* = *MQQueue*.**MaximumDepth**

MaximumMessageLength Property

Read-only. The MQI MaxMsgLength attribute. This is the maximum length message, in bytes, that this queue will accept.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *maxmlength&* = *MQQueue*.**MaximumMessageLength**

MessageDeliverySequence Property

Read-only. Message delivery sequence.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQMDS_PRIORITY

MQMDS_FIFO

Syntax:
To get: *messdelseq& = MQQueue*.**MessageDeliverySequence**

Name Property

Read-only. The MQI QName attribute. This is the name of a queue defined on the local queue manager.

Defined in:
MQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *name$ = MQQueue*.**Name**

OpenInputCount Property

Read-only. Number of opens for input.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *openincount& = MQQueue*.**OpenInputCount**

OpenOptions Property

Read-only. These are the options for which the queue is initially opened (specified in the AccessQueue method in the MQQueueManager Class). However, these may be extended if an implicit re-open is performed.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual, under the MQOPEN call

Syntax:
To get: *openopt&* = *Queue*.**OpenOptions**

OpenOutputCount Property

Read-only. Number of opens for output.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *openoutcount&* = *MQQueue*.**OpenOutputCount**

OpenStatus Property

Read-only. Indicates if the queue is Opened or not. Initial value is TRUE after AccessQueue method.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
- TRUE (-1)
- FALSE

Syntax:
To get: *status&* = *MQQueue*.**OpenStatus**

ProcessName Property

Read-only. The MQI ProcessName attribute.

Defined in:
MQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *procname$* **=** *MQQueue*.**ProcessName**

QueueType Property

Read-only. The MQI QType attribute.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQQT_ALIAS

MQQT_LOCAL

MQQT_MODEL

MQQT_REMOTE

Syntax:
To get: *queuetype& = MQQueue*.**QueueType**

ReasonCode Property

Read-only. Returns the reason code set by the last method or property access issued against the object.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual.

Syntax:
To get: *reasoncode& = Queue*.**ReasonCode**

RemoteQueueManagerName Property

Read-only. Name of remote queue manager.

Valid for remote queues only.

Defined in:
MQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *remqmanname$ = MQQueue*.**RemoteQueueManagerName**

RemoteQueueName Property

Read-only. The name of the queue as it is known on the remote queue manager.

Valid for remote queues only.

Defined in:
MQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *remqname$ = MQQueue*.**RemoteQueueName**

RetentionInterval Property

Read-only. The period of time for which the queue should be retained.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *retinterval& = MQQueue*.**RetentionInterval**

Scope Property

Read-only. Controls whether an entry for this queue also exists in a cell directory.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQSCO_Q_MGR

MQSCO_CELL

Syntax:
To get: *scope&* = *MQQueue*.**Scope**

ServiceInterval Property

Read-only. The MQI QServiceInterval attribute.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *serviceinterval&* = *MQQueue.***ServiceInterval**

ServiceIntervalEvent Property

Read-only. The MQI QServiceIntervalEvent attribute.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQQSIE_HIGH

MQQSIE_OK

MQQSIE_NONE

Syntax:
To get: *serviceintervalevent& = MQQueue*.**ServiceIntervalEvent**

Shareability Property

Read-only. Queue shareability.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQQA_SHAREABLE

MQQA_NOT SHAREABLE

Syntax:
To get: *shareability& = MQQueue*.**Shareability**

TransmissionQueueName Property

Read-only. Transmission queue name.

Valid for remote queues only.

Defined in:
MQQueue Class

Data Type:
String of 48 characters

Syntax:
To get: *transqname$* = *MQQueue*.**TransmissionQueueName**

TriggerControl Property

Read-write. Trigger control.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQTC_OFF

MQTC_ON

Syntax:
To get: *trigcontrol&* = *MQQueue*.**TriggerControl**

To set: *MQQueue*.**TriggerControl** = *trigcontrol&*

TriggerData Property

Read-write. Trigger data.

Defined in:
MQQueue Class

Data Type:
String of 64 characters

Syntax:
To get: *trigdata$* = *MQQueue*.**TriggerData**

To set: *MQQueue*.**TriggerData** = *trigdata$*

TriggerDepth Property

Read-write. The number of messages that have to be on the queue before a trigger message is written.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *trigdepth&* = *MQQueue*.**TriggerDepth**

To set: *MQQueue*.**TriggerDepth** = *trigdepth&*

TriggerMessagePriority Property

Read-write. Threshold message priority for triggers.

Defined in:
MQQueue Class

Data Type:
Long

Syntax:
To get: *trigmesspriority&* = *MQQueue*.**TriggerMessagePriority**

To set: *MQQueue*.**TriggerMessagePriority** = *trigmesspriority&*

TriggerType Property

Read-write. Trigger type.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQTT_NONE

MQTT_FIRST

MQTT_EVERY

MQTT_DEPTH

Syntax:
To get: *trigtype&* = *MQQueue*.**TriggerType**

To set: *MQQueue*.**TriggerType** = *Trigtype&*

Usage Property

Read-only. Indicates what the queue is used for.

Defined in:
MQQueue Class

Data Type:
Long

Legal Values:
MQUS_NORMAL

MQUS_TRANSMISSION

Syntax:
To get: *usage&* = *MQQueue*.**Usage**

ClearErrorCodes Method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQQueue Class and the MQSession Class.

Defined in:
MQQueue Class

Syntax:
Call *MQQueue*.**ClearErrorCodes**

Put Method

Places a message onto the queue.

This method takes an MQMessage object as a parameter. The Message Descriptor (MQMD) properties of this object may be altered as a result of this method. The values they have immediately after this method has run are the values that were put onto the MQSeries queue.

Modifications to the MQMessage object after the Put has completed do not affect the actual message on the MQSeries queue.

Defined in:
MQQueue Class

Syntax:
Call *MQQueue*.**Put**(*Message , PutMsgOptions&)*

Parameters:
Message

MQMessage Object representing message to be put.

PutMsgOptions&

MQPutMessageOptions object containing options to control the put operation.

Get Method

Retrieves a message from the queue.

This method takes an MQMessage object as a parameter. It uses some of the fields in this object's MQMD as input parameters - in particular the MessageId(MsgId) and CorrelationId(CorrelId), so it is important to ensure that these are set as required (see the *MQSeries Application Programming Reference* manual for details).

If the method succeeds then the MQMD and Message Data portions of the MQMessage object are completely replaced with the MQMD and Message Data from the incoming message. The MQMessage control properties are set as follows

- MessageLength is set to length of the MQSeries Message
- DataLength is set to length of the MQSeries Message
- DataOffset is set to zero.

Defined in:
MQQueue Class

Syntax:
Call *MQQueue*.**Get**(*Message , GetMsgOptions&*)

Parameters:
MQMessage

MQMessage Object representing message to be retrieved.

GetMsgOptions&

MQGetMessageOptions object to control the get operation.

MQMessage Class

This class represents an MQSeries message. It includes properties to encapsulate the MQSeries message descriptor (MQMD), and provides a buffer to hold the application-defined message data.

The class includes methods (Write methods) to copy data from a LotusScript application to an MQMessage object and similarly methods (Read methods) to copy data from an MQMessage object to a LotusScript application. The class manages the allocation and deallocation of memory for the buffer automatically. The application does not have to declare the size of the buffer when an MQMessage object is created as the buffer grows to accommodate data written to it.

Note You will not be able to place a message onto an MQSeries queue if the buffer size exceeds the MaximumMessageLength property of that queue.

Once it has been constructed an MQMessage Object may be Put onto an MQSeries queue using the MQQueue.Put method. This method takes a copy of the MQMD and message data portions of the object and places that copy on the queue - so the application may modify or delete an MQMessage object after the Put, without affecting the message on the MQSeries queue. The queue manager may adjust some of fields in the MQMD when it copies the message on to the MQSeries queue. These adjustments are made to the copy of the MQMD held in the MQMessage object so that, unless subsequently modified, the MQMessage object is a true copy of what actually went onto the MQSeries queue.

An incoming message may be read into an MQMessage Object using the MQQueue.Get method. This replaces any MQMD or message data that may already have been in the MQMessage object with values from the incoming message, adjusting the size of the MQMessage object's data buffer to match the size of the incoming message data.

Containment:

Messages are contained by the MQSession Class.

Properties:

The control properties are:

- CompletionCode Property
- DataLength Property
- DataOffset Property
- MessageLength Property
- ReasonCode Property

The Message Descriptor Properties are:

- AccountingToken Property
- AccountingTokenHex Property
- ApplicationIdData Property
- ApplicationOriginData Property
- BackoutCount Property
- CharacterSet Property
- CorrelationId Property
- CorrelationIdHex Property
- Encoding Property
- Expiry Property
- Feedback Property
- Format Property
- MessageId Property
- MessageIdHex Property
- MessageType Property
- Persistence Property
- Priority Property
- PutApplicationName Property
- PutApplicationType Property
- PutDateTime Property
- ReplyToQueueManagerName Property
- ReplyToQueueName Property
- Report Property
- UserId Property

Methods:

- ClearErrorCodes Method
- ClearMessage Method
- ReadUnsignedByte Method
- ReadShort Method
- ReadLong Method
- ReadString Method
- ResizeBuffer Method
- WriteUnsignedByte Method
- WriteShort Method
- WriteLong Method
- WriteString Method

LotusScript Events:

- Mqerror
- Mqwarning

Creation:

New creates a new MQMessage object. Its Message Descriptor properties are initially set to default values, and its Message Data buffer is empty

Syntax:

Dim *msg* **As New MQMessage** or

Set *msg* = **New MQMessage**

Property Access:

All properties can be read at any time.

The control properties are read-only, except for DataOffset which is read-write. The Message Descriptor properties are all read-write, except BackoutCount which is read-only.

Note however that some of the MQMD properties may be modified by the Queue Manager when the message is put onto an MQSeries queue. See the *MQSeries Application Programming Reference* manual for details.

You can pass binary data to an MQSeries message by setting the CharacterSet property to the Coded Character Set Identifier of the queue manager (MQCCSI_Q_MGR), and passing it a string. You can use the chr$ function to set non-character data into the string.

Data Conversion:

The ReadLong, ReadShort,WriteLong, and WriteShort methods perform data conversion. They convert between the LotusScript internal formats, and the MQSeries message formats as defined by the Encoding and CharacterSet properties from the message descriptor. When writing a message you should, if possible, set values into Encoding and CharacterSet that match the characteristics of the recipient of the message before issuing a WriteLong or WriteShort method. When reading a message this is not normally required as these values will have been set from those in the incoming MQMD.

CompletionCode Property

Read-only. Returns the MQSeries completion code set by the most recent method or property access issued against this object.

Defined in:
MQMessage Class

Data Type:
Long

Legal Values:
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
To get: *completioncode& = MQMessage*.**CompletionCode**

DataLength Property

Read-only. This property returns the value

Message.MessageLength - Message.DataOffset

It can be used before a Read method, to check that the expected number of characters are actually present in the buffer.

The initial value is zero.

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *bytesleft& = MQMessage*.**DataLength**

DataOffset Property

Read-write. The current position within the Message Data portion of the message object.

The value is expressed as a character offset from the start of the message data buffer; the first character in the buffer corresponds to a DataOffset value of zero.

A read or write method commences its operation at the character referenced by DataOffset. These methods process data in the buffer sequentially from this position, and update DataOffset to point to the character (if any) immediately following the last character processed.

DataOffset may only take values in the range 0..MessageLength inclusive. When DataOffset = MessageLength it is pointing to the end (first invalid character) of the buffer. Write methods are permitted in this situation - they extend the data in the buffer and increase MessageLength by the number of characters added. Reading beyond the end of the buffer is not supported.

The initial value is zero.

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *currpos&* = *MQMessage*.**DataOffset**

To set: *MQMessage*.**DataOffset** = *currpos&*

MessageLength Property

Read-only. Returns the total length of the Message Data portion of the message object in characters, irrespective of the value of DataOffset.

The initial value is zero. It is set to the incoming Message Length after a Get method invocation that referenced this message object. It is incremented if the application uses a Write method to add data to the object. It is unaffected by Read methods.

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *msglength& = MQMessage*.**MessageLength**

ReasonCode Property

Read-only. Returns the reason code set by the most recent method or property access issued against this object.

Defined in:
MQMessage Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual.

Syntax:
To get: *reasoncode& = MQMessage*.**ReasonCode**

AccountingToken Property

Read-write. The MQMD AccountingToken - part of the message Identity Context.

Note When setting this property you must specify all 32 characters.

Its initial value is all NULLS.

Defined in:
MQMessage Class

Data Type:
String of 32 characters

Syntax:
To get: *actoken$* = *MQMessage*.**AccountingToken**

To set: *MQMessage*.**AccountingToken** = *actoken$*

AccountingTokenHex Property

Read-write. The MQMD AccountingToken - part of the message Identity Context.

Every two characters represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "a", the pair of characters "6" and "2" represent the single character "b", and so on.

You must supply 64 valid hexadecimal characters.

Its initial value is "0..0"

Defined in:
MQMessage Class

Data Type:
String of 64 hexadecimal characters representing 32 ASCII characters

Syntax:
To get: *actokenh$* = *MQMessage*.**AccountingTokenHex**

To set: *MQMessage*.**AccountingTokenHex** = *actokenh$*

ApplicationIdData Property

Read-write. The MQMD ApplIdentityData - part of the message Identity Context.

Its initial value is all blanks.

Defined in:
MQMessage Class

Data Type:
String of 32 characters

Syntax:
To get: *applid$* = *MQMessage*.**ApplicationIdData**

To set: *MQMessage*.**ApplicationIdData** = *applid$*

ApplicationOriginData Property

Read-write. The MQMD ApplOriginData - part of the message origin context.

Its initial value is all blanks.

Defined in:
MQMessage Class

Data Type:
String of 4 characters

Syntax:
To get: *applor$* = *MQMessage*.**ApplicationOriginData**

To set: *MQMessage*.**ApplicationOriginData** = *applor$*

BackoutCount Property

Read-only. The MQMD BackoutCount.

Its initial value is 0

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *backoutct&* = *MQMessage*.**BackoutCount**

CharacterSet Property

Read-write. The MQMD CodedCharSetId. This specifies the character set used for the application data in the message.

Its initial value is the special value MQCCSI_Q_MGR.

If CharacterSet is set to MQCCSI_Q_MGR (the CharacterSet for the data is the same as that for the queue manager), the WriteString method will not perform codepage conversion.

For example:

```
msg.CharacterSet = MQCCSI_Q_MGR
```

```
msg.WriteString(chr$(n))
```

where 'n' is greater than or equal to zero and less than or equal to 255, results in a single byte of value of 'n' being written to the buffer.

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *ccid&* = *MQMessage*.**CharacterSet**

To set: *MQMessage*.**CharacterSe**t = *ccid&*

Example:
If you want the string written out in codepage 437, issue

```
Message.CharacterSet = 437
```

```
Message.WriteString ("string to be written")
```

Set the value you want in the CharacterSet before issuing any WriteString calls.

CorrelationId Property

Read-write. The CorrelId to be included in the MQMD of a message when put on a queue, also the Id to be matched against when getting a message from a queue.

Note When setting this property you must specify all 24 characters.

Its initial value is all Nulls.

Defined in:
MQMessage Class

Data Type:
String of 24 characters

Syntax:
To get: *correlid$* = *MQMessage*.**CorrelationId**

To set: *MQMessage*.**CorrelationId** = *correlid$*

CorrelationIdHex Property

Read-write. The CorrelId to be included in the MQMD of a message when put on a queue, also the CorrelId to be matched against when getting a message from a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "a", the pair of characters "6" and "2" represent the single character "b", and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is "0..0".

Defined in:
MQMessage Class

Data Type:
String of 48 hexadecimal characters representing 24 ASCII characters

Syntax:
To get: *correlidh$* = *MQMessage*.**CorrelationIdHex**

To set: *MQMessage*.**CorrelationIdHex** = *correlidh$*

Encoding Property

Read-write. The MQMD field that identifies the representation used for numeric values in the application message data.

Its initial value is the special value MQENC_NATIVE, which varies by platform.

Note Setting this property to MQENC_INTEGER_UNDEFINED has the same results as setting it to the default, MQENC_NATIVE.

This property is used by the following methods:

- ReadLong
- WriteLong
- ReadShort
- WriteShort

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *encoding&* = *MQMessage*.**Encoding**

To set: *MQMessage*.**Encoding** = *encoding&*

If you are preparing to write data to the message buffer, you should set this field to match the characteristics of the receiving queue manager plaform if you know what it is.

Expiry Property

Read-write. The MQMD expiry time field, expected in tenths of a second.

Its initial value is the special value MQEI_UNLIMITED

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *expiry&* = *MQMessage*.**Expiry**

To set: *MQMessage*.**Expiry** = *expiry&*

Feedback Property

Read-write. The MQMD feedback field.

Its initial value is the special value MQFB_NONE.

Defined in:
MQMessage Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual

Syntax:
To get: *feedback&* = *MQMessage*.**Feedback**

To set: *MQMessage*.**Feedback** = *feedback&*

Format Property

Read-write. The MQMD format field. Gives the name of a built-in or user-defined format that describes the nature of the Message Data.

Its initial value is the special value MQFMT_NONE.

Defined in:
MQMessage Class

Data Type:
String of 8 characters

Syntax:
To get: *format$* = *MQMessage*.**Format**

To set: *MQMessage*.**Format** = *format$*

MessageId Property

Read-write. The MessageId to be included in the MQMD of a message when put on a queue, also the Id to be matched against when getting a message from a queue.

Note When setting this property you must specify all 24 characters.

Its initial value is all Nulls.

Defined in:
MQMessage Class

Data Type:
String of 24 characters

Syntax:
To get: *messageid$* = *MQMessage*.**MessageId**

To set: *MQMessage*.**MessageId** = *messageid$*

MessageIdHex Property

Read-write. The MessageId to be included in the MQMD of a message when put on a queue, also the MessageId to be matched against when getting a message from a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "a", the pair of characters "6" and "2" represent the single character "b", and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is "0..0".

Defined in:
MQMessage Class

Data Type:
String of 48 hexadecimal characters representing 24 ASCII characters

Syntax:
To get: *messageidh$* = *MQMessage*.**MessageIdHex**

To set: *MQMessage*.**MessageIdHex** = *messageidh$*

MessageType Property

Read-write. The MQMD MsgType field.

Its initial value is MQMT_DATAGRAM.

Defined in:
MQMessage Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual

Syntax:
To get: *msgtype&* = *MQMessage*.**MessageType**

To set: *MQMessage*.**MessageType** = *msgtype&*

Persistence Property

Read-write. The message's persistence setting.

Its initial value is MQPER_PERSISTENCE_AS_Q_DEF.

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *persist&* = *MQMessage*.**Persistence**

To set: *MQMessage*.**Persistence** = *persist&*

Priority Property

Read-write. The message's priority.

Its initial value is the special value MQPRI_PRIORITY_AS_Q_DEF

Defined in:
MQMessage Class

Data Type:
Long

Syntax:
To get: *priority& = MQMessage*.**Priority**

To set: *MQMessage*.**Priority** = *priority&*

PutApplicationName Property

Read-write. The MQMD PutApplName - part of the Message Origin context.

Its initial value is all blanks.

Defined in:
MQMessage Class

Data Type:
String of 28 characters

Syntax:
To get: *putapplnm$ = MQMessage*.**PutApplicationName**

To set: *MQMessage*.**PutApplicationName** = *putapplnm$*

PutApplicationType Property

Read-write. The MQMD PutApplType - part of the Message Origin context.

Its initial value is MQAT_NO_CONTEXT

Defined in:
MQMessage Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual

Syntax:
To get: *putappltp&* = *MQMessage*.**PutApplicationType**

To set: *MQMessage*.**PutApplicationType** = *putappltp&*

PutDateTime Property

Read-write. This property combines the MQMD PutDate and PutTime fields. These are part of the Message Origin context that indicate when the message was put.

The LotusScript Extension converts between LotusScript date/time format and the Date and Time formats used in an MQSeries MQMD. If a message is received which has an invalid PutDate or PutTime, then the PutDateTime property after the get method will be set to EMPTY.

Its initial value is EMPTY

Defined in:
MQMessage Class

Data Type:
Variant of type 7 (date/time) or EMPTY.

Syntax:
To get: *datetime* = *MQMessage*.**PutDateTime**

To set: *MQMessage*.**PutDateTime** = *datetime*

ReplyToQueueManagerName Property

Read-write. The MQMD ReplyToQMgr field.

Its initial value is all blanks

Defined in:
MQMessage Class

Data Type:
String of 48 characters

Syntax:
To get: *replytoqmgr$* = *MQMessage*.**ReplyToQueueManagerName**

To set: *MQMessage*.**ReplyToQueueManagerName** = *replytoqmgr$*

ReplyToQueueName Property

Read-write. The MQMD ReplyToQ field.

Its initial value is all blanks

Defined in:
MQMessage Class

Data Type:
String of 48 characters

Syntax:
To get: *replytoq$* = *MQMessage*.**ReplyToQueueName**

To set: *MQMessage*.**ReplyToQueueName** = *replytoq$*

Report Property

Read-write. The message's Report options.

Its initial value is MQRO_NONE.

Defined in:
MQMessage Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual.

Syntax:
To get: *report& = MQMessage*.**Report**

To set: *MQMessage*.**Report** = *report&*

UserId Property

Read-write. The MQMD UserIdentifier - part of the message Identity Context.

Its initial value is all blanks.

Defined in:
MQMessage Class

Data Type:
String of 12 characters

Syntax:
To get: *userid$ = MQMessage*.**UserId**

To set: *MQMessage*.**UserId** = *userid$*

ClearErrorCodes Method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQMessage Class and the MQSession Class.

Defined in:
MQMessage Class

Syntax:
Call *MQMessage*.**ClearErrorCodes**

ClearMessage Method

This method clears the data buffer portion of the MQMessage object. Any Message Data in the data buffer is lost, as MessageLength, DataLength, and DataOffset are all set to zero.

The Message Descriptor (MQMD) portion is unaffected; an application may need to modify some of the MQMD fields before reusing the MQMessage object. If you wish to set the MQMD fields back to initial values you should use New to replace the object with a new instance.

Defined in:
MQMessage Class

Syntax:
Message.**ClearMessage**

ReadLong Method

This method reads 4 characters from the Message Data buffer, starting with the character referred to by **DataOffset** and returns it as a Long (signed 4-byte) integer value.

The method fails if *Message*.**DataLength** is less than 4 when it is issued.

DataOffset is incremented by 4 and **DataLength** is decremented by 4 if the method succeeds.

The 4 characters of message data are assumed to be a binary integer whose encoding is specified by the *Message*.**Encoding** property. Conversion to LotusScript representation is performed for the application.

Defined in:
MQMessage Class

Syntax:
bigint& = *Message*.**ReadLong**

ReadShort Method

This method reads 2 characters from the Message Data buffer, starting with the character referred to by **DataOffset** and returns it as an Integer (signed 2-byte) value.

The method fails if *Message*.**DataLength** is less than 2 when it is issued.

DataOffset is incremented by 2 and **DataLength** is decremented by 2 if the method succeeds.

The 2 characters of message data are assumed to be a binary integer whose encoding is specified by the *Message*.**Encoding** property. Conversion to LotusScript representation is performed for the application.

Defined in:
MQMessage Class

Syntax:
count% = *Message*.**ReadShort**

ReadString Method

This method reads n characters from the Message Data buffer (where n can be any number in the range 1 to 32000 inclusive) starting with the character referred to by **DataOffset** and returns it as a LotusScript string.

The method fails if *Message*.**DataLength** is less than n when it is issued.

DataOffset is incremented by n and **DataLength** is decremented by n if the method succeeds.

The n characters of message data are assumed to be a string whose codepage is specified by the *Message*.**CharacterSet** property. Conversion to LotusScript representation is performed for the application.

Defined in:
MQMessage Class

Syntax:
firstname$ = *Message*.**ReadString**(*length&*)

Parameter:
Length&

Long. Length of string field in characters.

ReadUnsignedByte Method

This method reads a single character, referred to by the **DataOffset**, from the Message Data buffer and returns it as a single unsigned integer.

The method fails if the *Message*.**DataLength** is less than 1 when it is issued.

DataOffset is incremented by 1 and **DataLength** is decremented by 1 if the method succeeds.

The character of the message data is assumed to be binary. There is no data conversion.

Defined in:
MQMessage Class

Syntax:
Message.**ReadUnsignedByte**(*value%*)

Parameter:
value%

Integer. Value to be written.

ResizeBuffer Method

This method alters the amount of storage currently allocated internally to hold the Message Data Buffer. It gives the application some control over the automatic buffer management, in that if the application knows that it is going to deal with a large message, it can ensure that a sufficiently large buffer is allocated. The application does not need to use this call - if it does not, then the automatic buffer management code will grow the buffer size to fit.

Caution If you resize the buffer to be smaller that the current **MessageLength**, you risk losing data. If you do lose data , the method returns a **CompletionCode** of MQCC_WARNING and a ReasonCode of MQRC_DATA_TRUNCATED.

If you resize the buffer to be smaller than the value of the **DataOffset** property the:

- **DataOffset** property is changed to point to the end of the new buffer
- **DataLength** property is set to zero
- **MessageLength** property is changed to the new buffer size

The default buffer size if 2K bytes.

Defined in:
MQMessage Class

Syntax:
Message.**ResizeBuffer**(*Length&*)

Parameter:
Length&

Long. Size required in characters.

WriteLong Method

This method takes a signed 4-byte integer value and writes it into the Message Data buffer as a 4-character binary number starting at the character referred to by **DataOffset**. It replaces any data already at these positions in the buffer, and extends the length of the buffer (*Message*.**MessageLength**) if necessary.

DataOffset is incremented by 4 if the method succeeds.

The method converts to the binary representation specified by the *Message*.**Encoding** property.

Note A message with the Encoding property set to MQENC_INTEGER_UNDEFINED is processed in the same way as a message with the Encoding property set to MQENC_NATIVE.

Defined in:
MQMessage Class

Syntax:
Message. **WriteLong**(*value&*)

Parameter:
value&

Long. Value to be written.

WriteShort Method

This method takes a signed 2-byte integer value and writes it into the Message Data buffer as a 2-character binary number starting at the character referred to by **DataOffset** . It replaces any data already at these positions in the buffer, and will extend the length of the buffer (*Message*.**MessageLength**) if necessary.

DataOffset is incremented by 2 if the method succeeds.

The method converts to the binary representation specified by the *Message*.**Encoding** property.

Note A message with the Encoding property set to MQENC_INTEGER_UNDEFINED is processed in the same way as a message with the Encoding property set to MQENC_NATIVE.

Defined in:
MQMessage Class

Syntax:
Message.**WriteShort**(*value%*)

Parameter:
value%

Integer. Value to be written.

WriteString Method

This method takes a LotusScript string and writes it into the Message Data buffer starting at the character referred to by **DataOffset** . It replaces any data already at these positions in the buffer, and will extend the length of the buffer (*Message*.**MessageLength**) if necessary.

DataOffset is incremented by the length of the string in characters if the method succeeds.

The method converts characters into the codepage specified by the *Message*.**CharacterSet** property.

Defined in:
MQMessage Class

Syntax:
Message. **WriteString**(*value$*)

Parameter:
value$

String. Value to be written.

WriteUnsignedByte Method

This method writes an unsigned byte integer into the current position (**DataOffset**) of the Message Data buffer as a single character binary number.

The method fails if the value is less than zero or greater than 255.

It replaces any data already at this position in the buffer, and will extend the length of the buffer (*Message*.**MessageLength**) if necessary.

Defined in:
MQMessage Class

Syntax:
Message.**WriteUnsignedByte**(*value%*)

Parameter:
value%

Integer. Value to be written.

MQPutMessageOptions Class

This class encapsulates the various options that control the action of putting a message onto an MQSeries Queue.

Containment:
Contained by the MQSession Class .

Properties:
- CompletionCode Property
- Options Property
- ReasonCode Property
- ResolvedQueueManagerName Property
- ResolvedQueueName Property

Methods:
- ClearErrorCodes Method

LotusScript Events:
- Mqerror
- Mqwarning

Creation:
New creates a new MQPutMessageOptions object and sets all its properties to initial values.

Syntax:
Dim *pmo* **As New MQPutMessageOption**s or

Set *pmo* = **New MQPutMessageOptions**

CompletionCode Property

Read-only. Returns the completion code set by the last method or property access issued against the object.

Defined in:
MQPutMessageOptions Class

Data Type:
Long

Legal Values:
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
To get: *completioncode& = PutOpts*.**CompletionCode**

Options Property

Read-write. The MQPMO Options field that controls the putting a message on a queue.

The options are listed in the *MQSeries Application Programming Reference* manual, for example:

MQPMO_SYNCPOINT - the message is not visible until the unit of work is completed

MQPMO_NO_SYNCPOINT - the message is available immediately and cannot be deleted by backing out a unit of work

MQPMO_DEFAULT_CONTEXT - the queue manager sets the context fields in the message.

The initial value is MQPMO_NONE.

Note The options not supported are:

MQPMO_PASS_IDENTITY_CONTEXT

MQPMO_PASS_ALL_CONTEXT

Defined in:
MQPutMessageOptions Class

Data Type:
Long

Syntax:
To get: *options&* = *PutOpts*.**Options**

To set: *PutOpts*.**Options** = *options&*

ReasonCode Property

Read-only. Returns the reason code set by the last method or property access issued against the object.

Defined in:
MQPutMessageOptions Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual

Syntax:
To get: *reasoncode& = PutOpts*.**ReasonCode**

ResolvedQueueManagerName Property

Read-only. The MQPMO ResolvedQMgrName field. See the *MQSeries Application Programming Reference* manual for details. The initial value is all blanks.

Defined in:
MQPutMessageOptions Class

Data Type:
String of 48 characters

Syntax:
To get: *qmgr$ = PutOpts*.**ResolvedQueueManagerName**

ResolvedQueueName Property

Read-only. The MQPMO ResolvedQName field. See the *MQSeries Application Programming Reference* manual for details. The initial value is all blanks.

Defined in:
MQPutMessageOptions Class

Data Type:
String of 48 characters

Syntax:
To get: *qname$ = PutOpts*.**ResolvedQueueName**

ClearErrorCodes Method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQPutMessageOptions Class and the MQSession Class.

Defined in:
MQPutMessageOptions Class

Syntax:
Call *PutOpts*.**ClearErrorCodes**

MQGetMessageOptions Class

This class encapsulates the various options that control the action of getting a message from an MQSeries Queue.

Containment:
Contained by the MQSession Class .

Properties:
- CompletionCode Property
- Options Property
- ReasonCode Property
- ResolvedQueueName Property
- WaitInterval Property

Methods:
ClearErrorCodes Method

LotusScript Events:
- Mqerror
- Mqwarning

Creation:
New creates a new MQGetMessageOptions object and sets all its properties to initial values.

Syntax:
Dim *gmo* **As New MQGetMessageOptions** or

Set *gmo* = **New MQGetMessageOptions**

CompletionCode Property

Read-only. Returns the completion code set by the last method or property access issued against the object.

Defined in:
MQGetMessageOptions Class

Data Type:
Long

Legal Values:
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
To get: *completioncode&* = *GetOpts*.**CompletionCode**

Options Property

Read-write. The MQGMO Options field that controls the action of getting a message from a queue.

The options are listed in the *MQSeries Application Programming Reference* manual, for example:

MQGMO_SYNCPOINT - the message is marked as being unavailable to other applications, but only deleted from the queue when the unit of work is committed. The message is made available again if the unit of work is backed out.

MQGMO_NO_SYNCPOINT - the message is retrieved and deleted from the queue immediately.

MQGMO_ACCEPT_TRUNCATED - allows you to retrieve a message when the buffer is too small, although the complete message cannot be put in the buffer. Unless you have resized the buffer, you will get 2K bytes of the message.

MQGMO_CONVERT - requests that the application data in the message is converted to conform to the values of the CharacterSet and Encoding properties.

The initial value is MQGMO_NO_WAIT.

Defined in:
MQGetMessageOptions Class

Data Type:
Long

Syntax:
To get: *options&* = *GetOpts*.**Options**

To set: *GetOpts*.**Options** = *options&*

ReasonCode Property

Read-only. Returns the reason code set by the last method or property access issued against the object.

Defined in:
MQGetMessageOptions Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual

Syntax:
To get: *reasoncode& = GetOpts*.**ReasonCode**

ResolvedQueueName Property

Read-only. The MQGMO ResolvedQName field. See the *MQSeries Application Programming Reference* manual for details.

The initial value is all blanks.

Defined in:
MQGetMessageOptions Class

Data Type:
String of 48 characters

Syntax:
To get: *qname$ = GetOpts*.**ResolvedQueueName**

WaitInterval Property

Read-write. The MQGMO WaitInterval field. The maximum time, in milliseconds, that the Get will wait for a suitable message to arrive - if wait action has been requested by the Options property. See the *MQSeries Application Programming Reference* manual for details. Initial value is 0.

Defined in:
MQGetMessageOptions Class

Data Type:
Long

Syntax:
To get: *wait&* = *GetOpts*.**WaitInterval**

To set: *GetOpts*.**WaitInterval** = wait&

ClearErrorCodes Method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQGetMessageOptions Class and the MQSession Class.

Defined in:
MQGetMessageOptions Class

Syntax:
Call *GetOpts*.**ClearErrorCodes**

MQProcess Class

This represents an MQSeries process definition object (used with triggering). Using the MQLSX, you can interrogate the properties of the process definition object within your script.

For more information about triggering, see the *MQSeries Application Programming Guide*.

Containment:
Contained by the MQQueueManager Class.

Properties:
- AlternateUserId Property
- ApplicationId Property
- ApplicationType Property
- CompletionCode Property
- Description Property
- EnvironmentData Property
- Name Property
- OpenStatus Property
- ReasonCode Property
- UserData Property

Methods:
ClearErrorCodes Method

LotusScript Events:
- Mqerror
- Mqwarning

Property Access:
All properties are read-only

AlternateUserId Property

Read-only. The alternate userid used to validate access to the Process definition when it was opened.

Defined in:
MQProcess Class

Data Type:
String of 12 characters

Syntax:
To get: *altuser$* = *Process*.**AlternateUserId**

ApplicationId Property

Read-only. The MQI ApplId attribute.

Defined in:
MQProcess Class

Data Type:
String of 256 characters

Syntax:
To get: *identifier$* = *Process*.**ApplicationId**

ApplicationType Property

Read-only. The MQI ApplType attribute.

Defined in:
MQProcess Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual.

Syntax:
To get: *appltype&* = *Process*.**ApplicationType**

CompletionCode Property

Read-only. Returns the completion code set by the last method or property access issued against the object.

Defined in:
MQProcess Class

Data Type:
Long

Legal Values:
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
To get: *completioncode&* = *Process*.**CompletionCode**

Description Property

Read-only. The MQI ProcessDesc attribute.

Defined in:
MQProcess Class

Data Type:
String of 64 characters

Syntax:
To get: *description$* = *Process*.**Description**

EnvironmentData Property

Read-only. The MQI EnvData attribute.

Defined in:
MQProcess Class

Data Type:
String of 128 characters

Syntax:
To get: *env$* = *Process*.**EnvironmentData**

Name Property

Read-only. The MQI ProcessName attribute. This is the name of a process definition object, defined on the local queue manager.

Defined in:
MQProcess Class

Data Type:
String of 48 characters

Syntax:
To get: *name$ = process*.**Name**

OpenStatus Property

Read-only. Indicates if the process is Open or not.

Defined in:
MQProcess Class

Data Type:
Long

Legal Values:
- TRUE (-1)
- FALSE

Syntax:
To get: *status& = Process*.OpenStatus

ReasonCode Property

Read-only. Returns the reason code set by the last method or property access issued against the object.

Defined in:
MQProcess Class

Data Type:
Long

Legal Values:
See the *MQSeries Application Programming Reference* manual

Syntax:
To get: *reasoncode& = Process*.**ReasonCode**

UserData Property

Read-only. The MQI UserData attribute.

Defined in:
MQProcess Class

Data Type:
String of 128 characters

Syntax:
To get: *user$* = *Process*.**UserData**

ClearErrorCodes Method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQProcess Class and the MQSession Class.

Defined in:
MQProcess Class

Syntax:
Call *Process*.**ClearErrorCodes**

Appendix A MQLSX Link sample application

This appendix describes the MQLSX Link sample application. This application is designed to show how, from a Notes environment, you can run a program in a non-Notes environment and get data returned to you. It demonstrates how you can use the MQLSX with a Notes agent and the link database. The design of the link database is the same as the link database in an MQSeries product.

In summary what happens is:

- A Notes program takes the text you input, using the Notes form, and creates a Notes document in the Agent database
- A Notes agent creates an MQSeries message from the document and passes it to the MQSeries environment
- An MQSeries program changes the text in the message and creates an MQSeries reply message
- The Notes agent picks up the MQSeries reply message and updates the original document with the data

This sample is a way of implementing the MQSeries link (mqlink) using LotusScript and the MQLSX. The same link database is used in both cases.

This appendix describes:

- The components of the application, what they contain and the role of each individual part
- What you must do before you run the application
- How to run the application
- What happens when you run the application showing the movement of documents and messages
- Customization to change the data being passed

Look at the section on What happens when you run the MQLSX Link sample, for the diagram that shows how the components fit together and their relationship with Notes and MQSeries.

MQSeries LotusScript Extension Link sample application

The MQSeries LotusScript Extension Link sample application (MQLSX Link sample) allows you to develop a Notes application that can access an MQSeries existing application **without** the need to make any changes to the existing application.

It enables you to send data from a Notes application and get a reply, containing data, back from an MQSeries application. It demonstrates how you can use the MQLSX in conjunction with a Notes agent and the mqlink.nsf database, to interact with applications outside of Notes, using MQSeries.

Within your Notes application you write a structured message document that the link sample reads and processes in accordance with the requested entry in the link database.

It is designed to provide a base that you can change and extend to create your own applications.

Design of the MQLSX Link sample

The components are:

- Notes LotusScript application client database (gmqlclnt.nsf).
- Agent database (gmqlagnt.nsf)
- Link database (mqlink.nsf)
- MQSeries application (amqslnk0 sample program)
- MQSeries queue to receive messages (system.sample.notes.inqueue)
- MQSeries queue to return reply messages (system.sample.notes.outqueue)

Limitations

The limitations of this sample are:

- Both the message you must send and the reply message must be a structured message, of a predetermined length
- You can send the message to only one destination
- The reply message only updates the original Notes document

Limitations have been put on this sample to maintain a simple model that shows you how Notes and MQSeries can work together, and how the MQLSX can be used. The limitations are not related not the MQLSX.

Notes LotusScript application client database (gmqlclnt.nsf)

The Notes LotusScript application client database is made up of:

- A form named MQLSX Client Request
  This is the form you use to enter the data you want to send to your MQSeries application. The agent database and link database document names are automatically displayed for you, but you can override these if you wish. You should enter the name of your agent server database. Enter data in the 'Data to be sent' field and select OK.

  Note The type of data you enter must match the entry in the link database document you select.

  You also use this form to ask for a reply to a document you sent earlier. Display the document you sent earlier and select OK.

- A form named MQREQUEST.
  This is the form that the client application uses to create the document that it sends to the agent. The document it creates is held in the agent database and contains the data that is put into the MQSeries message. This document is also updated with the reply data (sent in a MQSeries message that is retrieved from the specified reply queue). The sample detects when a reply is available by looking at the status in the MQREQUEST document on the agent database. The values the status field can take are:

Value	Status of the request
1	Waiting
2	Available - reply message available
3	None - no reply needed
4	Error (e.g link entry error or send failure)

This form is displayed in the list when you run the sample, however it is for the sole use of the Notes sample program and it is **not** used for data entry.

- A form named MQCOMPLETE.
  This form is used to create a document when the reply data has been read from the MQREQUEST document on the agent database. The MQCOMPLETE document is sent to the agent database to indicate that the MQREQUEST document can be deleted.

  This form is displayed in the list when you run the sample, however it is for the sole use of the Notes sample program and it is **not** used for data entry.

- A form named MQSTARTAGENT.
  This form is used to create a document when you select OK on a MQREQUEST document, held on your own database, that you completed previously. The document is sent to the agent database, causing the agent to run and check to see if a reply has been received for this, or any other, MQREQUEST document.

  This form is displayed in the list when you run the sample, however it is for the sole use of the Notes sample program and it is **not** used for data entry.
- Any documents you create and save whilst running this application

MQSeries sample program (amqslnk0)

This program is also provided by the majority of the MQSeries family of products. It reads a message from the SYSTEM.SAMPLE.NOTES.INQUEUE queue. If a reply is required (signified by the presence of a reply queue name in the document on the mqlink database), amqslnk0 changes the order of the input data and creates a reply message that it puts on the specified reply queue, for example, SYSTEM.SAMPLE.NOTES.OUTQUEUE.

MQLSX Agent database (gmqlagnt.nsf)

The MQLSX Agent database is made up of:

- The sample Agent program.
  The agent runs "If Documents Have Been Created or Modified".
  The agent accesses the specified 'link database document' to obtain the:
  - Format of the message expect by the MQSeries application.
  - Names of the queues to be used.
  - Details of the fields in the MQREQUEST document and the MQSeries reply message.
- An agent parameters form.
  This form is used to create an agent parameters document. Should you need to recreate it, it must contain the following information (needed by the agent):
  - Name of Server that the link database is on (or blank for local)
  - Name of the Queue Manager
  - Name of the link database
  - Whether or not character fields should be padded with spaces

A sample agent parameters document is provided. You should modify it by changing the name of the server and queue manager to names that are correct for your system. This document must be available to the agent, otherwise an error is returned.

- A form named MQREPLYCONTEXT.
  This form is used to create a document that holds the control information that the agent needs to process a reply message for a specific request. An MQREPLYCONTEXT document is created for every message sent that requires a reply. The information held in this document is:
  - Request Document NoteID
  - Link Entry Document NoteID
  - MQSeries MsgId

This enables an MQSeries reply message to be matched up with the correct MQREQUEST document.
This form is for the sole use of the Notes sample program and it is **not** used for data entry.

- An MQREQUEST document.
  This document is sent by the client application and is a copy of the one held on the application database. It holds the:
  - Message data
  - Name of the link database document

The agent adds the:

  - Reply data
  - Time the message was processed by the MQSeries application
  - Error messages if any errors were encountered
- The agent also maintains the status of the request document:

This form is for the sole use of the Notes sample program and it is **not** used for data entry.

- An MQSTARTAGENT document.
  This document is sent by the client application when a reply to a previously sent message is requested. It activates the agent.
  This form is displayed in the list when you run the MQLSX Link sample, however it is for the sole use of the Notes program and it is **not** used for data entry.

- An MQCOMPLETE document.
  This document is sent by the client application when the reply to an MQREQUEST has been received. It indicates to the agent that the copy of the MQREQUEST document it identifies can now be deleted.
  This form is displayed in the list when you run the sample, however it is for the sole use of the Notes sample program and it is **not** used for data entry.

The link database (mqlink.nsf)

The link database holds details of each 'request type' you use when you run the sample application. The 'request type' is the 'link database document name' you specify when you run the client application. The information in each link database document defines the mapping between the fields of the input form and the MQSeries message offsets. Each link database document also holds field type and offset information that is used by the agent to create an MQSeries message, and to process the reply message, if one has been requested.

The link database is the same as the one supplied with the MQSeries products. There is no need to start with an empty link database if you already have one and can make use of it.

A sample link database entry:

Each link database document you create must follow this structure. It is important to get the start and end positions of the fields within the MQSeries request message and reply message correct. Remember, that the first character is in position zero.

- Entry
  The link database document name, the entry identifier. The MQENTRY name is used by the sample application.

- Database Information
  Information required by the existing MQSeries Lotus Notes link application. This is not used by the MQLSX Link sample.

- Request Offsets
  This describes the layout of the message you send to the MQSeries application. The name you give each field is of no significance to the agent. However, the order you list the fields must match the order in which they appear in the request document, and the format must follow that shown.
  In this example:
  function 0 2 CHAR (the sample Notes program puts the characters Msg in this field)
  Msg 3 104 CHAR (this is where the data you entered is held)

  Note You can have as many fields as you like to describe your message, however you can also have as little as one field. If each field is of the same type, there is more flexibility by having a single entry - more messages are likely to be able to use it.

- Reply Offsets
  This describes the layout of the data in the reply message. In each case you must describe how the information is held, whether it is character or numeric. If it is numeric, you specify whether it is big-endian (S390-Binary), or little-endian (Intel-Binary).
  For example:
  RData 0 101 CHAR (this is where the "Data received" information is in the reply message)
  Time 102 127 CHAR (this is where the "Time" information is in the reply message)

- Message Queuing Parameters
  This is where you specify the MQSeries queues you want to use and the format of data that the MQSeries message will contain.
  Queues
  If you do **not** want a reply from your MQSeries application, leave the reply queue name blank. If you do want a reply, the queue you specify must be local to your MQSeries application.
  Message format
  The format field is used when data conversion is necessary. The options are:
  Blank - Use this when the message contains non-character data.
  MQSTR - Use this when the fields in the message are all of type CHAR.
  User-defined format - Use this option when:
  - the message contains a mixture of character and non-character data
  - a user exit program is available at the server to enable MQSeries to provide the required conversion

- Error handling
  Here you enter the validation you want to take place before your Notes document is updated, and specify what information you want reported with the error . When the error conditions are met, an error message is displayed in the originating (MQLSX Client Request) document.
  For example:
  Error condition: client 0 5 CHAR
  Syntax for additional information: RData 0 101 CHAR
  In this case, a Notes mail memo, containing the reply message data (but not the time), is created if the reply message has the word client as the first six characters of the reply message.
  If you want to check for a 'not equal' value, prefix the character string with an exclamation mark (!).

Note The comparison of characters **is** case sensitive.

Before you run the MQLSX Link sample

This sample will not run successfully until you have completed the following:

- Install the MQLSX Agent database (gmqlagnt.nsf) and the link database (mqlink.nsf) on your Notes server. Select File - Database - New Copy.
- Install the MQLSX Client database (gmqlclnt.nsf) on your Notes Client.
- Ensure the queue manager you want to communicate with is running.
- Update the MQSeries Agent Parameters document, in the MQLSX Agent database, to reflect the name of the link database, the Notes server (initially set to Your Server Name), and the MQSeries Queue Manager you are using. The link database must be local and the name blank if your agent is running on a server.
  Check the box if you want the message data you send each time to be padded out with blanks.
- Change the name of the Notes server (that the MQLSX agent will run on) in the MQLSX Agent database. Initially the name of the server is given as Your Server Name. To change this:
  - Open the MQSLX Agent database (gmqlagnt.nsf)
  - Select Agents - MQLSX Agent - Schedule
  - Change the box 'Run only on .....'
  - Save
- Create the queues to hold the MQSeries request and the reply messages. The sample uses SYSTEM.SAMPLE.NOTES.INQUEUE and SYSTEM.SAMPLE.NOTES.OUTQUEUE. To create these queues, process amqslnk0.tst using the MQSeries runmqsc commands utility (runmqsc *QueueManagerName* < amqslnk0.tst).
  If you want to use different queues, you must change the Message Queuing Parameters in the MQENTRY document or use a different link database document.
- There is only one sample document in the link database (mqlink.nsf) called MQENTRY. If you want to add your own document to the link database:
  Open the database
  Select 'Create'
  Select 'External-Call Parameters'
  Complete the form (there is no validation performed against the data you enter)
  Select File - Save

- If you want to define your own MQSeries message formats, you must write the corresponding exit routine to perform data conversion for the destination (server) queue manager if you require it.

Running the MQLSX Link sample application

1. Check that all the appropriate set-up work has been completed. See "Before you run the MQLSX Link sample" for details.
2. Start MQSeries program, amqslnk0, running in your MQSeries environment. From a command line, enter

```
amqslnk0 [-q InputQName] [QMgrName]
```

where the default InputQName is SYSTEM.SAMPLE.NOTES.INQUEUE on the default queue manager

3. Single click on the MQLSX Client icon, this selects the database (or open the database by double clicking).
4. Select Create from the tool bar.
5. Select MQLSX Client Request to run the client application.
6. Use the tab function to move from one field to another. Change the Agent server and database names if you need to use different ones from those displayed.

   To change the default names displayed by the client application:
   - Select Design - Forms
   - Double click on the MQLSX Client Request form name
   - Select name of the field you want to change against Define
   - Select 'Default Value' against Event
   - Enter new default value
7. Change the link database document name if you want to use a different one from the default.
8. Enter the data you want to send to the MQSeries application.
9. Click OK.

   If the link database document you have chosen **does not** include the name of a reply queue, 'Reply Message is not expected ' is displayed in a message box.
   If the link database document does include the name of a reply queue, the sample will display the reply data and the time. It waits approximately 10 seconds for the MQSeries reply message to appear on the queue. When this time is exceeded, you are given the option of retrying (select OK) or cancelling.
   If you retry, it has the same 10 second threshold, after which the process is repeated.
   If you select cancel, you must save the document before exiting the client application if you want to get the reply at a later time.

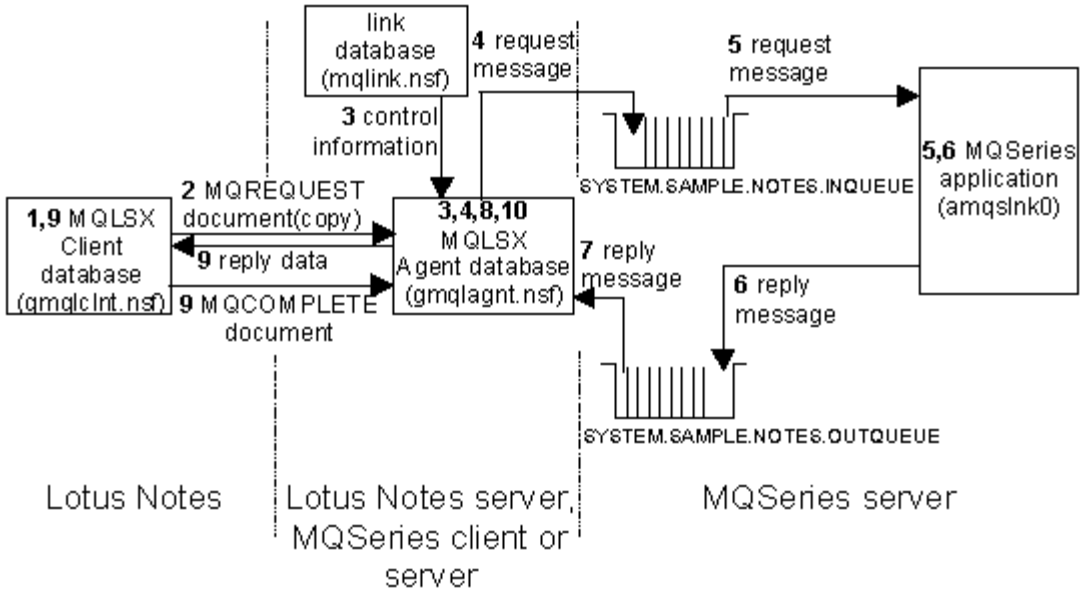
To request the reply to a request document you sent earlier

1. Double click on the MQLSX Client icon to open the database.
2. A list of documents you have saved are displayed in the sample view. Select the one you are interested in (double click).
3. Click OK.

What happens when you run the MQLSX Link sample

When you run this application, using the unmodified MQLSX Agent database and link database document, it composes a document that includes a request for reply information from the MQSeries program (amqslnk0 or an application of your own), and concludes by updating the original Notes document with the requested information.

Components of the MQLSX link sample application (request for reply data)



Using the MQLSX Client request form in the Client database, you enter the data you want to send and Click OK.

1. Using the sample input form on the Notes client, you enter the data you want to send. The input form also allows you to specify the link database document name, and the names of the agent server and database.
2. Click OK. A LotusScript program associated with the OK button composes a document using the MQREQUEST form. This MQREQUEST document is created on the client database (gmqlclnt.nsf) and copied to the Agent database (gmqlagnt.nsf).

3. The agent, which is 'change activated', runs when it detects the new document and reads it. Using the information in the Agent Parameters document, the agent:
   - Extracts the name of the link database document
   - Reads the link database document
   - Connects to the queue manager
   - Opens the input queue e.g. SYSTEM.SAMPLE.NOTES.INQUEUE
   - Constructs an MQSeries message, using the information in the link database document as well as the MQREQUEST document
   - Puts the MQSeries message on the queue
   - Creates an MQREPLYCONTEXT document if a reply message is expected
   - Looks for replies for any MQREPLYCONTEXT documents
4. The MQSeries sample server application, amqslnk0, gets the message from the queue (it uses the MQGMO_WAIT option) and processes it. If the data in the message is character, the words in the message are reversed, for example
   "Hello World!" becomes "World! Hello".

   If a reply has been requested:
5. the program amqslnk0 creates the MQSeries reply message, to include the time as well as the data, placing the message on the reply queue
6. The Agent gets the MQSeries message from the reply queue specified in the MQREPLYCONTEXT document, e.g. SYSTEM.SAMPLE.NOTES.OUTQUEUE
7. The Agent:
   - Uses the MQREPLYCONTEXT document to match the reply up with the MQREQUEST document it has on its database
   - Updates the status of the MQREQUEST document to '2' to indicate that the message is now available
   - Deletes the MQREPLYCONTEXT document

8. The LotusScript client application associated with the input form:
   - Polls the agent database to check for a change in status of the MQREQUEST document to '2'. It does this by sending null StartAgent documents to the Agent database, each time OK is clicked, to regularly activate the Agent to perform reply message processing
   - If the reply is not available and 10 seconds have passed, the sample displays a message box giving you the option to retry, or cancel the request
   - If you chose to cancel the request, save your document if you want to collect the reply at a later time
   - If you retry, the sample continues to poll for a change in status of the MQREQUEST document on the agent database for a further 10 second period
   - When the message is available, the sample updates the MQREQUEST document on the MQLSX Client database (gmqlclnt.nsf) and displays the reply data and time
   - Sends an MQCOMPLETE document to the agent
9. The agent:
   - Runs when the MQCOMPLETE document arrives
   - Deletes the MQREQUEST document that the MQCOMPLETE document refers to

When you request a reply to an MQREQUEST document at a later point in time:

- The sample sends a MQSTARTAGENT document to the agent
- The agent runs:
  - Checks to see if there are any MQREPLYCONTEXT documents
  - Checks to see if there are any messages on the reply queues identified by the MQREPLYCONTEXT documents
  - Processes the messages as normal, using the MQREPLYCONTEXT and MQREQUEST documents it has in its database

Error handling in the MQLSX Link sample application

Errors checking takes place throughout the MQLSX sample application components. In addition to Notes and LotusScript errors that can occur, there are also MQLSX error situations that can cause error messages specific to the MQLSX to be displayed.

MQLSX Client Error messages

Errors detected by the MQLSX Client are displayed using a message box.

The following error, warning, and informational messages may be displayed by the MQLSX client:

`ERROR: Error saving document.`

An attempt to save a document in the current database was unsuccessful.

`ERROR: Request failed.`

A detailed error message is displayed in the current document (error_field_msg field).

`ERROR: The agent document required by this request is not available and may have been deleted.`

The MQREQUEST document in the agent database could not be copied to the client and may have been deleted.

`ERROR: Open Agent database failed.`

The agent database on the Notes server could not be opened.

`WARNING: Request is already complete. Resend?`

`INFO: Reply message is not expected.`

`INFO: Reply message not available. Retry?`

MQLSX Agent error messages

Errors detected by the MQLSX Agent are printed to the Notes server console. An error can be detected by:

- Lotus Notes
- MQLSX
- The Link sample

Where an error is detected by Notes or the MQLSX, the Link sample adds further information to the error message before printing it to the console.

The format of a message therefore is different to what you would traditionally expect:

Errors detected by Lotus Notes
The format of these messages output by the Link sample is:

MQLSX Link Agent Notes Error: Agent_Insert_String
Notes_Error_Message (Notes error = Notes_Error_Number Line nnnn)

where:

Agent_Insert_String
is any additional useful information about the error that the agent can provide and may be omitted if blank.

Notes_Error_Message
is the LotusScript error message

nnnn
is the line number in the LotusScript source code where the error occurred

Errors detected by the MQLSX
The format of these messages output by the Link sample is:

MQLSX Link Agent Error: Agent_Error_Number Agent_Error_Message
Agent_Insert_String MQLSX_Error_Message (Notes error =
Notes_Error_Number Line nnnn)

where:

Agent_Error_Number
is an agent error number that may be omitted for MQLSX detected errors if zero

Agent_Error_Message
is an agent error message

Agent_Insert_String
is any additional useful information about the error that the agent can provide and may be omitted if blank.

MQLSX_Error_Message
is the LotusScript MQLSX error message and will typically provide the MQSeries or MQLSX reason code

nnnn
is the line number in the LotusScript source code where the error occurred

Errors detected by the Link sample Agent
The format of these messages output by the Link sample is:

MQLSX Link Agent Error: Agent_Error_Number Agent_Error_Message Agent_Insert_String

where:

Agent_Error_Number
is an agent error number that may be omitted for MQLSX detected errors if zero

Agent_Error_Message
is an agent error message

Agent_Insert_String
is any additional useful information about the error that the agent can provide and may be omitted if blank.

The error messages you can encounter that are detected by the Agent component of the Link sample are:

```
67001 View not found.
```

The 'Agent Parameters' View was not found in the agent database.

```
67002 Parameter document not found.
```

The agent parameters document was not found in the agent database.

```
67003 Link Entry Error: start or end position value not
valid.
```

An error was detected in a Link database entry as indicated.

```
67004 Link Entry Error: one or more field items are
missing.
```

An error was detected in a Link database entry as indicated.

```
67005 Link Entry Error: data type not valid.
```

An error was detected in a Link database entry as indicated.

```
67006 Link Entry Error: data length not valid for NUM.
```

An error was detected in a Link database entry as indicated.

The data length specified for fields of type NUM, INTEL-BINARY or S390-BINARY must be 2 or 4.

```
67007 Link Entry Error: field length value too big.
```

An error was detected in a Link database entry as indicated.

The field length for fields of type CHAR may not exceed 32000

67008 Link database could not be opened.

The specified Link database could not be opened

67009 Reply processing error.

A reply processing error occurred. More information, that may include a reason code, is provided in the accompanying insert string

67010 Backout threshold of failing message exceeded; message will be discarded.

A failing message that was previously backed out was discarded. Information about the exact cause of the failure is provided in earlier error messages.

67011 Numeric value is not valid or causes overflow.

An non-numeric or out-of-range numeric value was entered by the user or returned in an MQSeries reply message.

Two-byte NUM, INTEL-BINARY, and S390-BINARY values numbers are unsigned numbers in the range 0 to 65535.

Four-byte NUM, INTEL-BINARY, and S390-BINARY values are signed numbers in the range -2147483648 to 2147483647.

The Agent_Insert_String may be blank or one of the following:

Link Entry document not found(used with Reply processing error)

Request document not found(used with Reply processing error)

Error accessing queue manager(used with MQLSX detected error)

MQLSX Client Request error messages

The following messages may be displayed in the MSLSX Client Request document (in the error_field_message field):

Reply message failed Link entry error condition checks.

The reply message failed the specified Link entry error condition checks.

Link Entry Document Error.

The specified Link entry document cannot be opened.

Numeric value is not valid or causes overflow.

An non-numeric or out-of-range numeric value was entered by the user or returned in an MQSeries reply message.

Two-byte NUM, INTEL-BINARY, and S390-BINARY values numbers are unsigned numbers in the range 0 to 65535.

Four-byte NUM, INTEL-BINARY, and S390-BINARY values are signed numbers in the range -2147483648 to 2147483647.

```
Backout threshold of failing message exceeded; message
will be discarded.
```

A failing message that was previously backed out was discarded. Information about the exact cause of the failure is provided in earlier error messages.

```
An error was detected by the agent. Consult the agent
messages for more information.
```

This provides indication to the user that the agent detected an error details of which are provided in the agent messages.

Customizing the MQSeries Link sample application

There are several ways in which you can change this sample, yet still get the benefit of using it without having to do any further programming.

You change the MQLSX Client database (gmqlclnt.nsf):

- If you want to change the length of time that the program waits before checking to see if there is a reply message
- If you want to send or receive anything other than the fields defined in the sample

You change the MQLSX Agent database (gmqlagnt.nsf):

- If you want to send or receive anything other than the fields defined in the sample
- If you want to change the name of the Notes server the link database runs on

Note The name of the Notes server can only be different from that of the agent if the agent is run manually.

Changing the wait time

Unless changed, the sample waits for 10 seconds before it checks to see if a reply message is available. To change this:

- Open the MQLSX Client database (gmqlclnt.nsf)
- Select Design - Forms - MQLSX Client Request
- Click on the Define OK button
- Select the Event Declarations option
- Change the value WAIT_TIME from 10 to the new value (in seconds)
- Save the form

Changing the name of the Notes server and Queue Manager for mqlink.nsf

To change this:

- Open the MQLSX Agent database (gmqlagnt.nsf)
- Under Folders & Views in the navigator, Select Agent Parameters
- Double click on mqlink.nsf
- Select Action - Edit Document
- Change the necessary fields and save the document

Changing the send and reply fields

Unless changed, the sample sends two fields of data (known as Func and SData) to the MQSeries application, and receives two fields of data (known as RData and Time) back in the reply message.

The fields sent are:

1. A fixed text string of three characters 'Msg'. The field name on the form is Function, in the code this is known as Func.
2. A field that contains the data you enter when you run the sample. This may or may not be padded out with spaces, it depends on what you select in the Agent Parameters document. This field is known as SData.

Changes are necessary to both the MQLSX Client database, the MQLSX Agent database and the link database:

MQLSX Client database

In this database you need to make changes to the:

- MQLSX Client Request form
- MQREQUEST form (to be copied to the MQLSX Agent database)

Looking at the MQLSX Client Request, you will find the code for the classes under '(Declarations)' and the code that calls the classes under the Event 'Click'.

MQLSX Agent database

In this database you need to make changes to the:

- MQLSX Agent
- MQREQUEST form (may be copied from the MQLSX Client database)
- Agent Parameters form

Looking at the MQLSX Agent, you will find the code for the classes under '(Declarations)' and the code that calls the classes under the Event 'Initialize'.

Link database

In this database you need to make changes to the:

- External Call Parameters document (or create a new one)

Note If the agent is running on a server, the link database must reside on the same server as the agent. The link database can only be on a different server from the agent if the agent is run manually.

Steps to take

The following lists the areas that you need to modify if you want to send or receive a different number of fields of data to that supplied within the sample:

- **MQREQUEST form**:
  - Open the MQLSX Client database (gmqlclnt.nsf)
  - Select Design - Forms - MQREQUEST (double-click)
  - Modify the form to include the fields you want and save it
  - Copy the form to the MQLSX Agent database (or make sure the form on the MQLSX Agent database has the same changes made to it)

- **Class/method code of the MQLSX Client Request form**:
  - Open the MQLSX Client database (gmqlclnt.nsf)
  - Select Design - Forms - MQLSX Client Request (double-click)

If you want to change the form:

- Modify the form to include the field you want and save it

If you want to change information relating to the field on the form, such as the Agent server name, the default values of one or more fields, or Define OK button.

To change the Define OK button:

- Reveal the lower pane to display the definitions and script
- Select Define OK(button) and Event '(Declarations)'
- Add/modify the Property Set statements in the TempDocument class to cover each piece of data you want in the send and reply messages
- Change the constant declaration MQ_MSG_FUNCTION, to alter the field 'Function' (the sample puts a text string 'Msg' in this filed) sent, or delete it if it is not required by your MQSeries application
- Add a Property Set statement to the CurrentDocument class for each piece of data you want in the reply message and change the ReInitialize method to initialize these properties
- Change the CopyMqMsgItems method in the CurrentDocument class to deal with the properties you now have in the send and reply messages
- Modify the UpdateWithReply method in the RequestDocInfo class to process the properties you now have in your reply message

- **Class/method code of the MQLSX Agen**t:
  - Open the MQSLX Agent database (gmqlagnt.nsf)
  - Select Agents and double click on MQLSX Agent
  - Reveal the lower pane to display the 'What should this agent do?'
  - Select Event ('Declarations)'
  - Change the Get Data and Set ReceiveData properties in the RequestDocument class to include all the properties you now have in the send and receive messages
  - Change the New method in the MQReplyMessage class to process all the properties you now have in the reply message.
- **Link database**:
  - If you change an existing document, you modify the Request Offsets and Reply Offsets
  - Alternatively you can create a new document from the External Parameters Call form

Data Conversion

If your MQSeries application and your Notes Server are running on platforms that have different character sets or different integer encodings, the data from your Notes application has to be converted. This can be performed by the MQLSX, the application, or by MQSeries on behalf of the application.

These are the steps that take place:

1. The agent builds the MQSeries request message using the information in Request Offsets in the link database entry. The format options are:

   CHAR
   The field is copied across unchanged and so appears in the codepage used by the Notes Server

   NUM
   The field is encoded as a 2-byte or 4-byte, signed or unsigned, integer in intel or S390-format according to the platform on which the agent is running. Use this format if conversion is performed at the receiver by MQSeries or the application.

   INTEL-BINARY
   The field is encoded as a 2-byte or 4-byte Intel format signed or unsigned integer.

   S390-BINARY
   The field is encoded as a 2-byte or 4-byte S390-format signed or unsigned integer. For messages received by the agent, both INTEL_BINARY and S390_BINARY formats are treated the same as NUM, with data conversion provided by the MQLSX.

Two-byte NUM, INTEL_BINARY, and S390-BINARY value numbers are unsigned numbers in the range 0 to 65535

Four-byte NUM, INTEL_BINARY, and S390-BINARY value numbers are signed numbers in the range -2147483648 to 2147483647.

2. When the agent puts the message on the request queue, it sets the message descriptor format field to the value given in the message format field of the link database entry. The possible values of the message format are:

   Blank
   Use this option when your application performs any necessary data conversion. Your application must check against the encoding and CodedCharSet fields in the message descriptor.

   MQSTR
   Use this option when all the fields in the message are of type CHAR. This indicates to MQSeries that the application data is all characters and MQSeries does any necessary data conversion. This is not recommended if you are converting between single-byte and double-byte character sets, as individual fields may expand or contract.

   User-defined format
   Use this option when a user exit program is provided at the receiver to enable MQSeries to perform the necessary conversion.

For more information see Using Data Conversion.

Designing your own applications using mqlink.nsf

When designing your application to incorporate Notes, there are some configuration factors that you need to consider:

- The MQLSX Client database can be installed on a Notes client or Notes server.
- The MQLSX Agent database must be run on a Notes server (unless the agent is to be run manually), with either an MQSeries client or MQSeries server installed.
- The link database (mqlink.nsf) must be on the same Notes server as the agent (unless the agent is to be run manually)
- Your MQSeries application must run on an MQSeries platform that supports your Agent environment.
- Messages put by the Agent are persistent.
- The Agent is not designed to be triggered by an MQSeries message.

Appendix B MQLSX link extra agent sample application

This appendix describes the MQLSX link extra agent sample application. This application is designed to show how, from an enterprise application, you can run a program that results in an update to a Notes database.

The Notes agent in this sample can be run manually or as a scheduled agent, but it can also run using a trigger monitor. The MQSeries trigger monitor for Lotus Notes agents can be used with both MQLSX and MQEI applications.

For more information see the *MQSeries Trigger Monitor for Lotus Notes agents User Guide*. The MQSeries Trigger Monitor for Lotus Notes Agents is available as a SupportPac from the IBM MQSeries home page:

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

This appendix describes:

- The components of the application, what they contain and the role of each individual part
- What you must do before you run the application
- How to run the application (manually, as a scheduled agent, or using the trigger monitor)
- What happens when you run the application
- How you can change the application (customization)

Look at the section on 'What happens when you run the MQLSX link extra agent sample', for the diagram that shows how the components fit together and their relationship with Notes and MQSeries.

Introduction

The MQLSX link extra agent sample provides enterprise applications with a means of sending data to a Lotus Notes environment via MQSeries. This data can be used to update or add one or more documents to a Notes database. Such updates may span more than one database. You specify what action to take using a combination of key fields and rules in a separate database (link database). An exit is provided for special error handling.

The MQSeries link extra agent runs as a Notes agent on either your Domino server or on your Notes client.

The function provided in the MQSeries Trigger Monitor for Lotus Notes Agents allows an application on any of the MQSeries supported platforms to initiate the sending of data to the Lotus Notes Server or workstation. The MQLSX link extra agent sample allows data from a non-Notes environment to be used to update Lotus Notes databases. A link database provides mapping between the system application and fields defined in the Lotus Notes databases.

Components

The MQLSX link extra agent sample consists of the following components:

- Agent database (gmqlxtra.nsf)
- The link database (mqlinkx.nsf)
- Lotus Notes sample applications databases (test1.nsf & test2.nsf)
  - test1.nsf contains salary information
  - test2.nsf contains address information
- Test application files (mqlxdemo.*)

Lotus Notes agent database (gmqlxtra.nsf)

As well as the agent program, this database contains a Notes document providing the default parameters needed if the agent is not started by the trigger monitor, or the default parameters used by the trigger monitor.

The defaults provided are:

QName:	NOTE
QMgrName:	Blank (Default queue manager)
Envdata:	mqlinkx.nsf
Envdata:	

Lotus Notes link extra database (mqlinkx.nsf)

This is a Lotus Notes database that you need to populate with entries to describe how your MQSeries message data maps to fields in a given Lotus Notes document. The Lotus Notes link extra database is an extension of the MQSeries link for Lotus Notes database and can be used to define entries for both MQSeries link and MQSeries link extra by using different forms.

The information held in a document on the database includes:

- The entry name from the link database document. This name matches that of the MQSeries queue where messages to be processed by MQSeries link extra agent are retrieved.
- The name of an optional view to be used to search for a Notes document (a faster alternative to searching the entire database). If this option is used, the first column of the view must contain the data for the first key field specified for that entry.
- Logical processing rules (e.g. If found Insert, Insert always, etc.) for each message retrieved.

Note If you are already using the MQSeries link extra SupportPac, you can use exactly the same database with this sample.

Lotus Notes sample applications databases (test1.nsf & test2.nsf)

Two sample Lotus Notes demonstration databases are provided with MQSeries link extra agent sample.

- test1.nsf contains salary information
- test2.nsf contains address information

The Link database has entries to enable documents contained in the demonstration databases to be updated. If you create your own applications, suitable Link database entries must be provided.

Note The demonstration example does not automatically use the document search by view function.

MQSeries sample application (mqlxdemo files)

An MQSeries demonstration application is provided to help you get started. This demonstration application is a C program designed to read messages from standard input. A data file is provided for use with the demonstration application. When the data file is piped to the demonstration application, messages are put that the MQLSX link extra agent uses to update the Lotus Notes demonstration databases. The demonstration application can use triggering to start the MQLSX link extra agent.

Comparison with the MQSeries link extra for Lotus Notes SupportPac

This sample uses exactly the same

- Set of rules
- mqlinkx.nsf database

The differences are:

- The mqlinkx executable file is replaced by a Notes agent database (gmqlxtra.nsf).
- The MQLSX link extra agent sample can be run using a trigger monitor (the alternative is to run it manually or as a scheduled agent).
- When a message cannot be processed and the retry limit is exceeded, the MQLSX link extra agent sample commits the message in order to remove it from the queue. If there is a backout requeue queue or a dead-letter queue defined, the failing message is put on it.
- The 'Additional Selection Formula', an optional parameter in the mqlinkx.nsf database, must conform to the LotusScript FTSearch rules (whereas previously this conformed to the Notes formula rules).
- The 'Additional Selection Formula' parameter may be used in addition to the 'View to Use for Searching' (whereas previously it could not).

  Caution If you specify numeric KEY fields in the link database, you MUST create a full text index for the target database.

Restrictions

- The agent cannot be used with the MQSeries Trigger Monitor for Lotus Notes agents on UNIX platforms with versions of Notes 4.5 and earlier.
- User messages are only provided in US English.

Recommendations

- Triggered Notes agents should not be long-running; they should allow themselves to be retriggered.

Design of the MQLSX link extra agent sample

The MQLSX link extra agent sample provides the same function as MQSeries link extra for Lotus Notes, with the addition of application type (APPLTYPE) 22 to support the triggering of Notes agents.

The add-in task (mqlinkx.exe) provided by MQSeries link extra for Lotus Notes is replaced by a Notes Agent (gmqlxtra.nsf).

The trigger monitor uses the MQSeries MQTMC2 structure. The only fields in this structure used by the Notes agent are QName, QMgrName, and EnvData. The EnvData field is used to hold both the name of the link database and the link database server name.

Note The name of the Notes database may not included spaces, however the link database server may have a name that includes spaces. A space must be added in EnvData after the name of the database, before the name of the link database server.

The trigger message document (using the MQTMC2 structure) may originate from:

- A trigger document passed from the trigger monitor
- An equivalent document in the Agent database (when you run the Notes agent manually)
- A combination, with the queue and queue name parameters supplied within the document from the trigger monitor and the link database and server names provided by the document in the Agent database.

If neither document is found, the default values are used. If the value of a parameter is unspecified, the default value is used.

The QName field in the trigger message document, as with any document using the link database, is compared to the 'Entry' field in the link database. When a match is found, the trigger document is processed using the conditions and rules in the corresponding document in the link database. After successfully applying the rules of the link document, the link database is searched to find a further matching 'Entry'. Processing of the trigger document ends when no further matching entries are found.

Error processing

If the agent is unable to process a message, and the retry limit is exceeded, the Notes agent:

1. Attempts to queue the message on the backout requeue queue (the name of which is held by the application queue) after which it commits the message

If this fails:

2. Puts the message onto the dead-letter queue (if one is defined to the queue manager) after which it commits the message

If this fails:

3. Commits the message, causing it to be removed from the application queue and discarded.

For information on triggering, see the *IBM MQSeries Application Programming Guide*.

Notes agent user exit

The Notes agent provides you with the option of using a LotusScript user (error) exit. The sample user exit provided is invoked when an error is detected. It outputs a message for information (the value of the MSG_USEREXITINVOKED string constant), and returns "Stop and Exit", at which it will stop the agent. An alternative return value SCERR_OK_TO_CONTINUE can be used to indicate that the exit has successfully handled the error and it is OK for the agent to continue.

```
'*********************************************************
'*  Sample LotusScript UserExit function
'*
'*  UserExit input parameters:
'*  Rule    As String
'*  Rule (one of: "0","1","2","3","4","5","6","7","8","9" )
'*  RuleDoc As NotesDocument  rule document
'*  Msg     As MQMessage      MQMessage
'*  QMgr    As MQQueueManager MQQueueManager
'*  UserDb  As NotesDataBase  User database
'*  UserExit As Long returns one of:
'*  SCERR_OK_TO_CONTINUE:
'*      the message was processed successfully and should
'*      be committed
'*  SCERR_STOP_AND_EXIT:
'*      the message was not processed successfully and
'*      be rolled back. The agent should terminate.
'*********************************************************


Function UserExit( Rule As String,  RuleDoc As NotesDocument,
Msg As MQMessage,
               QMgr As MQQueueManager, UserDb As
NotesDataBase) As Long
     Messagebox MSG_USEREXITINVOKED
     Print MSG_USEREXITINVOKED


     UserExit = SCERR_STOP_AND_EXIT
'*   UserExit = SCERR_OK_TO_CONTINUE
End Function
```

Setting up the MQLSX link extra agent sample

If you are installing the MQLSX link extra agent sample for the first time, follow the instructions under "Installing the MQLSX link extra agent sample for the first time".

If you already have the MQSeries link extra for Lotus Notes support pac installed on your system, follow the instructions under "Upgrading from the MQSeries link extra for Lotus Notes support pac".

Installing the MQLSX link extra agent sample for the first time

1. Copy the four Lotus Notes database files that are provided in the MQSeries link extra agent sample for Lotus Notes package into a suitable directory on your Notes workstation:
   - mqlinkx.nsf
   - gmqlxtra.nsf
   - test1.nsf (Salary Information database)
   - test2.nsf (Address Information database)

   Note These databases (test1.nsf and test2.nsf) do not use the document search by view function.
2. Add icons for these databases to your workspace.
3. Install the databases on your server if required using File - Database - New Copy and add the icons to your workspace.

Upgrading from the MQSeries link extra for Lotus Notes SupportPac

Install only gmqlxtra.nsf, test1.nsf, and test2.nsf as described when installing for the first time. The other databases are already available as part of the MQSeries link extra for Lotus Notes SupportPac.

Setting up MQSeries to run the MQLSX link extra agent sample

1. Start the MQSeries queue manager
2. Edit the MQSeries MQSC file command, gmqstrg0.tst (MQSeries Trigger Monitor for Lotus Notes), to match your environment. Replace the existing versions of test1.nsf and test2.nsf with the new versions, to take advantage of the performance benefits, as these databases specify a view name for searching.

   The user exit requires a queue to put error message details. The error queue name suffix must be the same as the name of the application queue name you are putting messages to. If you want to run the sample program mqlxdemo.exe, the MQSeries definition is already set up to use "NOTE" as the application queue name and error message queue name suffix. If you want to create other application queues, you must modify the gmqstrg0.tst file to maintain this relationship.
3. Use runmqsc to create the MQSeries link extra agent queue manager objects. Issue the command:

   ```
   runmqsc QMgrName < gmqstrg0.tst > mqlx.out
   ```

   Where:

   - QMgrName is the name of the queue manager; if you don't specify a value the default queue manager is used.
   - gmqstrg0.tst is the name of your MQSC command file.
   - mqlx.out is the name of the file that contains the runmqsc output results.
4. Check the output of runmqsc. The following queue manager objects should now exist:

   Queues:

   ```
   NOTE           (The MQSeries link extra agent application
   queue*)
   MQLX.HOLD.NOTE (The user exit error message queue)
   SYSTEM.SAMPLE.NOTES.AGENT.INITQ   (The initiation queue)
   ```

   Process:

   ```
   SYSTEM.SAMPLE.NOTES.AGENT.PROCESS (Process definition to
                 start MQSeries link extra agent process
   mqlinkx.exe)
   ```

   Note * The value of the ENTRY field used to search the MQSeries link database will be equal to the queue name.

Before you run the MQLSX link extra agent sample

There are a number of steps you must complete before you can successfully run the MQLSX link extra agent sample.

You can either run the MQSeries link extra agent sample manually, as a scheduled agent, or by using triggering using the MQSeries Trigger Monitor for Lotus Notes Agents. If you choose to run the MQSeries link extra agent sample using triggering, you should verify that MQSeries link extra agent can initiate updates to Lotus Notes.

If you want to start it by using triggering with the MQSeries Trigger Monitor for Lotus Notes Agents, you must:

1. Start the MQSeries queue manager
2. Create an MQSeries application program queue
3. Create an MQSeries agent initiation queue
4. Create an MQSeries agent process definition
5. Start the Trigger Monitor program

Start the MQSeries queue manager

Before you can create the queues you need to run this sample, you must start your MQSeries queue manager. This can be the default queue manager, or a user defined queue manager.

Create an MQSeries application program queue

You can create an MQSeries application program queue by using the supplied MQSC sample script (gmqstrg0.tst) or by using the runmqsc command shown below:

```
DEFINE QLOCAL('NOTE') REPLACE +
DESCR('Sample Notes Agent Application Program Queue') +
INITQ('SYSTEM.SAMPLE.NOTES.AGENT.INITQ') +
PROCESS('SYSTEM.SAMPLE.NOTES.AGENT.PROCESS') +
TRIGGER +
TRIGTYPE(FIRST) +
```

Definitions	
DESCR	Specifies a descriptive plain-text comment.
INITQ	Specifies the name of the initiation queue.
PROCESS	Specifies the name of the agent process definition.
TRIGGER	Specifies that triggering is active for this queue.
TRIGTYPE(FIRST)	Specifies that a trigger message should be written when the first message of suitable priority arrives on this queue.
TRIGDATA	Specifies information required by the agent to process messages from this queue. For the IBM MQSeries link LotusScript Extension (MQLSX), this optional.

Note For the MQSeries link LotusScript Extension, TRIGDATA is typically not use and can be specified as blank or omitted if not required.

Create an MQSeries agent initiation queue

You can create an MQSeries agent initiation queue by using the supplied MQSC sample script or by using the runmqsc command shown below:

```
DEFINE QLOCAL('SYSTEM.SAMPLE.NOTES.AGENT.INITQ') REPLACE +
DESCR('Sample Triggered Notes Agent Initiation Queue')
```

Definitions	
DESCR	Specifies a descriptive plain-text comment.

Create an MQSeries agent process definition

You can create an MQSeries agent process definition by using the supplied MQSC sample script or by using the runmqsc command shown below:

```
DEFINE PROCESS('SYSTEM.SAMPLE.NOTES.AGENT.PROCESS') REPLACE +
DESCR('Sample Triggered Notes Agent Process') +
APPLTYPE(22) +
APPLICID('gmqlxtra.nsf MQLSX link extra Agent') +
USERDATA('User data used by MQLSX link extra Agent')
```

Definitions	
DESCR	Specifies a descriptive plain-text comment.
APPLTYPE	APPLTYPE 22 specifies the application type as a Notes agent.
APPLICID	Specifies the name of the agent database (*.nsf) file followed by the agent name. Note that the agent name may contain spaces while the agent database file name may not. You must specify the fully qualified path and filename of the agent database if it is not the default Notes directory.
USERDATA	Specifies user data required by the agent. You can specify this as a blank or just omit it if you don't require it.

Start the Trigger Monitor program

You may want to start the Trigger Monitor program before you can run the MQLSX link extra agent.

On MQSeries for UNIX systems, OS/2 or Windows NT, the command is the same:

```
runmqtnm [-m QManagerName] [-q InitiationQueueName]
```

Note runmqtna is not supported on Windows 95.

If you are using an MQSeries client, the command is:

```
runmqtnc [-m QManagerName] [-q InitiationQueueName]
```

Where QManagerName is the name of your MQSeries queue manager and where InitiationQueueName is the name of your MQSeries initiation queue.

Note There is no trigger monitor for Windows 3.1.1 MQSeries clients.

Verifying MQSeries link extra agent can initiate updates to Lotus Notes

If you have chosen to start the MQSeries link extra agent by triggering, you should verify that MQSeries link extra agent can initiate updates to Lotus Notes. Two sample demonstration Lotus Notes databases are provided. Running the sample demonstrates how separate documents in two different Lotus Notes databases can be updated by putting messages on a single queue.

1. Start Lotus Notes.

   You should have the MQSeries link extra agent workspace icons representing the link database and two sample application databases. If you do not, add the icons for the three Notes databases to your workspace.

2. Start the MQSeries queue manager.

   Use runmqsc to check to see if the following queue manager objects are defined. If not, go back and perform the MQSeries Setup procedure.

   Queues:

   ```
   NOTE           (The MQSeries link extra agent application
   queue*)

   MQLX.HOLD.NOTE (The user exit error message queue)

   SYSTEM.SAMPLE.NOTES.AGENT.INITQ   (The initiation queue)
   ```

   Process:

   ```
   SYSTEM.SAMPLE.NOTES.AGENT.PROCESS (Process definition to

                 start MQSeries link extra agent process
   mqlinkx.exe)
   ```

   Note
   * The value of the ENTRY field used to search the MQSeries link database will be equal to the queue name.

3. Start the trigger monitor by issuing the runmqtrm command.

   ```
   C:> runmqtnm -m [QMgrName] -q
   SYSTEM.SAMPLE.NOTES.AGENT.INITQ

          or

       runmqtnc -m [QMgrName] -q
   SYSTEM.SAMPLE.NOTES.AGENT.INITQ

                                 (trigger monitor for client)
   ```

   Where [QMgrName] is the name of your MQSeries queue manager and SYSTEM.SAMPLE.NOTES.AGENT.INITQ is the name of the initiation queue the trigger monitor will monitor.

Note There is no trigger monitor for Windows 3.1.1 MQSeries clients.

4. Change to the directory where your mqlxdemo program file is stored.
5. Run the mqlxdemo program to put sample data on a queue called NOTE.

For example:

```
mqlxdemo NOTE [QMgrName] < mqlxdemo.dat
```

Where [QMgrName] is the name of the queue manager. If you do not specify a value here, the system default queue manager is used.

The application mqlxdemo reads the sample data in the mqlxdemo.dat file and writes the MQSeries messages to the NOTE queue. The NOTE queue being a triggered queue will cause the process defined by SYSTEM.SAMPLE.NOTES.AGENT.PROCESS to run the agent.

The agent,for each message retrieved, reads the link database to determine the processing rule, and the Lotus Notes document and fields to update. It uses the queue name as the name of the ENTRY identifier in the link database.

6. Open a view in the Salary (test1.nsf) or Address (test2.nsf) database. You should now see new document entries reflecting the data processed.

You can also run the agent program manually or as a scheduled agent (for example On Schedule Hourly) to perform these updates instead of using triggers.

Running the MQLSX link extra agent sample

Before you run the sample, make sure that you have completed all the steps described in "Before you run the MQLSX link extra agent sample".

The MQLSX link extra agent sample may be started via triggering, as a scheduled agent, or run manually. The name of the queue used must match the name of the link database entry.

It is necessary for the appropriate MQSeries subsystem to be installed wherever the agent triggered task will execute. This could be either the MQSeries server or MQSeries client.

To run the MQLSX link extra agent sample manually

To run the sample manually you need:

gmqlxtra.nsf

mqlinkx.nsf

test1.nsf

test2.nsf

mqlsxdemo

mqlsxdemo.dat

Having checked that these are installed on your system:

1. Start MQSeries by entering the command: **strmqm** *QMgrName*
2. Modify the MQSeries agent parameters document in the agent database for your environment, specify:
   queue name
   link database name (if different from mqlinkx.nsf)
   queue manager name (if you are not using the default queue manager)
3. Select Agents - MQLSX Link Extra Agent
4. Select Actions - Run

Using full text indices and views

If a target database has a full text index, existing documents are located more quickly but because the index is updated by the agent each time it creates a new document, document creation takes longer.

Whether an application performs better with or without a full text index therefore depends on the characteristics of the particular application.

Note If you are using numeric key fields to locate an existing document, the target database MUST have a full text index.

On a server, Notes periodically updates a full text index automatically. If the frequency of this update is adequate for your application, you can improve the performance of document creation in a database with a full text index, by removing the agent update statement from the LotusScript declarations event section.

This can be accomplished by commenting out the line by inserting a single quotation mark (') at the start of the line:

```
' Call oUserDb.UpdateFTIndex( False )
```

If a database is updated manually (e.g. by deleting or adding documents) you must ensure the full text index is deleted and recreated before running the agent.

Better performance is generally obtained if a View is specified to locate existing documents, particularly if the database has no full text index.

Using a Notes view to search for a document

If you have a large number of documents in the database that are to be updated with information from your host system, you are recommended to use a view to search for documents. However, you must adhere to the conditions attached to using views.

- A view must have at least one column sorted in ascending order.
- The first column of the view must contain the data for the first key field specified for that entry.

How the MQLSX link extra agent sample works

The MQLSX link extra agent sample demonstrates how you can use the MQSeries Trigger Monitor for Lotus Notes agents to trigger a Lotus Notes Agent called "MQSeries link extra agent" that is contained in the Notes database, gmqlxtra.nsf.

The MQSeries link extra agent makes use of the MQLSX. You should have the appropriate MQSeries subsystem installed wherever the agent will run.

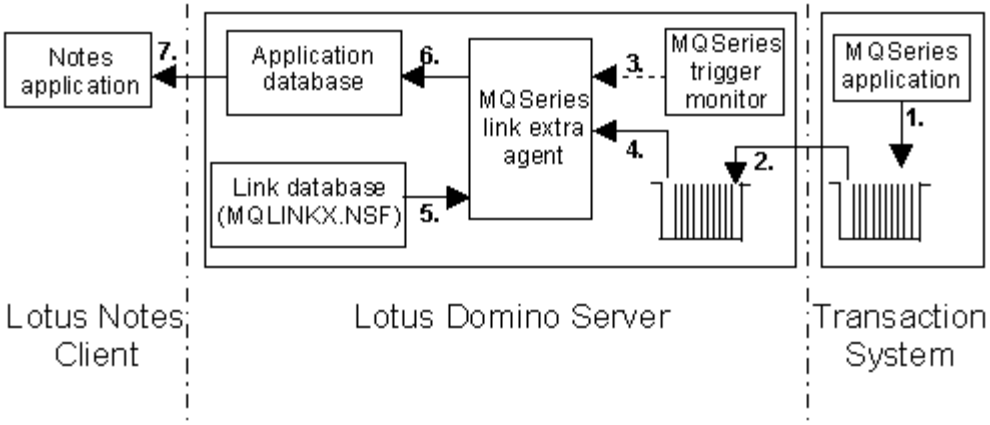
When triggering is required, an application queue with triggering turned on is needed with an associated process definition. The triggered queue has a defined process that starts the MQSeries link extra agent. The MQSeries link extra agent processes messages placed on an input queue and consults the Lotus Notes link database for the processing rule and the Notes document and fields to update. The MQSeries link extra agent program may call a customized LotusScript user exit.

The agent gets an MQSeries message from the specified queue and, for each link entry rule document with an Entry field matching the queue name, applies the update defined by the rule to the user database specified by that rule document.

Note A single message may match more than one rule document and a single rule document may select more than one user document for update.

Looking at the steps that take place if your MQSeries application, where the data to be passed to Notes originates, is running on a system remote from your Notes server:

Using the MQSeries link extra agent sample



1. An MQSeries application running on your enterprise system generates a message containing the data it wants to pass to the Notes environment, and puts it on a queue, defined as a tranmission queue.
2. MQSeries, running on the Domino server extracts the message and puts it on a local application queue. This local queue has been defined as a triggered queue.
3. The trigger monitor program recognizes a message has arrived on the queue, carries out the normal triggering checks, and if all the criteria are met, issues a call to run the Notes agent.
4. The MQSeries link extra agent gets the message from the application queue.
5. It reads the document in the link database, the queue name as the key (to match against the Entry field), for each message and determines the processing rules.
6. The appropriate Notes document is updated or created, as determined by the processing rules.
7. The Lotus client application opens the database and views the data. There is no dynamic display of any changed data, as it is the enterprise application initiating the change.

Link extra database contents (mqlinkx.nsf)

The link extra database contains the information needed by the Notes agent to determine what processing it needs to do for an incoming message:

- Entry
  The name of the MQSeries queue from which the Notes agent gets messages
- Description
  Text that is useful to you. For example, you may want include the type of information held in the messages, or the name of the transmission queues that supply the messages
- Database Information
  User Database Name: The name of the Notes database to be updated with the data in the MQSeries message
  View to Use for Searching: The name of a Notes view known to your Notes application that identifies the document to be updated. This must be shared view.

- Rule
  There are a set of predetermined rules from which you must select one. The rule you select (there is no default rule) determines how the Notes agent processes the MQSeries message, based on documents that already exist in the Notes database.
  In summary:
  Insert: A new document is created
  Update: The located document is updated
  Ignore: No action is taken
  Fail: A call is made to the user exit
  The rules are:

Rule no.	Condition	Action taken
0	If found insert, else fail	When no documents are found, call the user exit.
1	If found update, else fail	When no documents are found, call the user exit.
2	If found fail, else insert	When one or more documents are found, call the user exit.
3	If found update, else insert	When a matching document is not found, a new one is created, the user exit is not called.
4	If found update, else ignore	When no matching document is found, the message is ignored or discarded and the user exit is not called.
5	Insert always	A new document is always created regardless of whether a matching document is found, the user exit is not called.
6	If found ignore, else fail	When no matching documents are found, call the user exit.
7	If found insert, else ignore	When no matching documents are found, the message is ignored or discarded and the user exit is not called.
8	If found fail, else ignore	When one or more matching documents are found, call the user exit.
9	If found ignore, else insert	When a matching document is found, the message is ignored or discarded and the user exit is not called.

- Output Offsets
  Describes the layout of the data in the MQSeries message:
  Syntax:
  FieldName <space> Start<space>
  End<space>CHAR¦INTEL-BINARY<space>KEY<comparison>

Syntax	
Fieldname	Optional. The name of the field in the Notes form to be updated with data in this part of the MQSeries message
Start	The position in the MQSeries message at which the data in this field starts
End	The position in the MQSeries message at which the data in this field ends
CHAR ¦ INTEL-BINARY	The datatype of the field. Use CHAR for character data, and INTEL-BINARY for numeric data
KEY	Optional. Use the word KEY against a field to indicate that this field is of the selection criteria when searching for a document in a Notes database. **Caution** If you specify numeric KEY fields in the link database, you MUST create a full text index for the target database.
<comparison>	Optional and only used with a KEY field. A token that you set to <, >, +, =, CASE or NOCASE. When selected, is used to compare the field in the Notes document with the field in the MQSeries message. The default value is NOCASE. **Note** Use the word CASE to indicate that the search must be case sensitive. Use the word NOCASE to indicate that the search is not case sensitive.

- Additional Selection Formula
  Optional. Used to specify search criteria when searching for documents in a Notes database. All conditions must conform to the LotusScript FTSearch verb.

Note When KEY fields are not specified and the Additional Selection Formula is blank, all documents in the database are selected.

Error handling and status reporting in the MQLSX link extra agent sample

Messages are output by the sample both to report errors and provide status information. All messages are output by the trigger monitor to the Notes status bar or server console.

Status messages

The status messages that you may see in the Notes status bar when running the MQLSX link extra sample are:

Message	What it means...
Starting MQLSX link extra agent Version n.n	The Agent program is starting.
Using default parameters	The Agent could not find an agent parameter document is using the default parameters.
Using default queue manager	The Agent is using the default MQSeries queue manager.
Using default queue name	The Agent is using the default queue name.
Using default link database	The Agent is using the default link database.
Using parameters from agent parameter document.	The agent could not find a trigger message document and is using the agent parameter document.
Using parameters from trigger message document	The Agent is using parameters from the trigger message document.
User exit invoked	The user exit was invoked.
WARNING: No messages were found on the specified MQSeries queue	
End of messages	
Agent ended abnormally	
Agent ended normally	

Error Messages

Error messages output by the sample are detected by LotusScript, the MQLSX, or the agent. The format in each case is different.

Errors detected by Lotus Notes
The format of these messages output by the link extra agent sample is:

MQLSX link extra Agent Notes Error: Agent_Insert_String
Notes_Error_Message (Notes error = Notes_Error_Number Line nnnn)

where:

Agent_Insert_String
is any additional useful information about the error that the agent can provide and may be omitted if blank.

Notes_Error_Message
is the LotusScript error message

nnnn
is the line number in the LotusScript source code where the error occurred

Errors detected by the MQLSX
The format of these messages output by the link extra agent sample is:

MQLSX link extra agent Error: Agent_Error_Number
Agent_Error_Message Agent_Insert_String Notes_Error_Message (Notes error = Notes_Error_Number Line nnnn)

where:

Agent_Error_Number
is an agent error number that may be omitted for MQLSX detected errors if zero

Agent_Error_Message
is an agent error message

Agent_Insert_String
is any additional useful information about the error that the agent can provide and may be omitted if blank.

MQLSX_Error_Message
is the LotusScript MQLSX error message and will typically provide the MQSeries or MQLSX reason code

Notes_Error_Message
is the LotusScript error message

nnnn
is the line number in the LotusScript source code where the error occurred

Errors detected by the link extra agent sample
The format of these messages output by the link extra agent sample is:

MQLSX link extra agent Error: Agent_Error_Number
Agent_Error_Message Agent_Insert_String

where:

Agent_Error_Number
is an agent error number that may be omitted for MQLSX detected errors if zero

Agent_Error_Message
is an agent error message

Agent_Insert_String
is any additional useful information about the error that the agent can provide and may be omitted if blank.

The error messages you can encounter that are detected by the Agent component of the Link sample are:

Message	What it means...	Action
Link database could not be opened. (66001)	An error occurred opening the Link database.	Check that the Link database is available and its specification is correct.
User database could not be opened. (66002)	An error occurred opening the target user database.	Check that the target user database is available and its specification is correct.
Link Entry Error: start or end position value not valid. (66003)	The error was detected in the Link database field definition shown.	Correct the incorrect value (or values) and retry the operation. The start offset must be greater than zero and the end offset must be equal to or greater than the start offset.
Link Entry Error: one or more field items are missing. (66004)	The error was detected in the Link database field definition shown.	Correct the missing value (or values) and retry the operation.
Link Entry Error: data type not valid. (66005)	The error was detected in the Link database field definition shown.	Correct the incorrect value (or values) and retry the operation. Valid data types are CHAR, NUM, (and INTEL_BINARY).
Link Entry Error: data length not valid for NUM. (66006)	The error was detected in the Link database field definition shown.	Valid data lengths for NUM are 2, 4 and 8. Correct the incorrect value and retry the operation.
Link Entry Error: KEY position value not valid. (66007)	The error was detected in the Link database field definition shown.	Correct the incorrect value (or values) and retry the operation. Values other than "KEY" are not valid in the 'key' position.

Message	What it means...	Action
Link Entry Error: comparison position value not valid. (66008)	The error was detected in the Link database field definition shown.	Correct the incorrect value (or values) and retry the operation. Values other than "CASE" or "NOCASE" are not valid in the 'comparison' position.
Link Entry Error: form specification not valid. (66009)	The error was detected in the Link database field definition shown.	Correct the incorrect value (or values) and retry the operation. 'Key' and 'comparison' keywords and datatypes other than CHAR cannot be specified for a field used to specify the form name (i.e., with field name 'FORM').
Link Entry Error: field length value too big. (66010)	The error was detected in the Link database field definition shown.	Correct the incorrect value (or values) and retry the operation. Field lengths greater than 32000 are not supported.
Link Entry Error: key field length value is too big. (66011)	The error was detected in the Link database field definition shown.	Correct the incorrect value (or values) and retry the operation. Key field lengths greater than 64 are not supported.
Error saving Notes document. (66012)	An unexpected error occurred when attempting to save a document in the target Notes database.	Check that your target Notes databases can be successfully accessed and check that there is is sufficient system resources (e.g., disk space).
A number in an MQSeries message had an invalid exponent. (66013)	The exponent value of an 8-byte NUM (double precision) number is not valid.	Refer the problem to the provider of your server application.

Message	What it means...	Action
A rule document with the specified name was not found. (66014)		Check that a rule document exists for application queue name shown.
The view specified in the Link Entry rule document was not found. (66015).		Check that a view exists in the target database for the view name shown.
A Invalid rule was detected. (66016)		Replace the rule shown in the Link entry document with a supported value.
Backout threshold of failing message exceeded; message could not be requeued and was discarded. (66017)	A failing message was discarded because, after backing out and retrying the required number of times, it could not be successfully requeued on either the backout requeue queue or the dead letter queue.	Refer to the previous error messages displayed whilst running the link extra agent sample to determine the cause of the problem. Create a backout requeue queue or a dead-letter queue if required.
Backout threshold of failing message exceeded; message was requeued to: (66018)	A failing message was requeued to the queue indicated because, after backing out and retrying the required number of times, the message could not be processed successfully.	IRefer to the previous error messages displayed whilst running the link extra agent sample to determine the cause of the problem and retry the operation if required. **Note** If the message was requeued to the dead letter queue the message data will be prefixed by a dead letter header.
User exit returned "Stop and exit". (66019)	A user exit call specified by the processing Rule returned "Stop and Exit". Processing is ended.	Investigate why the user exit was invoked and returned the completion code shown if this was not the intended action.

Message	What it means...	Action
Numeric key fields are not valid with a target database that is not full text indexed. (66020)		Create a full text index in the target database or change the key field to use type CHAR.
The link database view shown was not found. (66021)	The view required to access information in the link database was not found.	Create the required view in the link database.

Agent_Insert_String

The Agent_Insert_String may be blank or one of the following:

MQLSX error getting message from application queue

Consult your MQSeries Application Programming Reference or MQSeries link LotusScript Extension User Guide for the reason code shown.

MQLSX message DataOffset error.

Check that your field definition values in the link database are correct for the incoming messages. Consult your MQSeries Application Programming Reference or MQSeries link LotusScript Extension User Guide for the reason code shown.

MQLSX message ReadString error.

Check that your field definition values in the link database are correct for the incoming messages. Consult your MQSeries Application Programming Reference or MQSeries link LotusScript Extension User Guide for the reason code shown.

MQLSX message ReadShort error.

An MQLSX error occurred when attempting to read a short integer from an MQMessage. Consult your MQSeries Application Programming Reference or MQSeries link LotusScript Extension User Guide for the reason code shown.

MQLSX message ReadLong error.

An MQLSX error occurred when attempting to read a long integer from an MQMessage. Check that your field offset values in the link database are correct for the incoming messages. Consult your MQSeries Application Programming Reference or MQSeries link LotusScript Extension User Guide for the reason code shown.

Error accessing queue manager

An MQLSX error occurred when attempting to access the queue manager. Check your target queue name is correct and the queue manager is started. Consult your MQSeries Application Programming Reference or MQSeries link LotusScript Extension User Guide for the reason code shown.

Index

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Sending your comments to IBM

MQSeries link LotusScript Extension

User's Guide - Release 1.3

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

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

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

- By fax:
  - From outside the U.K., after your international access code use 44 1962 870229
  - From within the UK., use 01962 870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: WINVMD(IDRCF)
  - Internet: idrcf@winvmd.vnet.ibm.com
  - Lotus Notes: idrcf@winvmd.vnet.ibm.com

Whichever you use, ensure that you include:

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