MQSeries Enterprise Integrator for Lotus Notes

User Guide

Release 1.0

Copyright

Original: Original book produced for IBM MQSeries Enterprise Integrator for Lotus Notes.

Preface

The MQSeries Enterprise Integrator for Lotus Notes User's Guide describes the IBM MQSeries Enterprise Integrator LotusScript Extension (MQEI LSX), and shows you how you can use it in your Lotus Notes applications.

Information in this book includes:

- What you need to consider before you install the MQEI LSX
- How to use the MQEI databases
- Guidance on how to design and program your applications using the MQEI LSX
- Code samples and how you can use them in your own applications
- Help if you have problems using the MQEI LSX
- A full reference guide to the MQEI LSX classes and their use
- Where to find more information about MQSeries™, Lotus Notes™, LotusScript™ CICS™ and IMS™

This book covers the MQEI LSX 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.

Contents

Chapter 1 MQSeries Enterprise Integrator for Lotus Notes

This book:

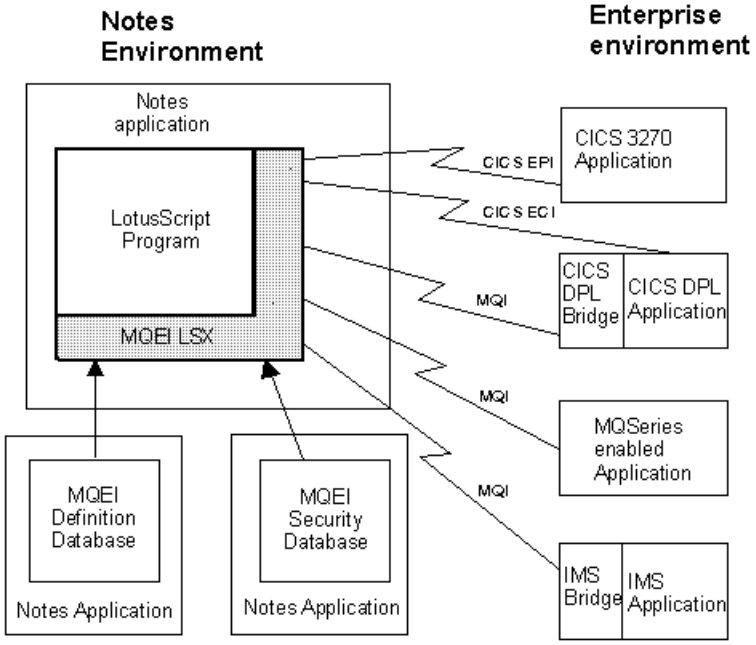
- Explains what you need to know before you install the MQSeries Enterprise Integrator for Lotus Notes (MQEI)
- Suggests how to get started
- Explains how to use the MQEI databases
- Explains where you need to consider a specific enterprise service when designing and programming your application.
- Provides you with help if you have problems using the MQEI
- Describes each of the MQEI LSX classes with their properties and methods
- Describes the samples that are provided

MQSeries Enterprise Integrator for Lotus Notes overview

MQSeries Enterprise Integrator for Lotus Notes (known in this book as MQEI) is a convenient way of accessing your organization's enterprise applications through a familiar Lotus Notes interface running on your workstation, or from a web browser if you are using a Lotus Domino server.

Enterprise applications are typically reliable, high-volume, high performance applications that you use to run your business. They might include CICS or IMS for example. The user interfaces for these enterprise applications are very likely to vary from system to system. The MQSeries Enterprise Integrator LotusScript Extension (known in this book as MQEI LSX) enables you to integrate your enterprise applications using a set of classes that provide a common application programming interface (API).

Components of the MQEI LSX



Example of the usefulness of MQEI

Consider an organization that has two main customer account applications. Both of these applications were written at different times and run under different transaction management systems. For example, one may run under IMS and the other may use CICS. Imagine the following scenario:

1. A customer telephones your organization to notify you of a change of address.
   This will require your organization to update all its databases that hold records of that customer.
2. To update the databases where the customer record is held, you have to log on to both the IMS and the CICS systems.
3. To change the address, you have to run the "Update Customer" transaction which is specific to each system.
   It is highly likely that you will have to enter the new addressing information twice (once on each system).
   It is also very likely that the two user interfaces will be very different and both systems accept totally different address formats.

MQSeries Enterprise Integrator for Lotus Notes will allow you to enter the new address once only, using a familiar Lotus Notes user interface or a web browser. It will then access the IMS and CICS systems on your behalf, and update the databases on those enterprise systems.

Features of the MQEI

The MQEI provides:

- Transparent access from LotusScript applications to a range of enterprise systems:
  - Any MQ-enabled application
  - CICS DPL, using the MQSeries CICS DPL bridge
  - CICS DPL, using the CICS client interface
  - CICS 3270, using the CICS client interface
  - IMS, using the MQSeries IMS bridge
- An MQEI Definition database containing message definitions that can be sent to enterprise systems and definitions of the applications (or "services") that run on these systems
- Security integration from Notes to enterprise applications
- The ability to populate the MQEI Definition database from CICS BMS maps

Objectives of the MQEI

The objectives of the MQEI are to provide:

- Access from both the Lotus Notes client and Lotus Domino server. Using a Lotus Domino server also gives you web access to the MQEI.
- A natural extension to the LotusScript development environment.
- A 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.
- An extended range of enterprise systems that can be accessed. As well as native MQSeries applications, access is provided to IMS applications and CICS DPL programs via the respective MQSeries bridges to those products. Access to CICS DPL programs and CICS 3270 transactions is provided via the CICS client interfaces (the ECI and EPI), allowing direct connectivity to CICS. The MQEI LSX is designed to allow easy future addition of other enterprise systems.
- Message building facilities enabling the programmer to build and interpret messages by field name, and to create messages that may be sent to multiple destinations. A utility is provided to create message definitions from CICS BMS map source files.
- Security features including the ability to sign-on automatically to enterprise systems.

Components

The MQEI consists of the following components:

1. MQEI LSX API, comprising:
   - EISession Class
   - EIService Class
   - EIMessage Class
   - EISendOptions Class
   - EIReceiveOptions Class
2. MQEI Definition database, a Lotus Notes database containing Lotus Notes documents that define the various message formats and enterprise services. You can administer these documents in the same way as you would administer any other Notes databases.

3. MQEI Security database, a Lotus Notes database containing Lotus Notes documents that define the enterprise security parameters for enterprise system users. You can administer these documents in the same way as you would administer any other Notes databases.
4. CICS BMS map conversion utility that allows you to convert BMS map files into message and field definitions in the MQEI Definition database.
5. MQEI Samples database, a Lotus Notes database containing working sample applications that show you how to communicate with each of the supported enterprise systems using the MQEI LSX.

Every enterprise application that your LotusScript program can access is represented by an instance of an EIService object. Examples of enterprise services include:

- MQ-enabled application
- CICS DPL programs
- CICS 3270 transactions
- IMS transactions

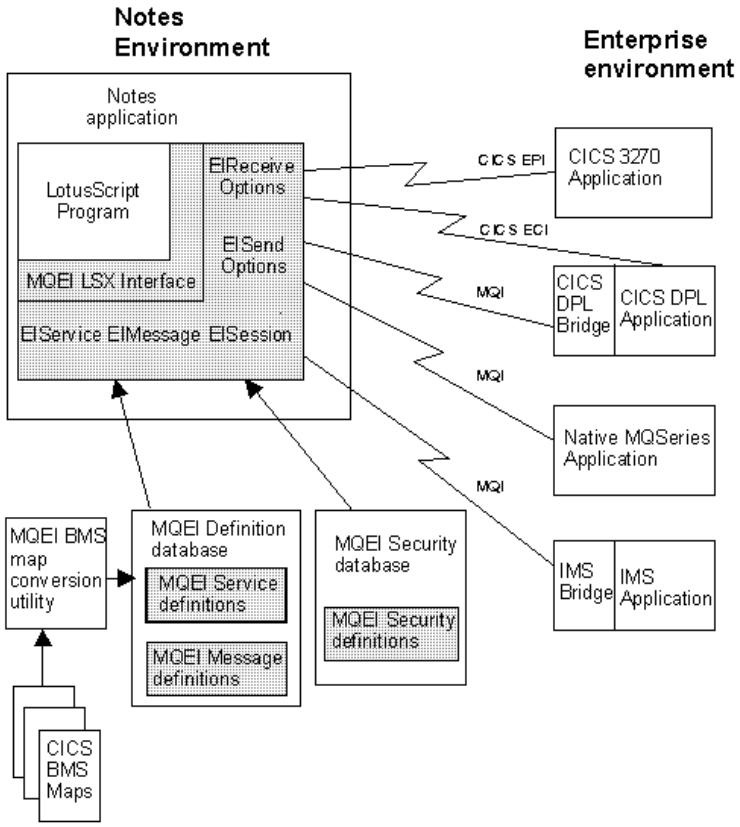
Communication, within an MQEI application, to an enterprise service is achieved using EIService methods to connect to the enterprise system and to send and receive EIMessage objects to and from the enterprise service.

An EIMessage object represents the actual data that is exchanged with the service and allows your LotusScript program to build and interpret this data, field by field, using field names, each field being a property of the EIMessage.

The EIService and EIMessage objects are built dynamically at run-time from enterprise services and message formats whose definitions reside on the MQEI Definition database. Variations on how an EIMessage is sent and received by the EIService are controlled by an EISendOptions object and an EIReceiveOptions object respectively.

Creation of all these objects in LotusScript is controlled by a singleton EISession class .

Components of the MQEI LSX



Environment specific information, such as the name of the MQEI Definition database and MQEI Security database, is held in an MQEI initialization file, mqei.ini.

About the MQEI samples

The sample applications provided, demonstrate how to communicate from Lotus Notes to the following enterprise systems:

- Any MQ-enabled application
  - Sample using a Native MQSeries service
- CICS DPL, using the MQSeries CICS DPL bridge
  - Sample using a CICS DPL via MQSeries service
- CICS DPL, using the CICS client interface
  - Sample using a CICS DPL direct service
- CICS 3270, using the CICS client interface
  - Sample using a CICS 3270 direct service (signon)
  - Sample using a CICS 3270 direct service (FILEA)
- IMS, using the MQSeries IMS bridge
  - Sample using an IMS via MQSeries service

For futher samples, information, and help, see the IBM Red book:

Lotus Notes and the MQSeries Enterprise Integrator, available at URL:

http://www.redbooks.ibm.com/SG242217/sg242217.html

or order book no: SG24-2217

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

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

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.

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.

Where to find more information

The following sections tell you where you can find more information about MQSeries, CICS, IMS and LotusScript.

Where to find more information about MQSeries

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

- *MQSeries An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Message Queue Interface Technical Reference*, SC33-0850
- *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 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
- *MQSeries System Administration*, SC33-1873-00

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

For the latest information about MQSeries, visit the MQSeries World Wide Web site at:

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

Where to find more information about CICS

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

- *CICS/ESA: Application Programming Reference*, SC33-0676
- *CICS/ESA: Application Programming Guide*, SC33-1169
- *CICS/ESA: CICS-RACF Security Guide*, SC33-1185
- *CICS Transaction Server for OS/390 CICS Server Support for CICS Clients*, SC33-1779
  (previously *CICS/ESA: Server Support for CICS Clients*, SC33-1591)
- *CICS Family: Library Guide*, GC33-0356
- *CICS Family: Client/Server Programming*, SC33-1435
- *CICS Clients: Administration Version 1.0*, SC33-1436
- *CICS Clients: Administration Version 2.0*, SC33-1792
- *CICS Transaction Server for OS/390 release 2 Problem Determination Guide*, GC33-1693-01
- *CICS Transaction Server for OS/390 release 1 Problem Determination Guide*, GC33-1693-00
- *CICS for MVS/ESA version 4 release 1 Problem Determination Guide*, SC33-1176-00
- *CICS/VSE version 2 release 3 Problem Determination Guide*, SC33-0716-02
- *CICS/VSE version 2 release 2 Problem Determination Guide*, SC33-0716-01
- *CICS for OS/2 Version 2.0.1 Problem Determination*, SC33-1005
- *CICS for OS/2 Version 3 Problem Determination*, SC33-1584
- *CICS on Open Systems Problem Determination Guide*, GC33-1565

For the latest information about CICS, visit the CICS World Wide Web site at:

http://www.hursley.ibm.com/cics/

Where to find more information about IMS

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

- *IMS/ESA Application Programming: Database Manager*, SC26-8015
- *IMS/ESA Application Programming: Design Guide*, SC26-8016
- *IMS/ESA Application Programming: Transaction Manager*, SC26-8017
- *IMS/ESA Customization Guide*, SC26-8020
- *IMS/ESA Master Index and Glossary*, SC26-8027
- *IMS/ESA Open Transaction Manager Access User Guide*, SC26-8026-01
- *IMS/ESA Operator's Reference*, SC26-8030
- *IMS/ESA Diagnosis Guide and Reference*, LY27-9620
- *IMS/ESA Messages and Codes*, SC26-8028

For the latest information about IMS, visit the IBM World Wide Web site at:

http://www.ibm.com/

Tip Use the IBM search engine to search for information about IMS.

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

For the latest information about LotusScript, visit the Lotus World Wide Web site at:

http://www.lotus.com/

Chapter 2 Getting Started

This chapter suggests how you might start to use the MQEI, starting with the information you need before you install it. The topics included are:

- Pre-installation considerations and requirements
- Possible system configurations
- Setting-up your MQEI initialization file
- Checking your installation of the MQEI
- Using the MQEI for the first time

Pre-installation considerations

Before you install the MQEI LSX, you need to make sure that you have the correct level of operating system and the correct level of Notes to run the MQEI.

The MQEI only runs in either the Lotus Notes Release 4.5.1 (or higher) client environment or the Lotus Domino Server 4.5.1 (or higher), Powered by Notes environment.

In addition to Notes, you need either the MQSeries (client or server) installed to enable you to connect to your MQSeries enterprise systems or you need to have the CICS client installed, to enable you to connect directly to your CICS enterprise systems.

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

MQEI LSX requirements

MQSeries requirements

If you use MQSeries to connect to an enterprise service, the MQEI LSX 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 Version 2.0 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. It provides significantly more function than an MQSeries client, by including a subset of the queue manager functions.
To differentiate this type of queue manager from that provided by other MQSeries products, the MQSeries for Windows queue manager is known as a leaf-node queue manager.

For more information see your *MQSeries for Windows User Guide.*

CICS requirements

If you are connecting directly to CICS enterprise systems, the MQEI LSX requires access to either a CICS client or a CICS server with a built-in client (from the following list) that is installed in the same environment:

- IBM CICS Client for OS/2
- IBM CICS Client for Windows
- IBM CICS Client for Windows 95
- IBM CICS Client for Windows NT
- IBM CICS for OS/2 (for the server with a built-in client)
- IBM Transaction Server for OS/2 Warp (for the server with a built-in client)
- IBM CICS for Windows NT Version 2 (for the server with a built-in client)

If you choose to use one of the CICS client environments, connect it to a CICS server that supports it. This does not have to be a server environment capable of running Notes.

Note
The MQEI LSX does not support connecting to the AIX Client for CICS/6000, the CICS Client for the Solaris Operating Environment, or the CICS Client for HP 9000. When using the CICS Client on Windows (that is, the CICS client that runs under Windows 3.1), it is not possible for the MQEI LSX to access CICS 3270 services. The MQEI LSX will not be able to access CICS 3270 direct services if you are connecting your CICS client to a CICS on System/390 server prior to CICS for MVS/ESA Version 4 Release 1 with PTF UN901412. It is not possible for the MQEI LSX to access CICS 3270 services and CICS DPL services from the Notes client or Domino server agent manager at the same time, using your CICS client.

Disk space requirements

The additional disk space requirements for the MQEI LSX executable code depend on the platform you are running:

AIX	10.5 MB
HP-UX	9.5 MB
OS/2	9.0 MB
Sun Solaris	10.5 MB
Windows 3.1	8.5 MB
Windows 95	9.0 MB
Windows NT	9.0 MB
Windows for Workgroups	8.5 MB
WIN-OS/2	8.5 MB

Plus:

- 3.0 MB is required for the on-line *MQSeries Enterprise Integrator User's Guide*, conversion tables, and the readme text
- 1.5 MB is required to hold the Adobe Acrobat PDF file (mqeihelp.pdf) version of the *MQSeries Enterprise Integrator User's Guide*

The MQSeries Enterprise Integrator package

The MQSeries Enterprise Integrator for Lotus Notes package is provided on a CD. For more information about installing the MQEI package on your system, see "Installing MQSeries Enterprise Integrator for Lotus Notes" later in this chapter.

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

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. It also contains information about installing the MQEI package on your operating system.
	mqei.ini	Sample MQEI initialization file
docs	mqeihelp.nsf	*MQSeries Enterprise Integrator User's Guide* as a Notes database.
	mqeihelp.pdf	The *MQSeries Enterprise Integrator User's Guide* as an Adobe Acrobat PDF file.
samples	mqeisamp.nsf	A Notes database containing the the Post-Installation Check Program and a sample for each enterprise system supported.
	mqeisamp.jcs	MQSeries for MVS/ESA Command File containing definition statements used by some of the samples.
	mqeisamp.tst	MQSeries MQSC Command File containing queue, and channel definitions to set up MQSeries to successfully run the samples.
database	mqeidata.nsf	MQEI Definition database. This is a Notes database containing MQEI definitions used by the MQEI Samples database.
	mqeidata.ntf	Lotus Notes design template for the mqeidata.nsf file.
	mqeisecu.nsf	MQEI Security database. This is a Notes database.
	mqeisecu.ntf	Lotus Notes design template for the mqeisecu.nsf file.

Directory	File name	What it is...
bin (AIX)	libeilsx.a	A directory containing the AIX 4 version of the MQEI LSX (MQEI LSX library). This is supported when used with the AIX Notes client or server code in the following environments: • MQSeries for AIX client running under AIX 4.1.4 or later • MQSeries for AIX server running under AIX 4.1.4 or later
	eilsxmqm	Dynamic load library for MQSeries server
	eilsxmqic	Dynamic load library for MQSeries client
	mqeilev	Code level service utility
	mqeibms	BMS Map utility
bin (HP-UX)	libeilsx.sl	A directory containing the HP-UX version of the MQEI LSX (MQEI LSX library). 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
	eilsxmqm	Dynamic load library for MQSeries server
	eilsxmqic	Dynamic load library for MQSeries client
	mqeilev	Code level service utility
	mqeibms	BMS Map utility

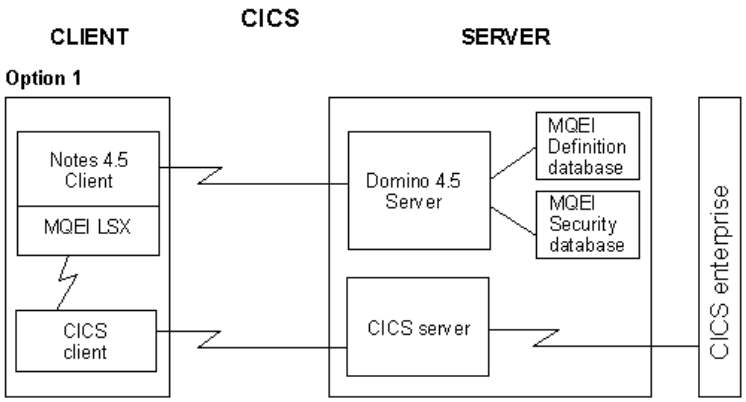
Directory	File name	What it is...
bin (OS/2)	EILSX.DLL	A directory containing the OS/2 version of the MQEI LSX (MQEI LSX library). This is supported when used with the OS/2 Notes client or server code in the following environments: • MQSeries OS/2 client running under OS/2 V2.1, OS/2 Warp, OS/2 Warp Connect, or OS/2 Warp server • MQSeries OS/2 server running under OS/2 V2.1, OS/2 Warp, OS/2 Warp Connect, or OS/2 Warp server
	MQEILEV.EXE	Code level service utility
	MQEIBMS.EXE	BMS Map utility
bin (Sun Solaris)	libeilsx.so	A directory containing the Sun Solaris (SPARC, Ultra SPARC) version of the MQEI LSX: • (MQEI LSX library) 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 2.5 or later • MQSeries Sun Solaris server running under Sun Solaris 2.5 or later
	eilsxmqm	Dynamic load library for MQSeries server
	eilsxmqic	Dynamic load library for MQSeries client
	mqeilev	Code level service utility

Directory	File name	What it is...
	mqeibms	BMS Map utility
bin (Windows 3.1, Windows for Workgroups and WIN-OS/2)	EILSX.DLL	A directory containing the Windows 3.1, the Windows for Workgroups, and WIN-OS/2 version of the MQEI LSX (MQEI LSX library). This is supported when used with 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 fow Workgroups • MQSeries for Windows leaf-node server running under WIN-OS/2
	MQEILEV.EXE	Code level service utility

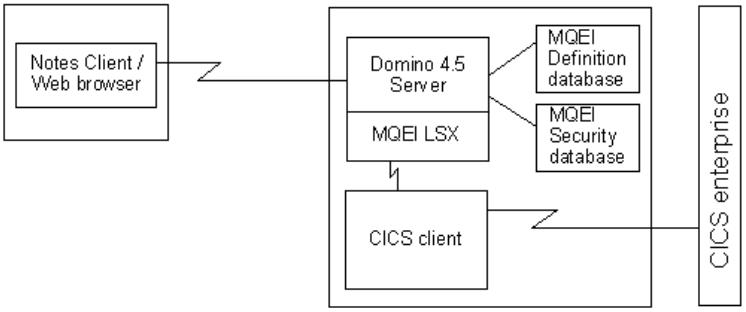
Directory	File name	What it is...
bin (Windows 95 and Windows NT)	EILSX.DLL	A directory containing the Windows NT and Windows 95 version of the MQEI LSX (MQEI LSX library). This is supported when used with Win32 Notes client or server code in the following environments: • MQSeries client on Windows NT 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
	MQEILEV.EXE	Code level service utility
	MQEIBMS.EXE	BMS Map utility
conv		A directory containing the files required to support character conversion.
	README.CCS	Details the supplied conversions
	MQEICCS.TBL	Used to establish allowed conversions
	nnnnmmmm.TBL	Table for supported conversions, where nnnn is the hexidecimal value of the coded character set identifier (CCSID) for the 'from' codepage, mmmm is the hexidecimal value of the CCSID for the 'to' codepage

Possible system configurations

The following configurations are possible if your applications use the CICS DPL direct service or the CICS 3270 direct service:



Note: The CICS server can be installed on a separate server machine.

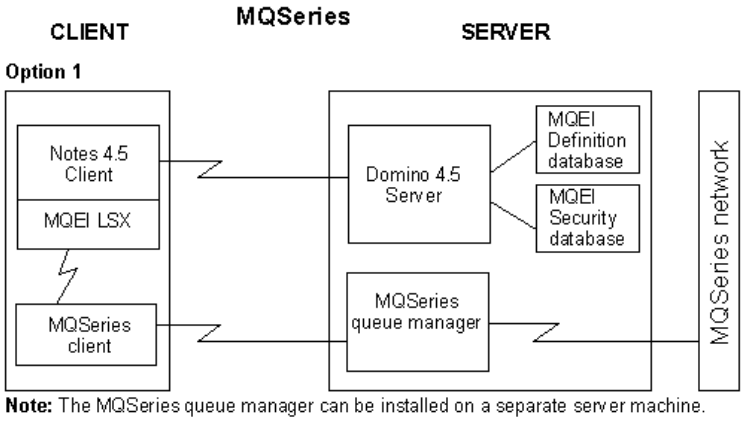


Option 3

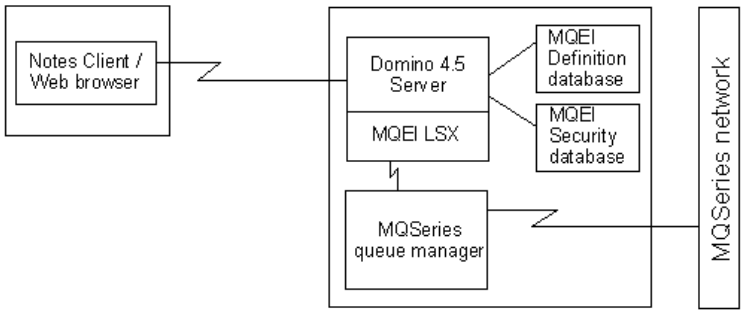
Similar to option 2 but install an CICS server on the server instead of a CICS client.
This is only possible for a CICS server with a built-in client (available on OS/2 and Windows NT).

If your enterprise system is AIX, HP-UX, or Sun Solaris, Option 1 is the only configuration available to you. The server must be a UNIX system, and the client must be OS/2, Windows 3.1, Windows 95, or Windows NT.

The following configurations are possible if your applications use the Native MQSeries service, IMS via MQSeries service or the CICS DPL via MQSeries service:



Note: The MQSeries queue manager can be installed on a separate server machine.



Option 3

Similar to option 2 but install an MQSeries client on the server instead of an MQSeries queue manager.

MQEI environment variables

There are eight environment variables that you need to know about when setting up the MQEI on your local system. You do not have to set them all.

- MQEI_INI_PATH
- MQEI_TRACE
- MQEI_TRACE_LEVEL
- MQEI_TRACE_PATH
- MQEI_XLAT_PATH
- MQEI_MQ_LIB
- MQEI_ECI_LIB
- MQEI_EPI_LIB

Note You must set the MQEI_INI_PATH environment variable. It points to where your mqei.ini file is located.

MQEI_INI_PATH

This environment variable is mandatory. You must set the MQEI_INI_PATH environment variable to locate your mqei.ini file. There must be one mqei.ini file on every system that you run MQEI LSX code and utility programs on.

For more information, see "Setting up your MQEI initialization file" later in this chapter.

MQEI_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 MQEI_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 11.

MQEI_TRACE_LEVEL

Use the MQEI_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 11.

MQEI_TRACE_PATH

If you have switched the trace facility on (using the MQEI_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 MQEI_TRACE_PATH environment variable, the traces files will be written to the current working directory.

You can identify a trace file by the .trc file extension.

For more information, see "Using trace" in Chapter 11.

MQEI_XLAT_PATH

If you use data conversion, you should set the MQEI_XLAT_PATH environment variable to locate the data conversion tables that are used by the MQEI LSX.

For more information, see "Data Conversion" in Chapter 4.

MQEI_MQ_LIB

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

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

Note On HP-UX, you *must* set the MQEI_MQ_LIB environment variable to enable libeilsx.sl to find eilsxmqm or eilsxmqic.

For more information, see "Dynamic loading and the MQEI LSX" in Chapter 11.

MQEI_ECI_LIB

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

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

For more information, see "Dynamic loading and the MQEI LSX" in Chapter 11.

MQEI_EPI_LIB

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

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

For more information, see "Dynamic loading and the MQEI LSX" in Chapter 11.

Installing MQSeries Enterprise Integrator for Lotus Notes

This section describes how to install the MQSeries Enterprise Integrator for Lotus Notes on your operating system.

Installing on AIX

Logged on as root:

1. Insert the MQEI CD_ROM into your CD-ROM drive, unless you 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'
4. Press List to get a list of all available software. To install all of the MQEI components, select the line showing '1.0.0.0 mqei', press enter. If you want to install specific components move the cursor to the component line and 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. Click OK
6. 'Are you Sure?', Click OK.
7. A summary installation panel is displayed.

8. Set the Notes_ExecDirectory environment variable to specify the Notes executable directory and ensure that the MQEI_INI_PATH and MQEI_XLAT_PATH environment variables are correctly set.

   For example, using the Korn shell:

```
Notes_ExecDirectory=/opt/lotus/notes/latest/ibmpow;export
Notes_ExecDirectory
MQEI_INI_PATH=/usr/lpp/mqm/mqei;export MQEI_INI_PATH
MQEI_XLAT_PATH=/usr/lpp/mqm/mqei/conv;export
MQEI_XLAT_PATH
```

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
```

2. Follow steps 2-8 as if you were running AIX 4.1.n.

The MQEI components are now installed on your system in:

- /usr/lpp/mqm/mqei
  The files in this directory are:
  readme - A ReadMe file. Text that MUST be read before using the MQEI.
  mqei.ini - The MQEI initialization file. This is an example that you can use to run the samples.
  All other files are needed by the uninstall option.
- /usr/lpp/mqm/mqei/bin
  A directory containing the MQEI executables.
- /usr/lpp/mqm/mqei/conv
  A directory containing the files needed to support character conversion.
- /usr/lpp/mqm/mqei/database
  A directory containing the MQEI Definition database, MQEI Security database, and a design template for each.
- /usr/lpp/mqm/mqei/docs
  A directory containing the MQEI User Guide in Portable Document Format (mqeihelp.pdf) and as a Notes database (mqeihelp.nsf).
- /usr/lpp/mqm/mqei/lib
  A directory containing the AIX version of the MQEI including MQSeries library stubs.

- /usr/lpp/mqm/mqei/samples
  A directory containing components needed to run the MQEI samples. This directory contains the MQEI Samples database (mqeisamp.nsf).

Installing on HP-UX

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

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/mqei.fpkg
   ```

substituting <cd_mount_point> with the name you mounted your CD-ROM device as or the directory in which the MQEI package is available to you.

3. From the Software Selection panel displayed:
   - Press the spacebar to highlight the line for the complete package or press enter to expand the package into its components, and highlight the ones you want to install.
   - 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. Click OK to exit from the log file window.
6. Press OK to continue install.
7. Press Yes to start install.
   The installation process starts and you can follow the progress from the information given in the window.
8. When status is completed, check there are no errors or warnings listed in the log file.
9. Click OK to exit from the log file window.
10. Click Done to exit from the install window.
11. Select File ..... Exit from the menu bar.
12. Set the Notes_ExecDirectory environment variable to specify the Notes executable directory and ensure that the MQEI_INI_PATH and MQEI_XLAT_PATH environment variables are correctly set.

    For example, using the Korn shell:

    ```
    Notes_ExecDirectory=/opt/lotus/notes/latest/hppa;export
    Notes_ExecDirectory
    MQEI_INI_PATH=/opt/mqm/mqei;export MQEI_INI_PATH
    MQEI_XLAT_PATH=/opt/mqm/mqei/conv;export MQEI_XLAT_PATH
    ```

13. Use the MQEI_MQ_LIB environment variable to enable libeilsx.sl to find either eilsxmqm or eilsxmqic.

    For example, using the Korn shell:

    ```
    MQEI_MQ_LIB=/opt/mqm/mqei/lib/eilsxmqm;export MQEI_MQ_LIB
    ```

The MQEI components are now installed on your system in:

- /opt/mqm/mqei
  The files in this directory are:
  readme - A ReadMe file. Text that MUST be read before using the MQEI.
  mqei.ini - The MQEI initialization file. This is an example that you can use to run the samples.
  All other files are needed by the uninstall option.
- /opt/mqm/mqei/bin
  A directory containing the MQEI executables.
- /opt/mqm/mqei/conv
  A directory containing the files needed to support character conversion.
- /opt/mqm/mqei/database
  A directory containing the MQEI Definition database, MQEI Security database, and a design template for each.
- /opt/mqm/mqei/docs
  A directory containing the MQEI User Guide in Portable Document Format (mqeihelp.pdf) and as a Notes database (mqeihelp.nsf).
- /opt/mqm/mqei/lib
  A directory containing the HP-UX version of the MQEI including MQSeries library stubs.
- /opt/mqm/mqei/samples
  A directory containing components needed to run the MQEI samples. This directory contains the MQEI Samples database (mqeisamp.nsf).

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:
   - Change to the drive from which you want to install the MQEI
   - Change to the \os2\mqei\en_us directory on the CD or server drive
   - At the command prompt, type INSTALL and press the enter key.
2. The MQEI Welcome window 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 with the necessary updates to the LIBPATH and PATH statement and setting of all the environment variables. 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 allows you to specify the drive and directory into which the components are to be installed. The default is c:\mqm\mqei. 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. The install program exits and returns to an OS/2 command prompt.

7. When the installation is complete, the Installation and Maintenance window is displayed. Select OK. The install program exits and returns to an OS/2 command prompt.

When installation is complete, a folder called MQSeries Enterprise Integrator is created on your OS/2 desktop. This folder contains:

- Enterprise Integrator Installation and Maintenance
- mqeihelp.pdf
- readme.txt

8. You should now shutdown and reboot your system.

The MQEI 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\mqei
  The files in this directory are:
  readme.txt - A ReadMe file. Text that MUST be read before using the MQEI.
  mqei.ini - The MQEI initialization file. This is an example that you can use to run the MQEI samples.
  All other files are needed by the uninstall option.
- mqm\mqei\bin
  A directory containing the MQEI executables and the OS/2 version of the MQEI.
- mqm\mqei\conv
  A directory containing the files needed to support character conversion.
- mqm\mqei\database
  A directory containing the MQEI Definition database and the MQEI Security database, and design templates for each of them.
- mqm\mqei\docs
  A directory containing the MQEI User Guide in Portable Document Format (mqeihelp.pdf) and as a Notes database (mqeihelp.nsf).
- mqm\mqei\samples
  A directory containing components needed to run the MQEI samples. This directory contains the MQEI Samples database (mqeisamp.nsf).

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 MQEI type:

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

substituting <cd_mount_point> with cdrom/domino_mq or the directory in which the MQEI 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 <mqei>
   [y,n?]
   ```

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

7. Set the Notes_ExecDirectory environment variable to specify the Notes executable directory and ensure that the MQEI_INI_PATH and MQEI_XLAT_PATH environment variables are correctly set.

   For example, using the Korn shell:

   ```
   Notes_ExecDirectory=/opt/lotus/notes/latest/sunspa ;export
   Notes_ExecDirectory
   ```

```
MQEI_INI_PATH=/opt/mqm/mqei;export MQEI_INI_PATH
MQEI_XLAT_PATH=/opt/mqm/mqei/conv;export MQEI_XLAT_PATH
```

The MQEI components are now installed on your system in:

- /opt/mqm/mqei
  The files in this directory are:
  readme - A ReadMe file. Text that MUST be read before using the MQEI.
  mqei.ini - The MQEI initialization file. This is an example that you can use to run the samples.
  All other files are needed by the uninstall option.
- /opt/mqm/mqei/bin
  A directory containing the MQEI executables.
- /opt/mqm/mqei/conv
  A directory containing the files needed to support character conversion.
- /opt/mqm/mqei/database
  A directory containing the MQEI Definition database, MQEI Security database, and a design template for each.
- /opt/mqm/mqei/docs
  A directory containing the MQEI User Guide in Portable Document Format (mqeihelp.pdf) and as Notes database (mqeihelp.nsf).
- /opt/mqm/mqei/lib
  A directory containing the Sun Solaris version of the MQEI including MQSeries library stubs.
- /opt/mqm/mqei/samples
  A directory containing components needed to run the MQEI samples. This directory contains the MQEI Samples database (mqeisamp.nsf).

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 MQSeries Enterprise Integrator for Lotus Notes.

Windows NT version 4 and Windows 95

From your Windows desktop:

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

1. The MQEI 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 Enterprise Integrator. Choose a name for the Program Group folder you want to add the MQEI 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
   Click OK when you have made your selection.
10. If you are using Windows 95, your AUTOEXEC.BAT file is updated with the following statements:
    - SET PATH="%PATH%;C:\MQM\MQEI\BIN"
    - SET MQEI_XLAT_PATH=C:\MQM\MQEI\CONV
    - SET MQEI_INI_PATH=C:\MQM\MQEI

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

11. The setup program will automatically add icons to the program group you specified earlier in the the installation process. These are:

- Help File
- Readme
- Uninstall MQSeries Enterprise Integrator

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

The MQEI 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\mqei
  The files in this directory are:
  readme.txt - A ReadMe file. Text that MUST be read before using the MQEI.
  mqei.ini - The MQEI initialization file. This is an example that you can use to run the samples.
  All other files are needed by the uninstall option.
- mqm\mqei\bin
  A directory containing the MQEI executables and the Windows 32 bit version of the MQEI.
- mqm\mqei\conv
  A directory containing the files needed to support character conversion.
- mqm\mqei\database
  A directory containing the MQEI Definition database, MQEI Security database, and a design template for each.
- mqm\mqei\docs
  A directory containing the MQEI User Guide in Portable Document Format (mqeihelp.pdf) and as a Notes database (mqeihelp.nsf).
- mqm\mqei\samples
  A directory containing components needed to run the MQEI samples. This directory contains the MQEI Samples database (mqeisamp.nsf).

If you chose to install the MQEI components in to a directory other than the default directory (mqm\mqei), substitute mqm\mqei with the location you specified earlier in the installation process.

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.

Tip You are recommended to exit any other Windows applications that you may have running before you start to install MQSeries Enterprise Integrator for Lotus Notes.

1. From Windows Program Manager:
   - Select File ..... Run
   - Type drive:\win16\mqei\setup
     substituting drive with the name of the drive from which you want to install the MQEI
   - Press enter
2. The MQEI Welcome window 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 The uninstall option removes files from the directories into which the install process places them.
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 window is displayed. The default name is MQSeries Enterprise Integrator for Lotus Notes. Choose a name and select the Next button to continue.
6. Start Copying Files window 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 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 install.
8. 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.

9. Installation is now complete. If you have chosen to view the ReadMe file, the Notepad application runs to display the file.
10. The folder on your desktop contains:
    - Uninstall MQSeries Enterprise Integrator
    - Readme
    - Documentation
11. Reboot your system.
    Your AUTOEXEC.BAT file is updated with the following statements:
    - SET PATH=%PATH%;C:\MQM\MQEI\BIN
    - SET MQEI_XLAT_PATH=C:\MQM\MQEI\CONV
    - SET MQEI_INI_PATH=C:\MQM\MQEI

The MQEI 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\mqei
  The files in this directory are:
  readme.txt - A ReadMe file. Text that MUST be read before using the MQEI.
  mqei.ini - The MQEI initialization file. This is an example that you can use to run the samples.
  All other files are needed by the uninstall option.
- mqm\mqei\bin
  A directory containing the MQEI executables and the Windows 16 bit version of the MQEI.
- mqm\mqei\conv
  A directory containing the files needed to support character conversion.
- mqm\mqei\database
  A directory containing the MQEI Definition database and the MQEI Security database.
- mqm\mqei\docs
  A directory containing the MQEI User Guide in Portable Document Format (mqeihelp.pdf) and as a Notes database (mqeihelp.nsf).
- mqm\mqei\samples
  A directory containing components needed to run the MQEI samples. This directory contains the MQEI Samples database (mqeisamp.nsf).

If you are installing MQEI for the first time

Copy the MQEI User Guide database (mqeihelp.nsf) in the docs directory to your Notes/data directory.

Copy the MQEI Samples database (mqeisamp.nsf) in the samples directory, and the MQEI Definition and MQEI Security databases and templates (mqeidata.nsf, mqeidata.ntf, mqeisecu.nsf, mqeisecu.ntf) in the database directory, to your Notes data directory. These three databases together contain the code and MQEI definitions that comprise the MQEI samples.

When copying Notes databases on AIX, HP-UX or Sun Solaris, ensure that appropriate write permission bits are preserved in the new database copies.

Copy the mqei.ini file in the mqei directory to a different directory (your Notes/data directory for example), and update the environment variable MQEI_INI_PATH to reflect the new directory.

For more information about environment variables, see "MQEI Environment variables" earlier in this chapter.

After restarting your system, you can run the MQEI samples .

See the Appendices for information about the samples.

When you start to create your own MQEI applications and definitions, you are recommended to create a new MQEI Definition database and a new MQEI Security database with new names. You can do this either by copying the mqeidata.nsf and mqeisecu.nsf files, or by selecting File-Database-New and specifying mqeidata.ntf or mqeisecu.ntf (as appropriate) as the database template. Creating new databases in this manner makes it much easier for you to apply IBM-supplied maintenance to the MQEI samples and your own MQEI applications.

Your mqei.ini file requires updating to reflect the names of your new databases.

For more information about the initialization file, see "Setting up your MQEI initialization file" later in this chapter.

If you are updating your MQEI installation

Copy the MQEI User Guide database (mqeihelp.nsf) in the docs directory to your Notes data directory.

If you have modified the code or MQEI definitions that comprise the MQEI samples (mqeidata.nsf, mqeisecu.nsf, mqeisamp.nsf), first backup these files. Then copy the MQEI Samples database (mqeisamp.nsf) in the samples directory, and the MQEI Definition and MQEI Security databases and templates (mqeidata.nsf, mqeidata.ntf, mqeisecu.nsf, mqeisecu.ntf) in the database directory, to your Notes data directory. This upgrades the MQEI samples to the level of MQEI you have just installed. This will overwrite the existing files with those names.

To upgrade the definitions in your own MQEI Definition and MQEI Security databases used by your MQEI applications, ensure the design is replaced by selecting File-Database-Replace Design and specifying mqeidata.ntf or mqeisecu.ntf (as appropriate) as the database template. You will also have to rebuild your MQEI Message definitions in your MQEI Definition database (see Chapter 3). This will bring your databases up to the level of MQEI you have just installed. Failure to do this may result in errors when you next run your MQEI applications.

Your mqei.ini file may require modification, but only if the update has changed the format of the mqei.ini file. If this is the case, the MQEI ReadMe file supplied with the update will indicate this.

Setting up your MQEI initialization file

This is a formatted plain text file, named mqei.ini, that is read by the EISession object and by utility programs such as the BMS map conversion utility. It contains environment specific information, such as:

- Basic system information including the local character set and encoding
- The name and location of the MQEI Definition database
- The name and location of the MQEI Security database

You can also include comments in your mqei.ini file. If you start a line with the semi-colon (;) character, it is treated as a comment. The legal values for all the keywords are included as comments in your sample mqei.ini file that is provided as part of the MQEI package.

You must set the MQEI_INI_PATH environment variable to point to your mqei.ini file. If you do not set the MQEI_INI_PATH environment variable, you get the reason code EIRC_INI_OPEN_ERROR.

You will need to edit the MQEI initialization file before you can use it. For more information, see "Editing the initialization file" later in this chapter.

Note There must be one mqei.ini file on every client and server that you run MQEI LSX code and utility programs on.

Editing the initialization file

Before you can use the MQEI initialization file, you will need to edit it depending on the attributes of your MQEI LSX system and where your MQEI databases are stored. You can edit the mqei.ini file using a standard text editor.

The mqei.ini file includes the following sections and keywords:

- Base (signified by the [Base] stanza)
  - CharacterSet
  - Encoding
- MQEI Definition database (signified by the [DefinitionDatabase] stanza)
  - DatabaseType
  - DatabaseName
  - ServerName
- MQEI Security database (signified by the [SecurityDatabase] stanza)
  - DatabaseType
  - DatabaseName
  - ServerName

Note
If the EISession class cannot find a *section* (Base, DefinitionDatabase or SecurityDatabase) within the mqei.ini file, it returns a reason code of EIRC_INI_SECTION_NOT_FOUND.
If the EISession class cannot find a *keyword* (CharacterSet, Encoding, DatabaseType, DatabaseName or ServerName) within a *section* of the mqei.ini file, it returns a reason code of EIRC_INI_KEYWORD_NOT_FOUND.
If the EISession class cannot find a *value* associated with a *keyword*, it returns a reason code of EIRC_INI_VALUE_NOT_SPECIFIED.

[Base]

This section must be present in the mqei.ini file. It describes certain basic attributes of the MQEI LSX system, and contains the following keywords.

CharacterSet

The character set of the local machine, expressed as a code page number, or the word 'Local'. If you set the CharacterSet to 'Local', the MQEI LSX determines the local character set automatically. It is mandatory.

You are recommended **not** to set this to a numeric value unless you need to override the local character set. However, you may need to do this if you are running Windows NT or Windows 95.

For more information, see "Data Conversion" in Chapter 11.

Legal Values :

- Local

  ```
  CharacterSet = Local
  ```

- A code page number

  ```
  CharacterSet = nnnn
  ```

Note If you specify an invalid value for CharacterSet, the EISession class returns EIRC_INVALID_CHARACTER_SET.

Encoding

The encoding of integer data on the local machine, expressed as the name of an operating system, or the word 'Local'. If you set Encoding to 'Local', the MQEI LSX determines the local encoding automatically. It is mandatory.

You are recommended **not** to set this to a specific value unless you need to override the local encoding.

Legal Values :

- Local

  ```
  Encoding = Local
  ```

- AIX

  ```
  Encoding = AIX
  ```

- HP-UX

  ```
  Encoding = HP-UX
  ```

- OS/2

  ```
  Encoding = OS/2
  ```

- Windows 3.1

  ```
  Encoding = Windows 3.1
  ```

- Windows 95

  `Encoding = Windows 95`
- Windows NT

  `Encoding = Windows NT`
- Sun OS

  `Encoding = Sun OS`
- Sun Solaris

  `Encoding = Sun Solaris`

Note If you specify an invalid value for Encoding, the EISession class returns the reason code, EIRC_INVALID_ENCODING.

[DefinitionDatabase]

This section must be present in the mqei.ini file. It describes the attributes of the MQEI Definition database to be used by the MQEI LSX and utility programs, and contains the following keywords.

DatabaseType

The type of the MQEI Definition database. It is mandatory.

Legal Values :

- LotusNotes

```
DatabaseType = LotusNotes
```

Note If you enter an invalid DatabaseType the EISession class returns the reason code, EIRC_INVALID_DB_TYPE.

DatabaseName

The location of the MQEI Definition database. It is mandatory.

For example:

```
DatabaseName = E:\NOTES\DATA\MQEIDATA.NSF (for Intel
platforms)
or

DatabaseName = /home/userid/notesr4/mqeidata.nsf (for UNIX
systems)
```

ServerName

The name of the Lotus Notes server upon which the MQEI Definition database resides. It is mandatory. Specify the keyword "Local" if the database resides on your local workstation otherwise the full server name must be specified.

For example:

```
ServerName = Local
```

[SecurityDatabase]

This section must be present in the mqei.ini file. It describes the attributes of the MQEI Security database to be used by the MQEI LSX and utility programs, and contains the following keywords.

DatabaseType

The type of the MQEI Security database. It is mandatory.

Legal Values :

- LotusNotes

  ```
  DatabaseType = LotusNotes
  ```

- None

  ```
  DatabaseType = None
  ```

Note If you enter an invalid DatabaseType the EISession class returns the reason code, EIRC_INVALID_DB_TYPE.

DatabaseName

The location of the MQEI Security database. It is mandatory if DatabaseType is "LotusNotes", otherwise it may be omitted.

For example:

```
DatabaseName = E:\NOTES\DATA\MQEISECU.NSF (for Intel
platforms)
or
DatabaseName = /home/userid/notesr4/mqeisecu.nsf (for UNIX
systems)
```

ServerName

The name of the Lotus Notes server upon which the MQEI Security database resides. It is mandatory if DatabaseType is "LotusNotes", otherwise it may be omitted. Specify the keyword 'local' if the database resides on your local workstation otherwise the full server name must be specified.

For example:

```
ServerName = Local
```

Example mqei.ini files

The mqei.ini file that is provided as part of the MQEI package contains the legal values for all of the keywords as comments to help you get started quicker. You can delete these if you prefer and just have a plain mqei.ini file with no comments.

```
;************************************************************
;                   Sample mqei.ini file
;************************************************************
[Base]
Encoding     = Local
;------------------------------------------------------------
; Legal Values:
;     Encoding     = Local                 <==== RECOMMENDED
;     Encoding     = AIX
;     Encoding     = HP-UX
;     Encoding     = OS/2
;     Encoding     = Sun OS
;     Encoding     = Sun Solaris
;     Encoding     = Windows 3.1
;     Encoding     = Windows 95
;     Encoding     = Windows NT
;------------------------------------------------------------
CharacterSet = Local
;------------------------------------------------------------
; Legal Values:
;     CharacterSet = Valid code page number
;     CharacterSet = Local                 <==== RECOMMENDED
;------------------------------------------------------------

;************************************************************
; DefinitionDatabase section...
;************************************************************
```

```
[DefinitionDatabase]
;------------------------------------------------------------
; Name of your MQEI Definition database
;------------------------------------------------------------
DatabaseName = mqeidata.nsf
;------------------------------------------------------------
; Type of your Definition Database (MUST BE: LotusNotes)
;------------------------------------------------------------
DatabaseType = LotusNotes
;------------------------------------------------------------
; Name of the Lotus Domino Server where your MQEI Definition
; database resides
;------------------------------------------------------------
ServerName   = Local
;------------------------------------------------------------
; Legal Values:
;     ServerName   = Local
;     ServerName   = Your Lotus Domino Server name
;------------------------------------------------------------

;************************************************************
; SecurityDatabase section...
;************************************************************
[SecurityDatabase]
;------------------------------------------------------------
; Name of your MQEI Security database
;------------------------------------------------------------
DatabaseName = mqeisecu.nsf
;------------------------------------------------------------
; Type of your Security Database
;------------------------------------------------------------
```

```
DatabaseType = LotusNotes
;------------------------------------------------------------
; Legal Values:
;     DatabaseType = LotusNotes
;     DatabaseType = None
;------------------------------------------------------------
;------------------------------------------------------------
; Name of the Lotus Domino Server where your
;      MQEI Security database resides
;------------------------------------------------------------
ServerName   = Local
;------------------------------------------------------------
; Legal Values:
;     ServerName   = Local
;     ServerName   = Your Lotus Domino Server name
;------------------------------------------------------------
```

The following is an example of a typical mqei.ini file if you are **not** using the MQEI Security database and the MQEI Definition database resides on a Domino server.

```
[Base]
  CharacterSet = Local
  Encoding = Local

[DefinitionDatabase]
  DatabaseType = LotusNotes
  DatabaseName = home/user1/notesr45/mqeidata.nsf
  ServerName = ABC Server1/XYZ

[SecurityDatabase]
  DatabaseType = None
```

The following is an example of a typical mqei.ini file if you **are** using the MQEI Security database and both databases reside on your own local workstation.

```
[Base]
  CharacterSet = Local
  Encoding = Local

[DefinitionDatabase]
  DatabaseType = LotusNotes
  DatabaseName = c:\path\mqeidata.nsf
  ServerName = Local

[SecurityDatabase]
  DatabaseType = LotusNotes
  DatabaseName = c:\path\mqeisecu.nsf
  ServerName = Local
```

Post-Installation Check program

To check you have installed the MQEI LSX correctly, you should run the Post-Installation Check program.

This program is a LotusScript agent that you run from the Actions menu when the MQEI Samples database (mqeisamp.nsf) is selected on your workspace.

The program checks that:

- You have an MQEI initialization file (mqei.ini) on your system that it can access.
- All keywords are present in the mqei.ini file.
- The supplied MQEI Definition database is located in the place indicated in the mqei.ini file.
- The supplied MQEI Security database, if used, is located in the place indicated in the mqei.ini file.
- All the definitions needed by the samples are present in the relevant database.

Running the Post-Installation Check program

To run this you must:

- Set the environment variable, MQEI_INI_PATH, to point to your mqei.ini file, or make sure your mqei.ini file is in the current working directory when you run Notes.
  Use the platform specific commands to set the environment variable:

To set the environment variable on OS/2, WIN-OS/2, Windows 3.1, Windows 95 and Windows NT:

Command	Effect
SET MQEI_INI_PATH=drive:\directory	Sets the directory where the mqei.ini file is stored
SET MQEI_INI_PATH=	Removes the MQEI_INI_PATH environment variable
SET MQEI_INI_PATH	Displays the current setting of the mqei.ini file directory path on OS/2, Windows for WorkGroups, and Windows 3.1
ECHO %MQEI_INI_PATH%	Displays the current setting of the mqei.ini file directory path
SET	Displays the contents of all the environment variables on OS/2, Windows 3.1, Windows for WorkGroups, and Windows NT

To set the environment variable on AIX, HP-UX and Sun Solaris:

Command	Effect
export MQEI_INI_PATH=/directory	Sets the directory where the mqei.ini file is stored
unset MQEI_INI_PATH	Removes the MQEI_INI_PATH environment variable
echo $MQEI_INI_PATH	Displays the current setting of the mqei.ini file directory path
set	Displays all the settings for all the environment variables for the session

- Add the MQEI Samples database (mqeisamp.nsf) to your workspace.
- Customize your mqei.ini file to your local environment.

For more information, see "Setting up your MQEI initialization file" earlier in this chapter.

Having completed these steps:

1. Select the MQEI Samples database so that it is highlighted - do **not** open it
2. Choose Actions / Run MQEI Post-Installation Check from the Notes menu bar
3. If the program is successful, the message "MQEI Post-Installation Check completed successfully" is displayed.

If the program finds a problem, a message is displayed and the check terminates. If there are definitions missing for samples that you do not want to run, and everything else is OK, you can continue.

If you need the definitions or if some error has been identified, correct the error and rerun the program.

If the eilsx library has not been loaded, the message

```
Error loading USE or USELSX module: eilsx
```

is displayed. If this occurs, check the installation documentation for details of where it should be and put the eilsx library in the correct location before you run it again.

Using the MQEI for the first time

Having successfully installed the MQEI LSX, the next step is to start using it. The possibilities are:

- You have an existing enterprise application for which you want to develop a Notes interface.
- You are developing a new application that uses both Notes and an enterprise system, e.g. CICS.

In both cases, the same principles apply when using the MQEI LSX. You:

1. Create definitions in your own MQEI Definition database that describe the:
   - Enterprise application you wish to access (MQEI Service definitions).
     To do this you need to know the properties of the enterprise application, such as transaction name or program name, MQSeries queue manager name or CICS system name, and any MQSeries queue names.
   - Messages that you want to send to the enterprise application (MQEI Message definitions).
     To do this you need to know the formats of the data to be sent to the application, such as the format of MQSeries messages, CICS COMMAREAs or CICS BMS maps.
     MQSeries message and CICS COMMAREAs are most likely to be defined by data structures in the COBOL, C, PL/I or other language copybooks used to build the application.
     CICS BMS map definitions are most likely to be in source files containing DFHMSD, DFHMDI, and DFHMDF macros.
   - Messages that you want to receive from the enterprise application (MQEI Message definitions).
     To do this you need to know the formats of the data to be received from the application, such as the format of MQSeries messages, CICS COMMAREAs or CICS BMS maps.
     MQSeries message and CICS COMMAREAs are most likely to be defined by data structures in the COBOL, C, PL/I or other language copybooks used to build the application.
     CICS BMS map definitions are most likely to be in source files containing DFHMSD, DFHMDI, and DFHMDF macros.
2. Create a Notes application that includes LotusScript programs that use the MQEI LSX, making use of the definitions you have created in the MQEI Definition database.
3. Optionally create definitions in your own MQEI Security database that describe the security details for each user of the application.

This requires each user of the Notes application to have an MQEI Security definition that contains their enterprise system userid and authenticator. As an alternative, you can prompt the user for this information, where practical, in your Notes application, in which case no MQEI Security definitions are required.

For more information on Security, see Chapter 5.

Where to go next...

For more information about how to create MQEI Service and MQEI Message definitions, see "The MQEI Definition database" in Chapter 3.

For more information about how to create MQEI Security definitions, see "The MQEI Security database" in Chapter 3. For more information on "Security" in general, see Chapter 5.

For information on "Design and Programming using the MQEI", see Chapter 4. For programming information specific to each particular type of enterprise service, see Chapters 6-10.

Tip A good starting point is to examine the sample applications in the MQEI Samples database. These are explained in Appendices A - F.

For futher samples, information, and help, see the IBM Red book:

Lotus Notes and the MQSeries Enterprise Integrator, available at URL:

http://www.redbooks.ibm.com/SG242217/sg242217.html

or order book no: SG24-2217

Chapter 3 Using the MQEI databases

This chapter covers:

- The roles and key features of the MQEI databases.
- Instructions on how to maintain the MQEI database information.
- Errors that can arise and what to do about them.

General

This section provides you with some general information about the MQEI databases.

It includes:

- About the MQEI databases
- Location of the MQEI databases
- Applying service upgrades to the MQEI databases
- About MQEI definitions
- Using the Action Bar

About the MQEI databases

The MQEI databases are Lotus Notes databases that contain definitions of resources that are used by the MQEI LSX at run-time. There are two of them:

- The **MQEI Definition database** contains definitions of enterprise services and enterprise message formats, and is mandatory.
- The **MQEI Security database** contains definitions of your users' enterprise security parameters, and is optional.

Each definition is stored in the database as a Notes document. Both databases are shipped as .NSF files. Design templates for both databases are also shipped as .NTF files.

The MQEI Definition database is shipped as mqeidata.nsf and the MQEI Security database is shipped as mqeisecu.nsf. You do not have to keep these names. You can call the MQEI databases whatever you like.

Both databases are explained in more detail in subsequent sections. The following sections contain general information that is applicable to both databases.

Help within the MQEI databases

You can get context sensitive help on any of the user input fields in both of the MQEI databases. Click on the descriptive text to the left of the user input field and a small box will appear telling you:

- What the field is about
- The legal values you can enter in the current field
- One or more examples of the values you can enter in the current field

Location of the MQEI databases

You are recommended to locate the MQEI databases on Domino servers, and to use Notes Access Control facilities to control which users have access to them. This approach makes administration of the databases simpler.

You can store more than one copy of the MQEI databases on your Domino server though you can only specify the location of one of them in your mqei.ini file.

If your users are accessing the MQEI databases from Web browsers, the MQEI databases must be located on a Domino server.

Applying service upgrades to the MQEI databases

When applying service upgrades to the databases, IBM will ship updated design template .NTF files. You should use these to replace the design of your copies of the databases by selecting File-Database-Replace Design from the Notes menu bar.

This will only replace the design of your database - not the documents stored within it. This means that you will retain any existing MQEI Definitions when you apply service upgrades to your MQEI databases.

Additionally, certain documents need to be "rebuilt" after service is applied.

For more information, see "Building MQEI Message definitions" later in this chapter.

About MQEI definitions

Related definition documents are grouped together in views. A navigator is provided so you can select the view you want to work with.

A view will be one of either:

- Messages & Fields
- Field Types
- Services
- Categories
- Security definitions

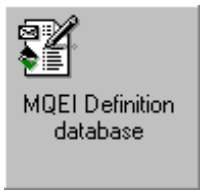
Once in a view, you may create, copy, view, change, delete and categorize definition documents using buttons contained in the action bar and the dialog list in the view. The action bar is the area directly below the toolbar. When working with a definition document, the action bar contains buttons that allow you to perform actions specific to that document type, such as edit, save, copy, re-position and close.

Using the Action Bar

Caution: Although it is possible to use keyboard short-cuts and standard pull-downs to perform actions such as deleting, copying and pasting definition documents, you are strongly recommended to use the buttons provided by the action bar in preference. Extra processing takes place when a button is pressed, which ensures that the integrity of the database is maintained. Certain definition documents contain references to related definitions, and this information is automatically maintained when a button is pressed but not when a keyboard short-cut or standard pull-down is used.

The one occasion when the use of keyboard short-cuts and standard pull-downs is recommended is when copying and pasting definition documents **between databases**. This may occur, for example, when updating your production master database from a test version. It is important to ensure that the definition documents being pasted do **not** exist on the target database. If they do, you should delete them first using the appropriate action bar button.

The MQEI Definition database



About the MQEI Definition database

The MQEI Definition database is a Lotus Notes database that contains definitions of the enterprise services with which you want to communicate, and the enterprise messages you want to send and receive. When you create an EIService or EIMessage object in your LotusScript program, the MQEI LSX reads the corresponding MQEI Message or MQEI Service definition from the MQEI Definition database.

This hides a lot of the complexity of communicating with the service and building and interpreting messages from the LotusScript program.

Definitions are stored as documents in the MQEI Definition database. You can create, view, modify, copy and delete definitions.

There are four object definitions that you can work with in this database:

- MQEI Message
- Field Type
- Field
- MQEI Service

Setting up the MQEI Definition database

The MQEI Definition database should reside on your Domino server. If your MQEI application runs under a Notes client, each user of the MQEI application should be given Reader access to the MQEI Definition database.

If your MQEI application runs under an agent on the Domino server, the agent owner (the person who last saved the agent) should be given Reader access.

You should have Author access or higher in order to create the definitions your MQEI applications need.

If you are developing an MQEI application which is to be deployed on several Domino servers across your organization, you are recommended to maintain a master version of the MQEI Definition database and distribute it using Notes replication. When setting up replication, ensure that deletions are replicated as failure to do this may result in duplicate definition documents which may cause an MQEI LSX error at run-time, or may cause the MQEI LSX to use obsolete definitions.

Note Deletions are replicated by default.

Building definition documents

MQEI Message definition documents need to be 'built' before they can be used by the MQEI LSX. This enables the MQEI LSX to run much quicker at run-time. You must rebuild whenever they have been modified. You can tell if an MQEI Message definition has been modified since it was last built by the icon next to the MQEI Field definition within the MQEI Message definition. If there is a red cross next to the MQEI Message definition field, you have made changes since the MQEI Message definition was last built and need to rebuild the MQEI Message definition. These are marked with a red cross. Failure to do this may cause an MQEI LSX error at run-time, or may cause the MQEI LSX to use obsolete definitions.

You should only Build MQEI Message definitions on the master version of the database. You can then distribute the built MQEI Message definitions using Notes replication as described above.

Caution If you do not do this, you may experience replication conflicts.

For more information, see "Building MQEI Message definition" later in this chapter.

MQEI Message definition

An MQEI Message definition defines the structure of an EIMessage that will be sent to, or received from, an enterprise service. An EIMessage object is created from an MQEI Message definition by the MQEI LSX. Note that an MQEI Message definition only contains application data. For example, it may represent a CICS DPL program COMMAREA, a CICS 3270 screen or the user portion of an MQSeries message.

Each MQEI Message definition acts as a container for the Fields that make-up the message, each of which is defined by a Field definition.

It comprises:

- Message Name
- Message Description
- Message Format

Message Name

Editable text field that defines the name of the MQEI Message definition. This is the Message Name that is quoted in your LotusScript application.

At the MQEI API, Message Name is accessible via the Name property of the EIMessage Class.

Message Description

Editable text field that describes what the MQEI Message definition contains. It is there for your information so that it is easier for you to remember what the MQEI Message definition contains.

Message Format

Editable text field that describes the format of the data within the message. Message Format is the name of the format that may be used by the enterprise service.

At the MQEI API, Message Format is accessible via the Format property of the EIMessage Class.

Key points:

- A new MQEI Message definition only requires a Message name initially.
- The other fields in the MQEI Message definition form are optional.
- The name of an MQEI Message definition must be unique.
- The names of MQEI Message definitions are case sensitive so you cannot have an MQEI Message definition with a message name of MQMsg and an MQEI Message definition and a message name of mqmsg.
- The name of an MQEI Message definition must have no more than 16 characters.

Building MQEI Message definitions

MQEI Message definitions need to be built for performance reasons. Before MQEI Messages are built, each Field definition is contained in a separate Notes document. After MQEI Message definitions are built, all the Field definitions are contained in one Notes document.

For more information, see "Building MQEI Message definitions" later in this chapter.

Field Type definition

A Field Type definition acts as a template for one or more Field definitions to save re-entering common information about Fields. Field Types are not used by the MQEI LSX API - they are used solely within the MQEI Definition database to define default values for Field definitions. Use Field Types to create default values for your Field definitions.

It comprises:

- Type Name
- Data Type
- String Type
- String Format
- Default Value
- Length

Type Name

Editable text field that defines the name of the user-defined Field Type.

Data Type

Standard keyword field that allows you to define the Data type of the Field Type.

The default value for this field is STRING.

String Type

Standard keyword field that allows you to specify if a field of data type STRING is fixed length or variable length.

The default value for this field is FIXED LENGTH.

At the MQEI API, String Type is accessible via the GetDataType method of the EIMessage class.

String Format

Standard keyword field that allows you to specify the format of a variable length string. The supported formats are:

- String data terminated by a null (or by the end of the message - or by the end of the message segment if IMS)
- String data preceded by two bytes specifying the length of the string data

The default value for this field is NULL TERMINATED.

Default Value
Editable text field that defines a default value for the field.

Length
Editable numeric field that defines the length of the field in bytes.

The default value for this field is 1.

You can define Field Types that you can use in your Field definitions. The advantage of doing this is that you can change the Field Type definition, and the corresponding Field definition values in a number of Field definitions will change automatically without you having to edit each individual Field definition.

For example, if a number of Field definitions have a Type of "PartNumberType" and you want to change the Field default value (or length or data type) in all occurrences, you can do this by editing the Field Type definition.

Key points:
- Use Field Type definitions as templates for Field definitions.
- Use the Field Type definition view to manage and edit your Field Type definitions.
- A new Field Type definition initially requires a Field Type name, a Data Type, and a Length.
- The Default Value is optional.
- The Field Type name of a Field Type definition must be unique.
- The Field Type name must have no more than 32 characters.
- The Length, String Type, and String Format are dependent on the Data Type of the Field Type definition.
- Field Types cannot be deleted whilst "in use" by Field definitions.

Caution Do not delete Field Type definitions using a keyboard short-cut.

Field definition

A Field definition describes a field within a message. A Field definition is owned by exactly one MQEI Message definition. It comprises:

- Message Name
- Field Name
- Description
- Field Type
- Data Type
- String Type
- String Format
- Default Value
- Length
- Segment #
- Has Attributes *
- Row *
- Column *
- Pad Character
- Alignment
- Position

Note Fields marked with # are only used with IMS via MQSeries service messages.
Fields marked with * are only used with CICS 3270 direct service messages.

Message Name

Pre-defined value defining the name of MQEI Message that contains the current Field definition. Fields definitions can only exist within MQEI Message definitions.

At the MQEI API, Message Name is accessible via the Name property of the EIMessage Class.

Field Name

Editable text field defining the name of the individual field within the Message. This is the name by which it is known to your LotusScript application.

At the MQEI API, Field Name is accessible via the GetFieldName method of the EIMessage Class.

Description
Editable text field that allows you to add a description of the field.

Field Type
Editable text or a standard keyword field that allows the current field definition to inherit the properties of an existing user-defined Field Type definition. The properties inherited are:

- Data Type
- String Type
- String Format
- Default Value
- Length

Data Type
Standard keyword field that allows you to define the data type of the field.

The default value for this field is STRING.

At the MQEI API, Data Type is accessible via the GetDataType method of the EIMessage Class.

String Type
Standard keyword field that allows you to specify if a field of data type STRING is fixed length or variable length.

The default value for this field is FIXED LENGTH.

At the MQEI API, String Type is accessible via the GetDataType method of the EIMessage class.

String Format
Standard keyword field that allows you to specify the format of a variable length string. The supported formats are:

- String data terminated by a null (or by the end of the message - or by the end of the message segment if IMS)
- String data preceded by two bytes specifying the length of the string data

The default value for this field is NULL TERMINATED.

Default Value
Editable text field that allows you to define a default value for the field.

Length
Editable text field that defines the length of the field in bytes.

The default value for this field is 1.

At the MQEI API, Length is accessible via the GetLength method of the EIMessage Class.

Segment
Editable text field that indicates the segment number the field resides in.

The default value for this field is 1.

At the MQEI API, Segment is accessible via the GetSegment method of the EIMessage Class.

Has Attributes
Standard keyword field that allows you to specify whether the field has any screen attributes associated with it or not.

The default value for this field is NO.

Row
Editable text field that allows you to specify the row number of the field if the message containing this field is a CICS 3270 screen.

Column
Editable text field that allows you to specify the column number of the field if the message containing this field is a CICS 3270 screen.

Pad Character
Standard keyword field. The pad character field applies only to data types, STRING, AUTHENTICATOR and SYSTEM_AUTHENTICATOR and allows you to specify the character that will be used to pad field values that are shorter than the length property when the message is sent. The only legal values are NULL and SPACE.

The default value for this field is NULL.

Alignment
Standard keyword field that allows the MQEI LSX to align the field on a designated boundary when sending or receiving the enterprise message. It is of use for enterprise messages that have been processed by a compiler that strengthens the alignment boundary of certain data types. For example, forcing a LONG data type onto an 8 byte boundary.

The default value for this field is 1 BYTE.

Position

Editable text field that allows you to specify the position of the field within the message.

Every field in a message has a "location" value associated with it so that a field can be correctly placed in a message. You specify the location by specifying the position of the field and also by:

- Selecting the alignment option when you define the field in the message. You have the choice of alignment on a byte, half-word, word or double-word boundary.
- Selecting the row and column options. This is appropriate when working with data displayed on a screen.

Key points:

- Before you define any Field definitions, consider creating Field Type definitions as templates for your Field definitions. This may save you time later if you have lots of Fields that contain the same values.
- If Field Type is specified, String Type, String Format, Default Value, Length, and Data Type are automatically derived from the Field Type definition.

  Note You can override the Default Value derived from the Field Type definition if you want to change it to a different value.
- A new Field definition initially only requires the following fields to be completed:
  - Message Name
  - Field Name
- All other fields in the Field definition form are optional.
- The Field Name of a Field definition must be unique and have no more than 32 characters.

MQEI Service definition

An MQEI Service definition defines the properties of an enterprise service to which messages will be sent and received. An EIService object is created from an MQEI Service definition by the MQEI LSX.

It comprises:

- Service Name
- Type of Service
- Service Step
- Service Context
- Connection Manager
- Outbound Connection
- Inbound Connection
- Character Set
- Encoding
- System Name

Service Name

Editable text field that defines the name of the MQEI Service definition. This is the name by which it is known in the LotusScript application.

At the MQEI API, Service Name is accessible via the Name property of the EIService class.

Type of Service

Standard keyword field that defines the type of enterprise service. Allowed keywords are:

- CICS 3270 Direct
- CICS DPL Direct
- CICS DPL via MQSeries
- IMS via MQSeries
- Native MQSeries

At the MQEI API, Type of Service is accessible via the ServiceType property of the EIService class.

Service Step

Editable text field that defines the name of the first executable step of the enterprise service as known on the enterprise system. This would normally be a program name or a transaction identifier. This field does not apply to the Native MQSeries service type.

Service Step may be case sensitive depending on the Type of Service you have chosen. For example, the MQSeries DPL bridge is case sensitive.

At the MQEI API, Service Step is accessible via the ServiceStep property of the EIService class.

Service Context

Editable text field that defines the name of any context with which the first executable step of the enterprise service will execute. For example, the CICS transaction identifier you wish a CICS DPL program to run under, or the name of a logical terminal (LTERM) you wish to pass to an IMS transaction.

Service Context may be case sensitive depending on the Type of Service you choose.

At the MQEI API, Service Context is accessible via the ServiceContext property of the EIService class.

Connection Manager

Editable text field that defines the name of the connection manager being used to communicate with the enterprise service. This would normally be an MQSeries queue manager or the name of a CICS server as known by a CICS client.

At the MQEI API, Connection Manager is accessible via the ConnectionManager property of the EIService class.

Note Connection Manager names are case sensitive.

For example:

```
newyork.mqserv3
CICSSYS1
```

Outbound Connection

Editable text field that defines the name of the connection being used for the transmission of outbound messages. For example, this could be the name of an MQSeries queue or a CICS terminal model name.

At the MQEI API, Outbound Connection is accessible via the OutboundConnection property of the EIService class.

Note Outbound Connection names are case sensitive.

For example:

```
PARTS.APPLICATION
```

Inbound Connection

Editable text field that defines the name of the connection being used for the transmission of inbound messages. For example, this could be the name of an MQSeries queue.

At the MQEI API, Inbound Connection is accessible via the InboundConnection property of the EIService class.

Note Inbound Connection names are case sensitive.

For example:

```
PARTS.REPLY
```

Character Set

Editable numeric field that defines the character set of the enterprise system.

At the MQEI API, Character Set is accessible via the CharacterSet property of the EIService class.

Encoding

Editable keyword field that defines the representation of numeric data on the enterprise system.

System Name

Editable text field that defines the name of the enterprise system upon which the enterprise service resides and upon which authentication takes place.

This is used to locate an MQEI Security definition in the MQEI Security database.

At the MQEI API, System Name is accessible via the SystemName property of the EIService class.

Key points:

- The Service definition name must be unique and must not exceed 16 characters.
- The allowed types of services are already defined in the dialog list.
- The presence of other properties and whether they are mandatory depends on the Type of Service chosen.

Tip For more information about what values you should specify in each of the fields when using MQEI Service definitions, see the help on that topic.

Categories view

You can categorize MQEI Message definitions and MQEI Service definitions so that you can organize them in a way that you will be able to find them easily if you have lots of MQEI definitions in your MQEI Definition database.

You may like to think of it as a filing system for all your MQEI Message and Service definitions. You can store them all in a view of their own so that you can find them easily.

For example, if you want to categorize all of the MQEI definitions used by the IMS via MQSeries sample, you can specify the category that you want them to appear under in the definition view for the MQEI Message definition or the MQEI Service definition you want to categorize.

For more information, see "Using MQEI Message definitions" and "Using MQEI Service definitions" later in this chapter.

Using MQEI Message definitions

This section describes how to:

- Create a new MQEI Message definition.
- Copy an MQEI Message definition.
- View an MQEI Message definition.
- Change an MQEI Message definition.
- Delete an existing MQEI Message definition and its Fields.
- Categorize a new or existing MQEI Message definition.
- Build an MQEI Message definition.
- View a built MQEI Message definition.
- Delete a built MQEI Message definition.

Creating a new MQEI Message definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Click "New Message" on the action bar.
4. Specify values in the "Message Name" field.

   Note
   The "Message Name" must not exceed 16 characters and must be unique.
   The "Message Description" field is optional and is only used to describe the content of the MQEI Message definition.
   The "Message Format" field is optional.
5. Click "Save & Close" or "Save & New" on the action bar
   If you choose "Save & New", you are expected to create a new MQEI Message definition. Only choose this option if you want to create further MQEI Message definitions.
6. Click "Build Message(s)" on the action bar to build the MQEI Message definition.

You are now ready to start adding Field definitions to your MQEI Message definition. For more information, see "Creating a new Field definition" later in this chapter.

Copying an MQEI Message definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition document you want to copy by clicking on the twisty.
4. Open the MQEI Message definition document by double clicking it.
5. Click "Copy" on the action bar.
6. Enter a new Message Name in the blank field.
7. Click "OK Copy" on the action bar if you want to copy the MQEI Message definition.

   Note Click "Cancel Copy" on the action bar if you want to cancel the copy operation.

Viewing an MQEI Message definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to view by clicking on the twisty.
4. View the MQEI Message definition by double clicking the MQEI Message definition document.

Changing an MQEI Message definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to change by clicking on the twisty.
4. Select the MQEI Message definition document you want to change.
5. Click "Edit Message / Field" on the action bar.
6. Change the values within the MQEI Message definition.
7. Click "Save & Close" on the action bar.
8. Re-build the MQEI Message definition.

Deleting an MQEI Message definition and its Fields

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to delete by clicking on the twisty.
4. Select the MQEI Message definition document you want to delete.
5. Click "Delete Message(s) / Field(s)" on the action bar.
6. Confirm that you want to delete the selected MQEI Message definition and its associated fields.

Categorizing a new or existing MQEI Message definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to categorize by clicking on the twisty.
4. Select the MQEI Message definition document you want to categorize.
5. Click "Edit Message / Field" on the action bar.
6. Enter a new category name in the Category field or choose an existing category name from the dialog list.

   Note You can select more than one existing category if you wish.
7. Click "Save & Close" or "Save & New" on the action bar.
   If you choose "Save & New" , you are expected to create a new MQEI Message definition. Only choose this option if you want to create a new MQEI Message definition.
8. Re-build the MQEI Message definition.

Building MQEI Message definitions

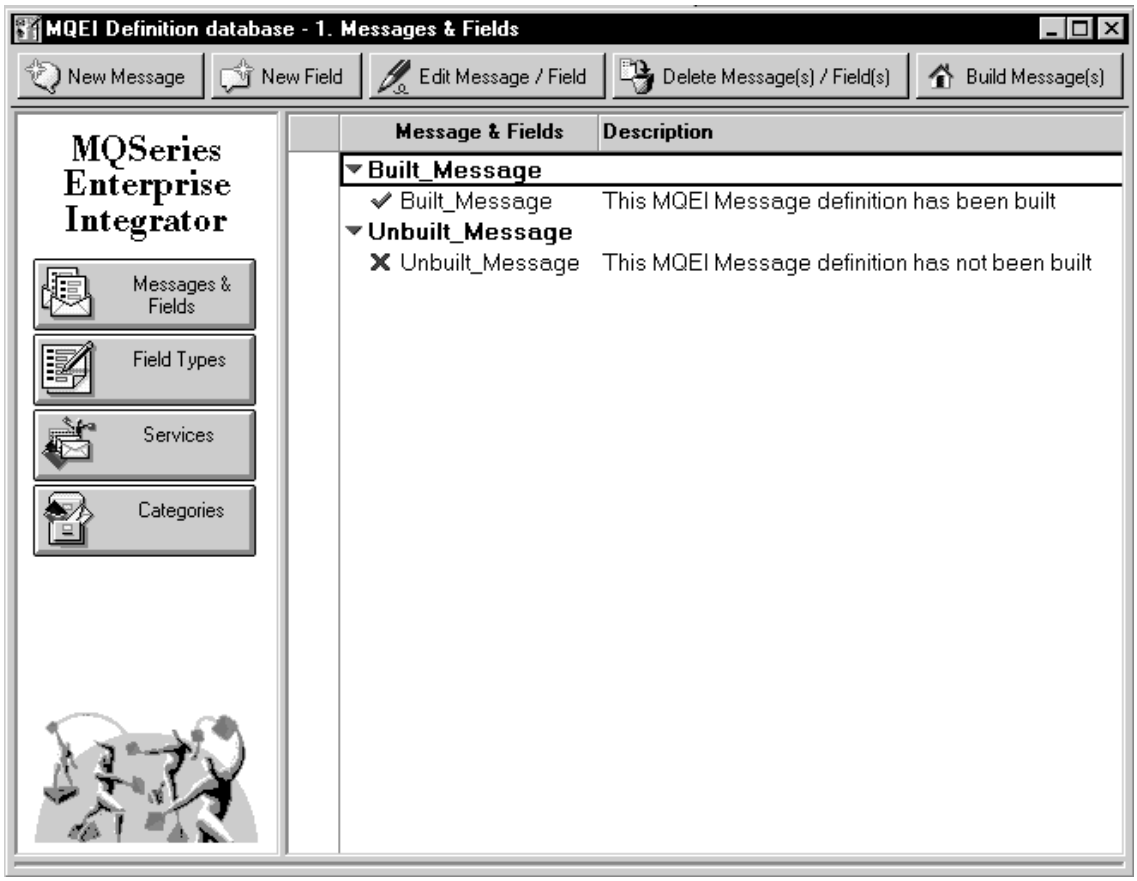
Changed MQEI Message definitions need to built for performance reasons. Until an MQEI Message definition is built, each of the MQEI Message and Field definitions are stored in separate Notes documents. This would make creation of an EIMessage object by the MQEI LSX extremely slow (especially where an MQEI Message definition contains a large number of Field definitions).

Once an MQEI Message definition is built, it is stored in a single document that you, as a user, cannot see.

You can see which fields have not been included in the last build by the icon next to the MQEI Field definition within the MQEI Message definition.

If there is a green tick next to the MQEI Message definition field, it was included in the last build.

If there is a red cross next to the MQEI Message definition field, you have made changes since the MQEI Message definition was last built and you need to rebuild the MQEI Message definition.



1. Expand the MQEI Message definition that you want to build.
2. Select the MQEI Message definition document.
3. Click "Build Message(s)".

You are asked to confirm whether you want to build only the selected MQEI Message definition, or all the MQEI Message definitions.

Confirm this by clicking "Yes" for all messages, "No" for selected messages, or "Cancel" to cancel the build operation.

Note When you build an MQEI Message definition, the status bar at the bottom of the Lotus Notes window tells you which MQEI Message definition and which Field definition is currently building.

Viewing a built MQEI Message definition

When an MQEI Message definition is built, all of the fields that are contained within a single MQEI Message definition are concatenated and stored in a single Notes document.

You can view these built MQEI Message definitions in Notes:

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. From the Notes menu bar, choose View - Show - Folders.
3. From the Folders pane, select "6. Built Messages".
   All of the built MQEI Message definitions appear in alphabetical order.
4. View the built MQEI Message definition document by double clicking the built MQEI Message definition document.

Deleting a built MQEI Message definition

Caution You should only delete built MQEI Message definitions from the Folders view if the MQEI LSX returns a reason code of EIRC_DUPLICATE_DEFN (5022).

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. From the Notes menu bar, choose View - Show - Folders.
3. From the Folders pane, select "6. Built Messages".
   All of the built MQEI Message definitions appear in alphabetical order.
4. Click "Delete Built Message(s)" on the action bar.
5. Confirm that you want to delete the selected MQEI Message definition(s).

Using Field Type definitions

This section describes how to:

- Create a new Field Type definition.
- Copy a Field Type definition.
- View a Field Type definition.
- Change an existing Field Type definition.
- Delete an existing Field Type definition.

Create a new Field Type definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Field Types".
3. Click "New Field Type" on the action bar.
4. Specify values for the Field Type in the New Field Type definition panel.

   Note The maximum length for "Type Name" is 32 characters.
5. Click "Save & Close" or "Save & New" on the action bar.
   If you choose "Save & New" , you are expected to create a new Field Type definition. Only choose this option if you want to create a new Field Type definition.

Copying a Field Type definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Field Types".
3. Open the Field Type definition document by double clicking it.
4. Click "Copy" on the action bar.
5. Enter a new Type Name in the blank field.
6. Click "OK Copy" on the action bar if you want to copy the Field Type definition.

   Note Click "Cancel Copy" on the action bar if you want to cancel the copy operation.

Viewing a Field Type definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Field Types".
3. View the Field Type definition by double clicking it.

Changing a Field Type definition and its "relations"

A "relation" is a Field definition that shares the same Field Type definition as the one that you want to change.

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Field Types".
3. Select the Field Type definition document you want to change.
4. Click "Edit Field Type" on the action bar.
5. Change the values.

   Note The maximum length for "Type Name" is 32 characters.
6. Click "Save & Close" on the action bar.
   When the dialog box appears, confirm whether you really want to change the Field Type definition and its relations.
7. Re-build all the MQEI Message definitions containing Field definitions that use the Field Type definition you have just changed.

Caution You must be aware that whenever you change a Field Type definition, you will also change any other Field definitions that use that Field Type definition.

Deleting a Field Type definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Field Types".
3. Select the Field Type definition that you want to delete.
4. Click "Delete Field Type(s)" on the action bar.
5. Confirm that you really want to delete the Field Type definition when prompted.

   Note Field Type definitions cannot be deleted if they are currently being used by a Field definition.
   In order to remove a Field Type definition from the MQEI Definition database, you must change the Field entries first.

Using Field definitions

This section describes how to:

- Create a new Field definition.
- Copy a Field definition.
- View a Field definition.
- Change an existing Field definition.
- Delete an existing Field definition.
- Move Field positions within an MQEI Message definition.
- Change Field definitions by changing the Field Type information.

Creating a new Field definition

Tip Before you create a new Field definition, you may want to create some Field Type definitions that contain default values for each of your Field definitions. This may save you time later.
For more information, see "Creating a new Field Type definition" earlier in this chapter.

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to add a new Field definition to by clicking on the twisty.
4. Select the MQEI Message definition document that you want to add a new Field definition to.
5. Click "New Field" on the action bar.
6. Specify the values for each field in the form.

   Note
   The "Field Name" must not exceed 32 characters.
   The "Field Description" must not exceed 48 characters.
7. Click "Save & Close" or "Save & New" on the action bar.
   If you choose "Save & New", you are expected to create a new Field definition. Only choose this option if you want to create a new Field definition.
8. Re-build the MQEI Message definition.

Copying a Field definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to copy a Field definition from by clicking on the twisty.
4. Open the Field definition document by double clicking it.
5. Click "Copy" on the action bar.
6. Enter a new Field Name in the blank field.
7. Click "OK Copy" on the action bar if you want to copy the Field definition.

   Note Click "Cancel Copy" on the action bar if you want to cancel the copy operation.

Viewing a Field definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to view a Field definition from by clicking on the twisty.
4. View the Field definition by double clicking it.

Changing an existing Field definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition containing the Field definition you want to change by clicking on the twisty.
4. Select the Field definition you want to change.
5. Click "Edit Message / Field" on the action bar.
6. Change the values in the Field.

   Note
   The "Field Name" must not exceed 32 characters.
   The "Field Description" must not exceed 48 characters.
7. Click "Save & Close" on the action bar.
8. Re-build the MQEI Message definition.

Deleting an existing Field definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to delete a Field definition from by clicking on the twisty.
4. Select the Field definition you want to delete.
5. Click "Delete Message(s) / Field(s)" on the action bar.
6. Confirm that you want to delete the Field definition when prompted.
7. Re-build the MQEI Message definition.

   Note You can mark more than one Field definition for deletion by either using the space bar or by clicking next to the Field definition with the mouse.

Moving Field positions within a Message definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Messages & Fields".
3. Expand the MQEI Message definition you want to move a Field definition within by clicking on the twisty.
4. Select the Field definition you want to move.
5. Double-click the document to view it - do not click the "Edit Message / Field" button on the action bar.
6. Click "Re-position" on the action bar.
7. Specify which existing Field definition you want to move the selected Field definition before or after.
8. Re-build the MQEI Message definition.

   Note Alternatively, if you know the exact position, you can specify it in the Position input field of the Field definition.

Changing a Field by changing the Field Type information

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Field Types".
3. Select the Field Type definition you want to edit.
4. Click "Edit Field Type" on the action bar.
5. Change the values.

   Note
   The "Type Name" must not exceed 32 characters.
6. Click "Save & Close" on the action bar.
7. When the dialog box appears, confirm whether you want to update the Field definition(s) and its relations that use this Field Type definition.
8. In the navigation pane, click "Messages & Fields".
9. Re-build all the MQEI Message definitions containing Field definitions that use the Field Type definition you have just changed.

Caution You must be aware that whenever you change a Field Type definition, you will also change any other Field definitions that use that Field Type definition.

Using MQEI Service definitions

This section describes how to:

- Create a new MQEI Service definition.
- Copy an MQEI Service definition.
- View an MQEI Service definition.
- Change an existing MQEI Service definition.
- Delete an existing MQEI Service definition.
- Categorize a new or existing MQEI Service definition.

Creating a new MQEI Service definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Services".
3. Click "New Service" on the action bar.
4. Specify values for each of the fields in the Service definition form.

   Note The "Service Name" must not exceed 16 characters.

   Note When you create a new MQEI Service definition within the MQEI Definition database, you must use the correct case when defining values in the Connection Manager, Outbound Connection and Inbound Connection fields.
   For example, if the Queue Manager is defined within MQSeries in uppercase, the value you specify in the Connection Manager field must also be in uppercase.
5. Click "Save & Close" or "Save & New" on the action bar.
   If you choose "Save & New", you are expected to create a new Service definition. Only choose this option if you want to create a new Service definition.

Copying an MQEI Service definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Services".
3. Open the MQEI Service definition you want to copy double clicking it.
4. Click "Copy" on the action bar.
5. Enter a new Service Name in the blank field.
6. Click "OK Copy" on the action bar if you want to copy the MQEI Service definition.

   Note Click "Cancel Copy" on the action bar if you want to cancel the copy operation.

Viewing an MQEI Service definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Services".
3. View the MQEI Service definition by double clicking it.

Changing an existing MQEI Service definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Services".
3. Select the MQEI Service definition you want to change.
4. Select "Edit Service" on the action bar.
5. Change the values in the MQEI Service definition.
6. Click "Save & Close" on the action bar.

Deleting an existing MQEI Service definition

1. Open MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Services".
3. Select the MQEI Service definition(s) you want to delete.
4. Choose "Delete Service(s)" on the action bar.
5. Confirm that you really want to delete the MQEI Service definition(s) when prompted.

Categorizing a new or existing MQEI Service definition

1. Open the MQEI Definition database (e.g. mqeidata.nsf).
2. In the navigation pane, click "Services".
3. Select the Service that you want to categorize.
4. Click "Edit Service" on the action bar.
5. Enter a new category name in the Category field or choose an existing category name from the dialog list.

   Note You can select more than one existing category if you wish.
6. Click "Save & Close" on the action bar.

The MQEI Security database

About the MQEI Security database

The MQEI Security database is a Lotus Notes database that contains definitions of the enterprise security parameters for each user of the MQEI applications you create. As such, it contains the user IDs and associated authenticators (usually passwords) needed for users to authenticate with the enterprise systems your MQEI applications communicate with. The MQEI LSX reads the information from the database when your LotusScript program creates an EIService object, and passes it to the enterprise system when appropriate.

The MQEI Security database is optional as not all systems will be using security. However, if you use the MQEI Security database, your LotusScript program will be able to obtain the enterprise system user ID and passwords from it. Definitions are stored as documents in the database. You can create, copy, view, change and delete MQEI Security definitions.

There is one object definition that you can work with in this database:

- MQEI Security

For more information on Security and how you can use the MQEI Security database, see Chapter 5.

Setting up the MQEI Security database

If used, the MQEI Security database should reside on your Domino server. If your MQEI application runs under a Notes client, each user of the MQEI application should be given Author access to the MQEI Security database.

If your MQEI application runs under an agent on the Domino server the **server** should be given Author access, and the agent owner (the person who last saved the agent) should be given Reader access.

You should have Author access or higher in order to create the MQEI Security definitions your users need.

For every user name you have given Author access to, (including the server), you should create an MQEI Security definition per enterprise system that requires authentication to take place or requires that a user ID be supplied. The MQEI Security database is set up so that each user will only see the definitions that belong to that user. You will be able to enter all the information required, except the authenticator (unless the MQEI Security definition is your own). You should then instruct each of your users to edit the MQEI Security definitions belonging to that user in order to add the authenticators (usually passwords), if authenticators are relevant to the enterprise systems in question.

Once an authenticator has been entered, the MQEI Security definition is encrypted. This will prevent you from editing the definition further (unless it is your own). You can still delete the definition.

If you are developing an MQEI application that is to be deployed on several Domino servers across your organization, you are recommended to maintain one MQEI Security database per distinct set of users. You can distribute an MQEI Security database using Notes replication, but do not replicate between different sets of users.

MQEI Security definition

Each MQEI Security definition contains the following information:

- Notes User Name
- System Name
- User ID
- Authenticator
- Verify Authenticator

Notes User Name

Editable text field. This information is initially taken from your Lotus Notes User ID file but you should override this if you are setting up a new MQEI Security definition for one of your users or for a server. (It controls who is allowed to access the MQEI Security definition).

System Name

Editable text field that defines the name of the enterprise system upon which an enterprise service resides. This is the system on which the User ID and authenticator are verified. The value that you specify in this field must match the value that you have specified in the "System Name" field of an MQEI Service definition in the MQEI Definition database.

At the MQEI API, System Name is accessible via the SystemName property of the EIService class.

User ID

Editable text field where you enter your User ID (or that of your user) for the enterprise system. For more information on Security, see Chapter 5.

At the MQEI API, User ID is accessible via the UserId property of the EIService class.

Authenticator (password)

Editable text field where you enter your authenticator for the enterprise system. This is normally a password. This is an encrypted field. For more information on Security, see Chapter 5.

At the MQEI API, Authenticator is accessible via the Authenticator property of the EIService class.

Verify Authenticator (password)

Editable text field where you verify your authenticator for the enterprise system. This is normally a password. This is an encrypted field and must match that entered in the Authenticator field. For more information on Security, see Chapter 5.

At the MQEI API, Authenticator is accessible via the Authenticator property of the EIService class.

Key points:

- A new MQEI Security definition initially only requires a Notes User Name and System Name.
- The only time a Notes User Name can be entered is when a new MQEI Security definition is created.
- Authenticator and Verify Authenticator can only be entered if the Notes user name matches your Notes user id (i.e., you own the MQEI Security definition).
- Once the authenticators have been entered, the document is encrypted and can no longer be modified by anybody except the owner.

Using MQEI Security definitions

This section introduces the MQEI Security database, and also describes how to:

- Create a new MQEI Security definition.
- Copy an MQEI Security definition.
- View an MQEI Security definition.
- Modify an existing MQEI Security definition.
- Delete an existing MQEI Security definition.

Creating a new MQEI Security definition

1. Open the MQEI Security database (e.g. mqeisecu.nsf).
2. In the navigation pane, click "Security definitions".
3. Click "New Security definition" on the action bar.
4. Specify values for each of the fields in the Security definition form.

   Note You must enter a System Name, Notes User Name and User ID in the fields.
5. Click "Save & Close" or "Save & New" on the action bar.
   If you choose "Save & New", you are expected to create a new Security definition. Only choose this option if you want to create a new Security definition.

Copying an MQEI Security definition

1. Open the MQEI Security database (e.g. mqeisecu.nsf).
2. In the navigation pane, click "Security definitions".
3. Expand the System Name you want to copy an MQEI Security definition from by clicking on the twisty.
4. Open the MQEI Security definition document by double clicking it.
5. Click "Copy" on the action bar.
6. Change the user details in the MQEI Security definition.
7. Click "OK Copy" on the action bar if you want to copy the MQEI Security definition.

   Note Click "Cancel Copy" on the action bar if you want to cancel the copy operation.

Viewing an MQEI Security definition

1. Open the MQEI Security database (e.g. mqeisecu.nsf).
2. In the navigation pane, click "Security definitions".
3. Expand the System Name you want to view an MQEI Security definition from by clicking on the twisty.
4. View the MQEI Security definition by double clicking it.

Changing an MQEI Security definition

1. Open the MQEI Security database (e.g. mqeisecu.nsf).
2. In the navigation pane, click "Security definitions".
3. Expand the MQEI Security definition you want to change by clicking on the twisty.

   Note If you can't expand the twisty, this implies that you are not the owner of the security definitions or do not have high enough access rights.
4. Select the MQEI Security definition document you want to change.
5. Click "Edit Security definition" on the action bar.
6. Change the values in the MQEI Security definition.
7. Click "Save & Close" on the action bar.

Deleting an MQEI Security definition

1. Open the MQEI Security database (e.g. mqeisecu.nsf).
2. In the navigation pane, click "Systems".
3. Expand the MQEI Security definition you want to delete by clicking on the twisty.

   Note If you can't expand the twisty, this implies that you are not the owner of the security definitions or do not have high enough access rights.
4. Select the MQEI Security definition document you want to delete.
5. Click "Delete Security definition(s)" on the action bar.
6. Confirm that you really want to delete the MQEI Security definition when prompted.

Chapter 4 Design and Programming using the MQEI LSX

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

It includes:

- Getting started
- Accessing the MQEI LSX
- Samples provided
- Example LotusScript using MQEI LSX
- Setting the value of a field in an EIMessage
- Getting the value of a field in an EIMessage
- Accessing fields in an EIMessage by their position
- Message subsets
- Variant Messages
- Varying length messages
- Controlling enterprise units if work
- Using the MQEI LSX from an agent
- Data Conversion
- Error handling

What to do when you start to create your Notes MQEI applications

The following identifies the basic steps that you need to follow when you start to create your Notes MQEI applications.

The language that your enterprise application is written in is one of many, but whatever the language, somewhere you define your data structures for the messages that the enterprise application sends and receives. For example, you may have copybooks or define your data structures in working storage if you are using COBOL. If you are using C, then you will probably have C header files.

Field Type definitions

Find where the data structures are defined for your enterprise application.

Examine the copybook, C header file, or however else they are defined and identify the common field types, e.g. 30 character strings for names, dates in mmddyyyy format, dates in ddmmyyyy format, 10 character strings for account numbers.

In Notes, go into your MQEI Definition database.

If you are installing MQEI for the first time, refer to Chapter 2 Getting Started.

Select Field Types from the navigator and create those that you need.

MQEI Message definitions

Go back to the navigator and select Messages and Fields.

Define a message, followed by the fields needed in the message to match the data structure your enterprise application is expecting, using the Field Types you have already set up whenever they are appropriate.

Continue defining messages until you have all that you need.

Remember to build the messages when they are complete, otherwise your LotusScript program cannot use them.

MQEI Service definitions

Before you can use the MQSeries Enterprise Integrator you must create an MQEI Service definition for the enterprise application you want to access.

Before you use the MQEI definitions with your LotusScript program

Before you can use the MQEI definitions with your LotusScript program, you must build the MQEI Message definitions. If you have forgotten to add any definitions you will need to add them; you can change a definition if an error has been made or a change has been made to the enterprise application, but always remember to build your MQEI Message definitions.

If you fail to build an MQEI Message definition and a previously built version exists in the database, LotusScript picks up the built version. Depending on the change and the checking you do within your LotusScript program, you may get unexpected results or the program may fail.

For more information about how to create definitions on the MQEI Definition database and MQEI Security database, see Chapter 3.

Accessing the MQEI LSX

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

```
Uselsx "eilsx"
```

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

MQEI Samples

To give you further help, a sample is provided to show how you can use the MQEI to communicate with each of the supported enterprise services. Each sample is a Lotus Notes application containing scripts that use the MQEI LSX to interact with an enterprise service.

MQEI Sample applications and the Web

To run Notes applications over the Web, they must be Web enabled - that is, written using a subset of Notes functions. The MQEI samples provided in the MQEI Samples database are not written for use over the Web and you will not be able to access them from your Web browser.

For web enabled samples, further information, and help, see the IBM Red book:

Lotus Notes and the MQSeries Enterprise Integrator, available at URL:

http://www.redbooks.ibm.com/SG242217/sg242217.html

or order book no: SG24-2217

Components

The Notes databases provided are:

- The MQEI Samples database
  This is a Notes database that contains the sample forms.
- The MQEI Definition database
  This contains MQEI Message and Service definitions that are used by all the samples.
- The MQEI Security database
  This contains MQEI Security definitions that you may need to modify if you want to run the samples with security.

Two command files are provided to define resources for the MQSeries samples:

- mqeisamp.tst sets up local resources on your workstation
- mqeisamp.jcs defines the resources for MQSeries for MVS/ESA

Error checking

The samples are 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, the samples are suitable to use as a base for your own LotusScript programs.

Error checking takes place throughout all the samples.

MQEI warnings are handled by displaying a message box specific to the warning; processing continues.

MQEI errors are handled using:

- ON ERROR routines and
- EVENT handlers.

For each object there is a subroutine that handles errors for the object. The subroutine is designed to check for specific reason codes and take appropriate action. The error codes are cleared before continuing.

For more information, see Appendices A-F.

Example LotusScript using MQEI LSX

The following example illustrates the basic MQEI LSX building blocks that you would use in your LotusScript program:

```
Dim sess As EISession
Dim serv As EIService
Dim msg1 As EIMessage
Dim msg2 As EIMessage
Dim sopts As EISendOptions
Dim ropts As EIReceiveOptions
.......
'*************************************************
'*Create a new session. The MQEI LSX reads the
'*mqei.ini file to determine the path for the
'*MQEI Definition database and MQEI Security
'*database.
'*The MQEI Definition database is opened. The
'*MQEI Security database is opened if the path and
'*name are included in the mqei.ini file.
'*************************************************
Set sess = New EISession
'*************************************************
'*Create a new EIService object based on an MQEI
'*Service definition on the MQEI Definition database.
'*This reads the MQEI Security database, if one
'*exists, for the userid and authenticator for the
'*target service.
'*************************************************
Set serv = sess.CreateService("name_of_service")
'*************************************************
'*Create a new EIMessage object for each message,
```

```
'*based on the MQEI Message definitions on the MQEI
'*Definition database.
'*Note: If the message you receive is an updated
'*version of the one you send, or vie versa, you
'*would only create one EIMessage object.
'*************************************************
Set msg1 = sess.CreateMessage("msg_out_name")
Set msg2 = sess.CreateMessage("msg_in_name")
'*************************************************
'*Create an EISendOptions object and
'*EIReceiveOptions object, using the default values.
'*************************************************
Set sopts = sess.CreateSendOptions
Set ropts = sess.CreateReceiveOptions
'*************************************************
'*Connect to the enterprise service via the
'*connection manager specified in the MQEI Service
'*definition on the MQEI Definition database.
'*************************************************
Call serv.Connect
'*************************************************
'*At this point you can now set the fields in the
'*EIMessage object, which in this case is called
'*msg1. If you want to use specific send options,
'*set them before sending each message.
'*************************************************
Call serv.SendMessage(msg1,sopts)
'*************************************************
'*If you want to use specific receive options, set
'*them before receiving each message.
'*************************************************
```

```
Call serv.ReceiveMessage(msg2,ropts)
'************************************************
'*You can now work with the contents of the
'*EIMessage you have received from the enterprise
'*service. At the end you must break your connection
'*to the service and delete the objects you have
'*previously created.
'************************************************
Call serv.Disconnect
Delete ropts
Delete sopts
Delete msg1
Delete msg2
Delete serv
Delete sess
```

Note The check to ensure all MQEI LSX EIMessage method and property names are spelt correctly takes place at run time, **not** compile time. For all other classes this is done at compile time.

Setting the value of a field in an EIMessage

The syntax of the MQEI interface in LotusScript allows you to perform the same function in a number of ways. The following example demonstrates the options you have when setting a field in an EIMessage to a particular value.

To do this, consider the situation where in your MQEI Definition database you define a message and name it MessageA. MessageA has 3 fields in it called FieldA, FieldB, and FieldC.

FieldA, the first field in the message, has a data type of String (fixed or varying)

FieldB, the second field in the message, has a data type of Long

FieldC, the third field in the message, has a data type of Byte

There are various ways you can set the value of a Field using the MQEI LSX.

Each of the following examples is preceded with:

```
Dim myMessageA As EIMessage

myMessageA = mySession.CreateMessage("MessageA")
```

Setting a field of data type String

The following applies to a field defined as a fixed length string and to a field defined as a varying length string.

- In this example, FieldA is set to the string "a string of data":

```
myMessageA.FieldA = "a string of data"
```

or

```
Call myMessageA.SetFieldValue("FieldA","a string of data")
```

or

```
Call myMessageA.SetFieldValue(myMessageA.GetFieldName(1),"a
string of data")
```

- In this example, FieldA is set to the string "123" using a LotusScript variant:

```
myMessageA.FieldA = "123"
```

or

```
myfldvar = "123"

myMessageA.FieldA = myfldvar
```

In this case the LotusScript variant takes on the data type of string because quotes are placed around the 123.

If the quotes are omitted, the LotusScript variant takes on a data type of Integer and you cannot assign an Integer to a string. In this case the MQEI LSX returns the completion code EICC_FAILED with the reason code EIRC_INVALID_DATATYPE.

Setting a field of data type Long

In this example, FieldB is set to the value 180 (the range for Long is -2147483647 to +2147483647):

```
myMessageA.FieldB = 180
```

or

```
Call myMessageA.SetFieldValue("FieldB",180)
```

In this example, the MQEI LSX is checking that the value 180 is acceptable to a field with data type Long. If you attempt to set a field to a value outside the range allowed by a data type, you get the completion code EICC_FAILED with the reason code EIRC_DATA_OVERFLOW.

Alternatively:

```
Dim myLong As Long

myLong = 180
```

followed by:

```
myMessageA.FieldB = myLong
```

or

```
Call myMessageA.SetFieldValue("FieldB",myLong)
```

In this example, LotusScript checks that the value is within the bounds allowed by the data type.

Example using a LotusScript variant:

```
myMessageA.FieldB = 180
```

or

```
myfldvar = 180

myMessageA.FieldB = myfldvar
```

In this case the LotusScript variant is a numeric and is within the bounds of a Long, therefore the setting of the field is successful. If the number used is outside the bounds of a long, the MQEI LSX returns the completion code EICC_FAILED with the reason code EIRC_DATA_OVERFLOW.

Note You cannot set a numeric field to a string:

```
myMessageA.FieldB = "180"
```

or

```
myfldvar = "180"

myMessageA.FieldB = myfldvar
```

In this case the LotusScript variant is a string and is rejected by the MQEI LSX with completion code EICC_FAILED with the reason code EIRC_INVALID_DATATYPE.

Setting a field of data type Byte

In this example, FieldC is set to the value 115 (the range for Byte is -127 to +127):

```
myMessageA.FieldC = 115
```

or

```
Call myMessageA.SetFieldValue("FieldC",115)
```

In this example, the MQEI LSX is checking that the value 115 is acceptable to a field with data type Byte. If you attempt to set a field to a value outside the range allowed by a data type, you get the completion code EICC_FAILED with the reason code EIRC_DATA_OVERFLOW.

Alternatively:

```
Dim myByte As Integer

myByte = 115
```

followed by:

```
myMessageA.FieldC = myByte
```

or

```
Call myMessageA.SetFieldValue("FieldC",myByte)
```

In this example, LotusScript checks that the value is within the bounds allowed by the data type of Integer. LotusScript does not have a data type of Byte, hence the reason for using Integer; if a field of data type Byte is set to a value out of range, it is rejected by the MQEI LSX with the completion code of EICC_FAILED and the reason code of EIRC_DATA_OVERFLOW.

Example using a LotusScript variant:

```
myMessageA.FieldC = 115
```

or

```
myfldvar = 115

myMessageA.FieldC = myfldvar
```

In this case the LotusScript variant is a numeric and is within the bounds of a Byte, therefore the setting of the field is successful. If the number used is outside the bounds of a Byte, the MQEI LSX returns the completion code EICC_FAILURE with the reason code EIRC_DATA_OVERFLOW.

Note You cannot set a numeric field to a string:

```
myMessageA.FieldC = "115"
```

or

```
myfldvar = "115"

myMessageA.FieldC = myfldvar
```

In this case the LotusScript variant is a string and is rejected by the MQEI LSX with completion code EICC_FAILED with the reason code EIRC_INVALID_DATATYPE.

Getting the value of a field in an EIMessage

In your MQEI Definition database you define a message and name it MessageA. MessageA has 3 fields in it called FieldA, FieldB, and FieldC.

FieldA, the first field in the message, has a data type of String (fixed or varying)

FieldB, the second field in the message, has a data type of Long

FieldC, the third field in the message, has a data type of Byte

There are various ways you can get the value of a Field in an EIMessage using the MQEI LSX. You can put the value into a variable that you specify to be the same as the data type of the field in the MQEI Definition database, or you can use a LotusScript variant.

Note If you use a LotusScript variant, the syntax is always the same because the LotusScript variant takes on the data type of the field to which you set it .

Each of the following examples is preceded with:

```
Dim myMessageA As EIMessage

myMessageA = mySession.CreateMessage("MessageA")
```

Getting a field of data type String

The following applies to a field defined as a fixed length string and to a field defined as a varying length string.

- Example using a variable with a data type of String:

```
myfieldx$ = myMessageA.FieldA
```

or

```
myfieldxname$ = myMessageA.GetFieldName(1)
myfieldx$ = myMessageA.GetFieldValue(myfieldxname$)
or
Set myfieldx$ = myMessageA.GetFieldValue("FieldA")
```

or

```
Set myfieldx$ =
myMessageA.GetFieldValue(myMessageA.GetFieldName(1))
```

Alternatively:

```
Dim myfieldx As String
```

followed by:

```
myfieldx = myMessageA.FieldA
or
myfieldxname = myMessageA.GetFieldName(1)
myfieldx = myMessageA.GetFieldValue(myfieldxname)
or
Set myfieldx = myMessageA.GetFieldValue("FieldA")
or
Set myfieldx =
myMessageA.GetFieldValue(myMessageA.GetFieldName(1))
```

Note You can get a numeric field and set it to a string, LotusScript converts the numeric into a string.

- Example using a LotusScript variant:

```
myfldvar = myMessageA.FieldA
```

Getting a field of data type Long

- Example using a variable with a data type of Long:

```
myfieldy& = myMessageA.FieldB
```

or

```
myfieldyname$ = myMessageA.GetFieldName(2)

myfieldy& = myMessageA.GetFieldValue(myfieldyname$)
```

or

```
Set myfieldy& = myMessageA.GetFieldValue("FieldB")

or

Set myfieldy& =
myMessageA.GetFieldValue(myMessageA.GetFieldName(2))
```

Alternatively:

```
Dim myfieldy As Long
```

followed by:

```
myfieldy& = myMessageA.FieldB
```

or

```
myfieldy& = myMessageA.GetFieldName(2)
```

or

```
Set myfieldy& = myMessageA.GetFieldValue("FieldB")

or

Set myfieldy& =
myMessageA.GetFieldValue(myMessageA.GetFieldName(2))
```

Note LotusScript checks that the field you are getting fits into the field you are setting it to. In this example, as well as being able to set a field of data type Long into 'myfieldy', you could also set a field of data type Byte into it. If the field does not fit, LotusScript raises an 'overflow 'error at run time. If you try to get a field of data type string and set it to a numeric, LotusScript raises a 'type mismatch' error.

- Example using a LotusScript variant:

```
myfldvar = myMessageA.FieldB
```

Getting a field of data type Byte

- Example using a variable with a data type of Byte:

```
myfieldz% = myMessageA.FieldC
```

or

```
myfieldzname$ = myMessageA.GetFieldName(3)

myfieldz% = myMessageA.GetFieldValue(myfieldxname$)
```

or

```
Set myfieldz% = myMessageA.GetFieldValue("FieldC")

or

Set myfieldz% =
myMessageA.GetFieldValue(myMessageA.GetFieldName(3))
```

Alternatively:

```
Dim myfieldz As Integer
```

followed by:

```
myfieldz% = myMessageA.FieldC
```

or

```
Set myfieldz% = myMessageA.GetFieldValue("FieldC")

or

Set myfieldz% =
myMessageA.GetFieldValue(myMessageA.GetFieldName(3))
```

Note LotusScript checks that the field you are getting fits into the field you are setting it to. In this example, as well as being able to set a field of data type Byte into 'myfieldz', you could also set a field of data type Short into it. If the field does not fit, LotusScript raises an 'overflow 'error at run time. If you try to get a field of data type string and set it to a numeric, LotusScript raises a 'type mismatch' error.

Note The data type Byte is not known to LotusScript, so any data type that accepts a numeric value is used.

- Example using a LotusScript variant:

```
myfldvar = myMessageA.FieldC
```

Accessing fields in an EIMessage by their position

The MQEI enables you to Get and Set a field in an EIMessage by using the name of the field. However, there may be situation where you do not want to hard-code the name of a field in your LotusScript program. In this situation, consider using the EIMessage GetFieldName method . You provide the position the field has in the message, the field index, and the name of that field is returned in the variable you provide.
For example:

Set *aname$ = myMessage*.**GetFieldName**(4)

would cause the variable *aname$* to be set to the name of the fourth field in the message.

If you want to know the value in the field, use the EIMessage GetFieldValue method.

For example:

Set *Fval = myMessage*.**GetFieldValue**(*aname$*)

If you want to set the field to a value, use the SetFieldValue method.

For example:

Call myMessageA.**SetFieldValue**("FieldB","string of data")

Message Subsets

When receiving an enterprise message using the EIService ReceiveMessage method, your LotusScript program may not be interested in all the fields. The MQEI LSX allows the EIMessage passed on the ReceiveMessage call to specify a subset of the fields in the message. In this case, the ReceiveMessage call only completes those fields in the EIMessage and ignores the rest of the enterprise message.

If the enterprise message represents a screen, where each field is located via row and column attributes, the fields in the EIMessage can be any subset of fields in the screen.

Otherwise, the fields in the EIMessage must be a contiguous subset, starting at the beginning, of the fields in the enterprise message.

For example, if you are not interested in say the first 10 fields of an EIMessage, but you are interested in 11 through to 15 inclusive, and you cannot locate the fields by row and column, you can either:

- Define fields 1 through to 10 as a single field, of data type string, followed by fields 11 through to 15 individually.
- Define all fields, 1 through to 15, individually.

Note To avoid any possible data conversion problems, you are recommended to define all fields in an EIMessage up to and including the last one you are interested in.

Variant Messages

One common circumstance is where an inbound enterprise message is expected, but the exact format of the message is not known. Such a message is called a *variant message.* An example of a variant message could be receiving either a data message or an error message from an enterprise application depending on the results of its processing.

This section explains how you can use the MQEI to receive a variant message into an EIMessage object. It builds on the ability of an EIService ReceiveMessage call to receive an enterprise message into the MQEI LSX inbound buffer, using MQSeries MQGET for example, and selectively map a subset of the message into the EIMessage object returned to the program (see Message Subsets).

Messages with tags

Often an enterprise message contains a field at or near the start that identifies the type of message it is. This field is sometimes known as a *tag* field. To make use of these you need to:

- Create an MQEI Message definition that maps just the tag field and those that precede it.
- Make a ReceiveMessage call as normal, passing the EIMessage object created from the MQEI Message definition. The MQEI LSX receives the message into the MQEI LSX inbound buffer, but returns only those fields in the enterprise message corresponding to those in the EIMessage object.
- Within your LotusScript program examine the value of the tag field and discover the format of the message.
- Make another ReceiveMessage call within your LotusScript program, passing an EIMessage object representing that particular message format, with the EIReceiveOptions ReceiveType property set to EIRT_RETURN. The MQEI LSX uses the message in the inbound buffer, and returns the completed EIMessage object.

Messages without tags

If there is no tag field in the enterprise message, a different technique must be adopted:

- Make a ReceiveMessage call passing the EIMessage object that represents the most likely format of the enterprise message. If the MQEI LSX detects that the formats do not match, a CompletionCode of EICC_WARNING and a ReasonCode of EIRC_WRONG_MESSAGE or EIRC_INSUFFICIENT_DATA will be returned. If this occurs, make another ReceiveMessage call within your LotusScript program, this time passing the EIMessage object that represents the next most likely format, with EIReceiveOptions ReceiveType property set to EIRT_RETURN. And so on, until the correct EIMessage object is used.

This latter technique can be used to good effect when receiving messages from a CICS 3270 direct service. The messages in this case are CICS screens, normally defined by BMS maps. The CICS 3270 direct sample uses variant message processing to discover whether CICS has returned an informational message on row 1, an error message on row 23, or the CESN transaction BMS map.

Note When one message format can be a subset of another message format, it is advisable that the first ReceiveMessage is passed the EIMessage object representing the larger format, because the MQEI LSX allows an EIMessage to be a subset of the real message.

Native MQSeries service type

If the enterprise service type is Native MQSeries, you can use the EIReceiveOptionsFormat property as an indicator of the format of the received enterprise message. If the enterprise application supplies a named format for the message, the EIReceiveOptions Format property will contain this value after a ReceiveMessage call.

The ReceiveMessage call also checks that the Format property of the passed EIMessage object matches the named format in the message. If it does not, a CompletionCode of EICC_WARNING and a ReasonCode of EIRC_WRONG_MESSAGE are returned. If this occurs, examine the EIReceiveOptions Format property within your LotusScript program to discover the format of the received message. Make another ReceiveMessage call within your LotusScript program, this time passing the EIMessage object with a matching Format property, with the EIReceiveOptions ReceiveType property set to EIRT_RETURN.

Note The format check only takes place if EIMessage Format is not EIFMT_NONE and the named format is not blank.

IMS via MQSeries service type

If the enterprise service type is IMS via MQSeries, you can use the EIReceiveOptions Format property as an indicator of the format of the received enterprise message. In this case, the IMS MOD name sent by IMS may identify the message format, and the EIReceiveOptions Format property will contain this value after a ReceiveMessage call.

The ReceiveMessage call also checks that the Format property of the passed EIMessage object matches the IMS MOD name. If it does not, a CompletionCode of EICC_WARNING and a ReasonCode of EIRC_WRONG_MESSAGE are returned. If this occurs, examine the EIReceiveOptions Format property within your LotusScript program to discover the format of the received message. Make another ReceiveMessage call within your LotusScript program, this time passing the correct EIMessage object with a matching Format property, with the EIReceiveOptions ReceiveType property set to EIRT_RETURN.

Note The format check only takes place if EIMessage Format is not EIFMT_NONE and the IMS MOD name is not blank.

Alternatives

A variation on these techniques is not to pass an EIMessage on the first ReceiveMessage call. You do this using the reserved keyword **Nothing** for the EIMessage parameter. The MQEI LSX receives the enterprise message into its inbound buffer, and sets the EIReceiveOptions Format property if appropriate, but does not attempt to return an EIMessage object.

This means a ReasonCode of EIRC_WRONG_MESSAGE will not occur on this first ReceiveMessage call. Make subsequent ReceiveMessage calls within your LotusScript program, passing an EIMessage object with the EIReceiveOptions ReceiveType property set to EIRT_RETURN, until the format is established.

This may result in a simplified program, because the code responsible for determining the variant message format can be separated from the code performing the actual receive of the message.

Varying length messages

When receiving a message that contains a repeating field at the end, and the exact number of occurrences of the repeating field is not known, the techniques described in Variant Messages have their limitations.

For example, if the number of occurrences of the repeating field varied from 10 to 99, then 90 MQEI message definitions would be required.

An alternative technique may be used in these circumstances:

- You may consider the repeating field to be one long string field. Create an MQEI Message definition that contains, as its last field, a Field definition of data type String, string type Variable Length, and string format Null Terminated.
- Make a ReceiveMessage call as normal, passing the EIMessage object created from the MQEI Message definition. The MQEI LSX receives the message into the MQEI LSX inbound buffer, and returns the completed EIMessage object.
- Within your LotusScript program assign the value of the last EIMessage field to a local string variable. Use the LotusScript **len** function to get the length of the data, which should be a multiple of the length of the repeating field. It is up to your LotusScript program to interpret the data in the local string variable.

Note

This technique can be used where the repeating field is in fact a repeating structure, as long as all the members of the structure have data type String. If numeric data types are involved, the technique does not work because the entire structure is data converted as if it were a string. Again, it is up to your LotusScript program to interpret the data.

This technique can also be used to receive a message that contains a varying length string as the last field in the message. Taking this further, if a message in its entirety is considered to be a varying length string, then this technique can be used to receive any message. Again, this is not appropriate for messages that contain numeric data, and it is up to your LotusScript program to interpret the data.

This technique is not appropriate for CICS screens received from a CICS 3270 service, or for messages received from IMS via MQSeries where the repeating field spans more than one message segment.

Controlling enterprise units of work

If the enterprise service you wish to communicate with supports transactional concepts, any message you send to the service will participate in a logical unit of work (UOW) on the enterprise system. This will be the case if the enterprise system is CICS or IMS.

Your LotusScript program can use the EISendOptions UnitOfWork property to control the enterprise unit of work, if the type of service supports this.

- If the type of service is CICS DPL direct or CICS DPL via MQSeries, enterprise units of work can be controlled.

For more information, see "Programming for a CICS DPL direct service" in Chapter 9 and Programming for a CICS DPL via MQSeries service" in Chapter 8, and the EISendOptions UnitOfWork description in Chapter 12.

- If the type of service is CICS 3270 direct or IMS via MQSeries, enterprise units of work can not be controlled. The units of work are dictated by the CICS or IMS transactions you run.
- If the type of service is Native MQSeries, enterprise units of work can not be controlled. Whether or not your message is part of an enterprise unit of work depends on the MQSeries application and the system it is running on.

Note A Notes client or Domino server does not support transactional concepts, so cannot be used to coordinate units of work on distributed systems. Each enterprise unit of work that your MQEI application initiates runs independently of others.

Note The EISendOptions UnitOfWork property always refers to a unit of work on the enterprise system. It has no effect on local commit or backout of messages, such as that provided by MQSeries. Control over local commit or backout is not supported by the MQEI LSX.

Data conversion

You do not have to deal with data conversion within your LotusScript program when using the MQEI LSX. On EIService SendMessage and ReceiveMessage calls, the MQEI LSX converts the data in an EIMessage for you, using the Encoding and CharacterSet defined by the MQEI Service definition.

What the MQEI LSX does

The MQEI LSX converts the data in an EIMessage for:

- Outgoing messages by comparing the Encoding and CharacterSet values specified on the [Base] stanza in your mqei.ini file against those associated with the EIService object (specified in the MQEI Service definition in the Definition database).
- Incoming messages for Services not using MQSeries (CICS DPL direct and CICS 3270 direct), in the same way as outgoing messages.
- Incoming messages for Services using MQSeries (Native MQSeries, CICS DPL via MQSeries, and IMS via MQSeries), by comparing the incoming MQSeries control information associated with the message with that of the mqei.ini [Base] stanza.

Conversion (when necessary) is done on a field by field basis - fields whose data types are numeric undergo translation based on the Encoding comparison, fields whose data types are character based undergo translation based on the CharacterSet comparison.

The reason code EIRC_DATA_CONVERSION_ERROR may be returned from the ReceiveMessage or SendMessage methods for CharacterSet conversions. This means that the MQEI LSX cannot find the necessary conversion tables to support a requested conversion.

The environment variable MQEI_XLAT_PATH is used by the MQEI LSX to establish the location of the conversion tables.

For more information on how these tables are used, what tables are supplied and possible problems associated with conversion, see Chapter 11.

For more information, see "Data conversion" in chapters 6 to 10 inclusive.

Using the MQEI LSX from an agent

The LotusScript NotesTimer class function does not work from an agent. In this situation, an agent must use a blocking ReceiveMessage call with a timeout. The CICS 3270 direct service does not provide a timeout facility, so a block is unlimited.

There are also extra aspects to consider in respect of security when you use agents. For more information, see "Notes agents" in Chapter 5.

From Domino Release 4.5.1 onwards, agent managers on a Domino server may run agents concurrently on separate threads in the http process used for Domino Web access. The MQEI LSX is capable of operating in this environment, so you can take advantage of this Domino feature when using the MQEI LSX.

Error handling

Each MQEI LSX object includes properties to hold error information and a method to clear (reset) them.

The properties are:

- CompletionCode
- ReasonCode

The method is:

- ClearErrorCodes

In addition, each object can also raise the following events:

- Eierror
- Eiwarning

How it works

When your MQEI LSX program invokes an MQEI LSX object method, or accesses a property of an MQEI LSX object the:

- ReasonCode and CompletionCode in the object concerned are updated.

If the action performed is successful, the values of these properties are:

- CompletionCode = EICC_OK
- ReasonCode = EIRC_NONE

However, if some part of the action was not successful, and the CompletionCode of the object is equal to EICC_FAILED:

- If an Eierror event handler has been registered for the object, it is called.
- If the CompletionCode is still EICC_FAILED, even after some event handling code, the MQEI LotusScript Error (EILSX_ERROR) is raised.

Alternatively, if some part of the action was not successful and the CompletionCode of the object is equal to EICC_WARNING:

- If an Eiwarning event handler has been registered for the object, it is called.

Note The MQEI LotusScript Error is never called for warnings.

You may wish to deal with all errors in a single place, in which case you do **not** register event handlers. In this situation you use the On Error statement in your LotusScript program to declare the part of the script to handle all MQEI LSX errors.

MQEI LotusScript Error

This LotusScript Error has the special value EILSX_ERROR, you can use this within your script by calling the On Error statement in your LotusScript to process it.

For example:

```
On Error EILSX_ERROR GoTo .....
```

Use the Error$ function to retrieve the associated error string. This is in the form:

EILSX: ReasonCode=nnnnn

where nnnnn is the ReasonCode of the MQEI LSX object in error.

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

MQEI Event Handlers

You can write event handlers to process the Eierror and Eiwarning events generated by MQEI LSX objects (the exception is the creation of the EISession object). All MQEI LSX events have 2 parameters:

- An object reference to the MQEI LSX object that generated the event
- The ReasonCode of the MQEI LSX object at the point the event was generated

All MQEI LSX event handlers must specify these two parameters.

For example, if an event handler is needed to process the Eierror event from an EIMessage object, you need the following:

Mainline script:

```
...
Dim myMessage As EIMessage
Set myMessage = mySession.CreateMessage("MSGA")
On Event Eierror From myMessage call handleMSGError
```

The handleMSGError subroutine could be coded as follows:

```
Sub handleMSGError(myMSG as EIMessage, myReasonCode as Long)
        If myReasonCode = EIRC_FIELD_NOT_FOUND then
                Messagebox "Field was not found"
                myMSG.clearErrorCodes
        End If
    ...
End Sub
```

Errors within an Event Handler

For any error encountered within an event handler:

- The ReasonCode and CompletionCode properties for the object concerned are changed to reflect the error.
- An MQEI event is **not** raised.
- The MQEI LotusScript error is **not** produced.

If an event handler is taking corrective action in respect of an object, you are recommended to do in-line error checking within the event handler. Call the ClearErrorCodes method for the object to indicate to the MQEI that you have resolved the error.

Note If you have resolved an error or want to bypass the error, you **must** call the ClearErrorCodes method for the object. Neither the MQEI LotusScript error or MQEI events are raised for subsequent errors on the object in question, until the ClearErrorCodes method has been run.

Programming event handling routines

The following guidelines are intended to help you to code effective error handling routines using the MQEI LSX:

- Register an event handler for Eierror events for all MQEI LSX objects created in your application.

Note This does not mean a separate event handler for each and every object since objects of the same type can share event handlers.

- Each event handler should process those ReasonCodes that can be handled, calling the ClearErrorCodes method when complete
- ReasonCodes that cannot be handled in the event handler, can be passed to a general LotusScript Error routine

These ideas can be seen in the following code fragments:

Mainline script**:**

```
On Error EILSX_ERROR GoTo CatchAll

Dim mySession As New EISession

Dim myService As EIService

Dim myMSGIN As EIMessage

Dim myMSGOUT As EIMessage

' >>> Create the EIService object and register the Service
event handler

Set myService = mySession.CreateService("SERVICE")
```

```
On Event Eierror From myService call handleServiceError

' >>> Create the EIMessages and register the Message event
handler,

' >>> NOTE: The 2 Message objects use the same event handler

Set myMSGIN = mySession.CreateMessage("MSGIN")

Set myMSGOUT = mySession.CreateMessage("MSGOUT")

On Event Eierror From myMSGIN call handleMSGError

On Event Eierror From myMSGOUT call handleMSGError

...
```

CatchAll:

```
   Messagebox "MQEI LSX LotusScript error not handled by
event handlers gave _ message = " &  Error  & " at line
number  " & Erl

   Exit Sub
```

Event handler script:

```
Sub handleSessionError(mySess As EISession, myReasonCode As
Long)
        Select Case myReasonCode
                Case EIRC_INVALID_OBJECT_NAME:
                        ...
                        mySess.ClearErrorCodes
                ...
                Case Else:
                        Print "ReasonCode = " & myReasonCode
& " not handled _ here"
        End Select

End Sub

Sub handleServiceError(myServ As EIService, myReasonCode As
Long)
        Select Case myReasonCode
                Case EIRC_INVALID_SYSTEM_NAME:
                        ...
                        myServ.clearErrorCodes
```

```
                ...
                Case Else:
                        Print "ReasonCode = " & myReasonCode
& " not handled _ here"
        End Select
End Sub
Sub handleMSGError(myMSG As EIMessage, myReasonCode As Long)
        Select Case myReasonCode
                Case EIRC_FIELD_NOT_FOUND:
                        ...
                        myMSG.ClearErrorCodes
                ...
                Case Else:
                        Print "ReasonCode = " & myReasonCode
& " not handled here"
        End Select
End Sub
```

Alternatives

You may wish to deal with all errors in a single place, in which case, you should never register event handlers. To do this you use the On Error statement in your LotusScript program to declare the part of the script to handle all MQEI LSX errors.

Error handling for the EISession object

The MQEI EISession object is a special case since:

- It is the only MQEI object that can be created using the LotusScript new method
- There is only ever one EISession object; any attempt to create more than one of these objects gives a reference to the first

For these reasons, error handling is a little different for the EISession object.

EISession creation

An EISession object is created using the new method supplied by LotusScript. MQEI events or the MQEI LotusScript error are **not** generated from this call, so an alternative way must be used to check on the status of the new call.

This can be done in line, for example:

```
Dim mySession As New EISession
If mySession.reasonCode <> EIRC_NONE Then
        Select Case mySession.reasonCode
     Case EIRC_INI_OPEN_ERROR:
         ...
     Case EIRC_INI_KEYWORD_NOT_FOUND:
         ...
     Case EIRC_INI_SECTION_NOT_FOUND:
         ...
     Case EIRC_DEFN_DB_SYSERROR:
       ...
     Case EIRC_SEC_DB_SYSERROR:
       ...
     Case Else:
       ...
   End Select
End If
```

Errors in the creation of the EISession object tend to be setup problems. You can identify and subsequently resolve these problems by running the Post-Installation Check program.

For more information, see Post-Installation Check in Chapter 2.

EISession persistence

You should note that since the EISession persists as long as Notes persists (on your Notes client or Domino Server), it retains the CompletionCode and ReasonCode from the last LotusScript run. Although these can be cleared using the ClearErrorCodes method, it is **not** recommended since errors in the EISession tend to be setup problems that need to be solved before any MQEI script can successfully complete.

Handling Warnings

Eiwarning events can be handled in exactly the same manner as the Eierror events except that the MQEI LSX LotusScript Error is not raised for events of this severity. However, warnings lend themselves well to in-line checking of ReasonCodes.
For example, if your application wishes to receive all the messages before proceeding:

```
Do until myService.ReasonCode = EIRC_NO_MESSAGE_AVAILABLE
        Call myService.ReceiveMessage(myMSGOUT)
Loop
```

Since no Eiwarning event handlers have been registered, they can never be called, and since EIRC_NO_MESSAGE_AVAILABLE is a warning, the code remains in-line.

Chapter 5 Security

This chapter explains the security mechanisms available when using the MQEI LSX to communicate with enterprise systems. It explains:

- The UserId and Authenticator properties in the EIService class
- How the MQEI Security database is used
- Authenticator and System_Authenticator data types
- Extra considerations when using Notes agents

General

If you are developing an MQEI application which communicates with an enterprise application, the enterprise application may be running in a secure environment. For example, it may require a userid and password so it can perform authentication of its users, or it may simply require a userid so it can identify users for system use charging or problem contact purposes. If this is the case, the MQEI LSX must pass the appropriate enterprise userid and authenticator (if required) to the enterprise application.

The MQEI LSX provides two properties of the EIService class to enable this to take place:

- UserId property
- Authenticator property

The MQEI LSX also provides a repository for enterprise userids and authenticators to enable your MQEI applications to seamlessly access secure enterprise applications without prompting the application user for a userid and authenticator. This repository is a Lotus Notes database called the MQEI Security database. It is particularly useful if your MQEI application is a Notes agent running on a Domino Server, where prompting for a userid and authenticator is not practical.

The MQEI Security database is described in "The MQEI Security database" in Chapter 3.

Each type of enterprise service has different security requirements, and different methods of performing authentication. The MQEI LSX hides this from you by automatically passing the enterprise userid and authenticator to the enterprise service where it is possible for it to do so. However there are certain types of service where your LotusScript program must explicitly pass the enterprise userid and authenticator to the enterprise service by setting them in an EIMessage object.

For more information, see "Security" in the Chapter (6 to 10) appropriate to the type of service your MQEI application is using.

UserId property and security

The EIService UserId property is your userid on the enterprise system (not your Notes userid), and may be different for each Service you use. The MQEI LSX uses the UserId , if required, to logon or signon to the enterprise system described by the EIService.

The MQEI LSX can use it with a password (authenticator) to meet system security requirements, but it can be used by itself for other purposes. The userid in a message enables your enterprise system to identify the owner of a piece of work, for example, when charging for the use of a system or as a point of contact when a problem occurs.

To set the UserId property you can:

- Extract it from the MQEI Security database (this is done automatically when the EIService object is created if an appropriate MQEI Security definition exists).
- Prompt the person running the application and use LotusScript to set it:

```
userid$ = Inputbox$("Please enter your userid")
serv.userid= userid$
```

- Extract it from somewhere that is specific to your environment and use LotusScript to set it:

```
userid$ = "myname"
serv.userid= userid$
```

Authenticator property and security

The EIService Authenticator property holds a password or ticket, hence you can only write to it. If you attempt to read it, you are returned the value EIAUT_HIDDEN.

To set the EIService Authenticator property, you can:

- Extract it from the MQEI Security database, this is done automatically when the EIService object is created if an appropriate MQEI Security definition exists.
- Prompt the person running the application and use LotusScript to set it:

```
auth$ = Inputbox$("Please enter your password")

serv.authenticator= auth$
```

- Extract it from somewhere that is specific to your environment and use LotusScript to set it:

```
serv.authenticator=.....
```

How the MQEI Security database is used

Whenever you create an EIService object the MQEI Definition database is read to get the details of the corresponding MQEI Service definition. Amongst the information held about a service is the SystemName property. If there is an MQEI Security database, the MQEI LSX reads it, using your Notes userid and the SystemName property as the key to locate an MQEI Security definition:

- If no MQEI Security definition is found, the UserId property is set to EIUI_NOT_APPLICABLE and the Authenticator property is set to blanks. This is not considered to be an error and the MQEI LSX continues.
- If an MQEI Security definition is found, the UserId and Authenticator (password) information is extracted and put in the appropriate EIService properties.

If you choose to overwrite this information by either prompting the person running the application, or extracting the information from somewhere else, you cannot revert back to the definitions in the MQEI Security database until you create a new EIService object.

Note It is possible to have a UserId without an Authenticator.

Authenticator and System Authenticator data type

If you want to define you own authenticator fields to go in an EIMessage, necessary on some enterprise systems where the MQEI LSX is not able to perform authentication automatically, e.g. MQSeries native and CICS 3270 direct, you have a choice of data type.

You are recommended to use a data type of:

- AUTHENTICATOR when you do **not** want to use the authenticator property from the EIService object.
  Use this when you want to get the information from another source under your own control. You can set the Authenticator property in the EIService object.
  If you try to read a field with data type Authenticator, the MQEI LSX returns the value EIAUT_HIDDEN.
- SYSTEM_AUTHENTICATOR when you want the password filled in automatically from the Authenticator property in the EIService object when the message is sent.
  If you try to read a field with data type System Authenticator, the MQEI LSX returns the value EIAUT_HIDDEN.
  If you try to write to a field with data type System Authenticator, the MQEI LSX returns the reason code EIRC_PROTECTED_FIELD.

For example, if you want to run the CICS CESN transaction (or your own equivalent), you would define:

Field	Data type
Userid	String
Password	System_Authenticator
NewPassword	Authenticator

Note There is **no** automatic update of the information in the MQEI Security database.

Changing enterprise passwords

It is the responsibility of your users to ensure that any passwords held on the MQEI Security database are kept up to date. You can do this by:

- Instructing your users to change their password in the MQEI Security database whenever they change the corresponding password on the enterprise system.
- Providing an MQEI application that your users can run that changes the password on an enterprise system and then changes the password in the MQEI Security database to match. Changing the password on the enterprise system should be achieved by your application using the MQEI LSX. Changing the password on the MQEI Security database should be achieved by your application using the NotesDatabase, NotesDocument classes and so on, from a LotusScript program.

Note The second approach is only possible for certain types of enterprise system that allow passwords to be changed in this manner, e.g. CICS via the CESN transaction.

Notes agents

If your MQEI application is an agent running on a Domino Server, extra considerations apply if the MQEI Security database is being used.

Scheduled agent

When the agent manager runs a scheduled agent it runs under the server's name, but it is given the same ACL rights as the agent's signer (the person who last saved the agent). You can think of the agent as being a proxy for its signer.

Web application

When a Domino Web application invokes an agent it again runs under the server's name but the access rights granted to the agent depend on the release of Notes. For releases prior to 4.6, the agent is given the same ACL rights as the agent's signer. For 4.6 onwards, the user can select whether the ACL rights are that of the agent's signer or that of the Web user - either the user's authenticated identity, or as "Anonymous".

When the MQEI LSX runs under these circumstances, the only user name available to it is that of the **server** although it will have the authority of the agent signer or Web user when accessing the MQEI databases. You need to:

- Ensure the agent signer has Reader access to the MQEI Security database.
- If you are using Notes 4.6 or later and the "Run Agent as Web user" option is selected, ensure your Web users (or "Anonymous") have Reader access to the MQEI Security database.
- Ensure the server has Author access to the MQEI Security database.
- Create MQEI Security definitions (one per enterprise system) for the **server's name**.

Chapter 6 Programming for a Native MQSeries service

Whatever enterprise system your application communicates with, there are some specific aspects to consider.

Here the mapping of the MQEI properties to MQSeries attributes is covered, along with other aspects that you may need to consider when communicating with a Native MQSeries application.

MQEI Native MQSeries support

Creating an MQEI Service definition with a service type of Native MQSeries allows you to exchange messages with MQSeries enabled applications in your enterprise.

The MQEI LSX uses the MQSeries programming interface, the MQI, to access the local MQSeries queue manager. This can be a direct connection or via an MQSeries client. This gives access to your entire MQSeries network. The Notes client or Domino server using the MQEI LSX needs to reside on the same machine as the local MQSeries queue manager or MQSeries client.

The MQI is a powerful programming interface that provides a wide range of message and queuing functionality. Because of this, not all features of the MQI are available from the MQEI LSX.

The following sections explain how the MQEI API relates to MQSeries.

For more information, see the *MQSeries Application Programming Guide* manual.

Creating an MQEI Service definition

The key properties when creating an MQEI Service definition for a Native MQSeries service are ConnectionManager, OutboundConnection and InboundConnection:

- The ConnectionManager property gives the name of the MQSeries queue manager to which connection will be made. If the ConnectionManager property is left blank the default queue manager is selected.
- The OutboundConnection property gives the name of the MQSeries queue used to send messages to the MQSeries enabled application.
- The InboundConnection property gives the name of the MQSeries queue used to receive messages back from the application. At least one of InboundConnection and OutboundConnection must be specified, and they may be the same queue.

Creating MQEI Message definitions

An MQEI Message definition for a Native MQSeries service corresponds to the application data part of an MQSeries message. The source for the MQEI Message definition would typically be a copybook or header file containing the definition of the message data structure in an appropriate language.

An MQSeries message consists of control information and application data. The control information part of the message is called the MQSeries Message Descriptor (MQMD). You do not need to worry about the MQMD as it is built automatically by the MQEI LSX and should **not** be part of your MQEI Message definition.

Connecting to MQSeries

The EIService Connect method call translates to an MQI call to connect to the MQSeries queue manager given by the ConnectionManager property, followed by MQI calls to open the MQSeries queues given by the InboundConnection and OutboundConnection properties.

If the EIService UserId property contains a userid at the time of the Connect call, the OutboundConnection queue is opened in a mode that allows the MQEI LSX to pass an enterprise userid to MQSeries when a message is sent.

For more information, see "Security" later in this chapter.

Sending a message

Each EIService SendMessage method call translates to an MQI call to put an MQSeries message on the queue given by the OutboundConnection property. The EIMessage parameter fields are used to provide the application message data.

The EISendOptions parameter controls how the message is put on the queue. The EISendOptions properties of interest are MessageType, Identifier, Priority, and Delivery.

The MessageType property can be specified to supply a type indicator for the message, which may be examined by the MQSeries application that receives the message. Depending on its value, it also governs how the other EISendOptions properties are interpreted and what control information is set in the message, as follows:

- EIMT_DATAGRAM should be used if no reply is expected back. If EISendOptions MessageType has this value, the Priority property is used to set the priority of the message. The Identifier property can be specified to supply a unique identity to the message (the message identifier). If none is specified, MQSeries generates one automatically and sets it in the Identifier property when the SendMessage call has completed. The Delivery property is used to set persistence of the message.
- EIMT_REQUEST should be used when a reply is expected back. If EISendOptions Message Type has this value, the Priority, Identifier, and Delivery properties are treated as described for EIMT_DATAGRAM. Additionally the MQEI LSX sets control information to instruct the MQSeries application to send the reply message to the queue named by the EIService InboundConnection property, and to set the correlation identifier of the reply message from the message identifier of the request message.

  For more information, see "Receiving a message" later in this chapter.

- EIMT_REPLY should be used to reply to a received message that had a MessageType of EIMT_REQUEST.
  If EISendOptions MessageType has this value, the Priority and Identifier properties are ignored, and the message priority, message identifier and other control information are set by the MQEI LSX from the control information of the last received message that had a MessageType of EIMT_REQUEST.
  This enables the LotusScript program to act as an MQSeries server application in accordance with MQSeries recommendations. If no such message has been received by the EIService an MQEI reason code of EIRC_CALL_SEQUENCE_ERROR is returned. If such a message has been received, but its control information specified to send a reply message to a queue other than that named by the EIService OutboundConnection property, an MQEI reason code of EIRC_CONNECTION_UNKNOWN is returned. The value returned in the EISendOptions Identifier property when the SendMessage call has completed will be the identity that was assigned to the message.

If EISendOptions MessageType is anything else, the MQEI LSX assumes a reply may be expected, so the actions taken are the same as for EIMT_REQUEST.

The EIMessage can also provide a named format for the message from its Format Property . This named format may be examined by the MQSeries application that receives the message. An EIMessage Format of EIFMT_NONE means no named format is set in the message.

Receiving a message

Each ReceiveMessage method call translates to an MQI call to get an MQSeries message from the queue given by the InboundConnection property. The EIMessage parameter's fields are completed from the application message data assuming they match. Because the application message data in an incoming MQSeries message is just an unstructured buffer, it is only possible for the MQEI LSX to detect a mismatch if either the length of the EIMessage exceeds the length of the data, or the named format of the message does not match the Format property of the EIMessage parameter. This latter check is only made if the message actually has a named format and EIMessage Format is not EIFMT_NONE. If the EIMessage does not match the data, an MQEI reason code of EIRC_WRONG_MESSAGE is returned and a different EIMessage should be tried.

For more information, see "Variant Messages" in Chapter 4.

Whether a match occurred or not, the named format of the message is returned in the EIReceiveOptions Format property when the call has completed. When trying different EIMessages, start with the longest message first.

When calling the ReceiveMessage method to receive a message from an MQSeries application, the EIReceiveOptions Identifier property can be used to target a specific message, or can be used to receive the first message on the queue. The recommendation is to target a specific message, as this guarantees that the wrong message is not picked up in a multi-user environment. Here, the Identifier property equates to the correlation identifier of the message, so by using Identifier you are locating a message by its correlation identifier.

For more information, see "Sending a message" earlier in this chapter.

Typically, the Identifier to be used would be that from the EISendOptions object used on any corresponding SendMessage call. If no Identifier is specified in EIReceiveOptions the correlation identifier is returned in the Identifier property when the call has completed.

When calling ReceiveMessage to receive a message, the EIReceiveOptions WaitType and WaitInterval properties can be used to make the call block or not. A blocking call will wait for the specified WaitInterval. If no message arrives an MQEI reason code of EIRC_NO_MSG_AVAILABLE is returned.

The message type value from the received message is returned in the EIReceiveOptions MessageType property when the call has completed.

Disconnecting from MQSeries

An EIService Disconnect call, or destruction of the EIService, uses MQI calls to close any open queues and disconnect from the queue manager.

This does not lose outstanding messages from the queue given by the InboundConnection property. The existence of such messages is governed by the usual MQSeries rules regarding message persistence and recoverability. Messages can be recovered following a subsequent EIService Connect for the same EIService instance, or a new instance of the EIService, within the same, or a different, Notes client or Domino server agent.

Programming a conversation

A Native MQSeries conversation can consist of any combination of sends and receives of messages. The usual link between the messages that make up a conversation is the message identifier and correlation identifier, represented by the EISendOptions and EIReceiveOptions Identifier properties.

It is possible to use a single EIService object to run several conversations in parallel. An example would be a query application, where three queries needed to be made to the same MQSeries application but with different parameters. Three SendMessage calls could be made one after the other. The management of the Identifiers is the responsibility of the LotusScript program.

There is a restriction when programming parallel conversations where the LotusScript program is receiving messages with an EIReceiveOptions MessageType of EIMT_REQUEST. Only the control information of the **last** such message received is retained by the EIService object. This implies that any reply to this message **must** be sent by the LotusScript program before the next inbound message is received.

Errors

No specific action needs to be taken by the LotusScript program if a failure occurs when sending or receiving a message.

Unsupported MQSeries functions

Local commit and backout of messages is not supported by the MQEI LSX.

Security

Security here relates to any authentication performed by the MQSeries application to which messages are sent, and to any security checking performed by the queue manager.

If the MQSeries application is using its own security, it is assumed that any userid is passed to it either as part of the message control information or as part of the application message data, and that any password is passed to it as part of the application message data.

The MQEI Service definition SystemName property can be used by the MQEI LSX to read an MQEI Security definition for the Notes user from the MQEI Security database. This definition should contain the MQSeries application userid and password for the Notes user, and is used to set the EIService UserId and Authenticator properties. Alternatively the LotusScript program can prompt the user for this information and set EIService UserId and Authenticator itself.

When the EIService Connect call is made, if the EIService UserId property contains a userid, the OutboundConnection queue is opened with an additional option that permits a userid to be set in the control information of messages put on that queue. The userid in the control information is set from EIService UserId when each message is sent. This userid may also be used by MQSeries to check queue access authority, if MQSeries is configured for this.

When the EIService Connect call is made, if the EIService UserId property is empty, the OutboundConnection queue is opened as normal, and the userid in the control information is left to be set by the queue manager according to MQSeries rules. If this is the case, any MQSeries application userid must be a field in the EIMessage.

It is the responsibility of the LotusScript program to set any userid and authenticator fields of the EIMessage representing the MQSeries message. A userid field may be set by the program copying the EIService UserId property. An authenticator field will be set automatically by the MQEI LSX from the EIService Authenticator property during SendMessage if it is defined with a data type of System_Authenticator, or may be set explicitly by the program if it is defined with a data type of Authenticator.

Data conversion

It is recommended that data conversion of MQSeries messages is performed by the MQEI LSX.

This is achieved by specifying the CharacterSet and Encoding properties in the MQEI Service definition. An outbound message is converted to the specified character set and numeric encoding from the local settings specified in the MQEI initialization file (mqei.ini). An inbound message is converted to the local settings from the character set and encoding specified in its control information.

Alternatively outbound data conversion can be carried out under MQSeries, by the MQSeries system programmer providing MQSeries data conversion exits whose names match the Format properties of your EIMessage objects. If this approach is adopted, then any outbound MQEI LSX data conversion can be turned off by specifying a CharacterSet and an Encoding in the MQEI Service definition that match those in the MQEI initialization file.

Note The MQEI LSX will still perform any inbound data conversion based on the message control information settings.

If you are familiar with the MQI

This section may prove helpful if you are familiar with the MQSeries MQI.

Mapping of MQEI properties to a Native MQSeries service

The following lists the properties of the EIService, EIMessage, EISendOptions and EIReceiveOptions classes that are relevant to the MQI.

MQEI Class and Property	**MQI Equivalent**
EIService CharacterSet	CodedCharSetId (MQMD)
EIService ConnectionManager	Queue Manager Name (MQCONN)
EIService Encoding	Encoding (MQMD)
EIService InboundConnection	Queue ObjectName (MQOPEN MQOD)
EIService SystemName	No equivalent
EIService OutboundConnection	Queue ObjectName (MQOPEN MQOD)
EIService UserId	UserIdentifier (MQMD) Note 1
EIService Authenticator	No equivalent
EIMessage Format	Format (MQMD)
EISendOptions Delivery	Persistence (MQMD) Note 2
EISendOptions Identifier	MsgId and CorrelId (MQMD) Note 2
EISendOptions MessageType	MsgType (MQMD)
EISendOptions Priority	Priority (MQMD) Note 2
EIReceiveOptions Identifier	CorrelId (MQMD)
EIReceiveOptions MessageType	MsgType (MQMD)
EIReceiveOptions Format	Format (MQMD)
EIReceiveOptions WaitInterval	WaitInterval (MQGMO)
EIReceiveOptions WaitType	Options (MQGMO) Note 3

Note 1: Only if OutboundConnection opened with set identity context authority.

Note 2: Settings dependent on EISendOptions MessageType

Note 3: MQGMO_WAIT or MQGMO_NOWAIT

The MQSeries MQI calls used are MQCONN, MQOPEN, MQPUT, MQGET, MQCLOSE, and MQDISC.

What happens during a Connect

The OutboundConnection queue is opened with options MQOO_OUTPUT and MQOO_FAIL_IF_QUIESCING.

The InboundConnection queue is opened with options MQOO_INPUT_AS_Q_DEF and MQOO_FAIL_IF_QUIESCING. If the MQOPEN for the InboundConnection queue fails with MQRC_OPTION_NOT_VALID_FOR_TYPE the MQOPEN is automatically retried with MQOO_INPUT_EXCLUSIVE.

If the EIService UserId property contains a userid at the time of the Connect call, the OutboundConnection queue is additionally opened with MQOO_SET_IDENTITY_CONTEXT.

What happens during a SendMessage

The EISendOptions MessageType property governs how many of the MQMD fields are set, as follows:

- If EISendOptions MessageType is EIMT_DATAGRAM, MQMD MsgType is set to MQMT_DATAGRAM and the following occurs:
  - MQMD Priority is set from EISendOptions Priority, a value of EIPRI_DEFAULT being translated to MQPRI_PRIORITY_AS_Q_DEF.
  - MQMD Persistence is set from EISendOptionsDelivery. EIDEL_ASSURED translates to MQPER_PERSISTENT, EIDEL_EXPRESS translates to MQPER_NON_PERSISTENT, and EIDEL_DEFAULT translates to MQ_PERSISTENCE_AS_Q_DEF.
  - MQMD Report is set to MQRO_NEW_MSG_ID and MQRO_COPY_MSG_ID_TO_CORREL_ID.
  - MQMD MsgId and CorrelId are set from EISendOptions Identifier if specified. If no Identifier is specified, they are set to MQMI_NONE and MQCI_NONE respectively.
- If EISendOptions MessageType is EIMT_REQUEST:
  - MQMD MsgType is set to MQMT_REQUEST
  - MQMD ReplyToQ is set from EIService InboundConnection
  - MQMD ReplyToQMgr is set to blanks.
  - Other actions are the same as for EIMT_DATAGRAM,
- If EISendOptions MessageType is EIMT_REPLY:
  - MQMD MsgType is set to MQMT_REPLY

    - The MQMD Priority, Persistence, MsgId, CorrelId, Report, ReplyToQ and ReplyToQMgr are set from the MQMD of the last received message that had an MQMD MsgType of MQMT_REQUEST.
- If EISendOptions MessageType is anything else, MQMD MsgType is set to EISendOptions MessageType without translation, and the actions taken are the same as for EIMT_REQUEST.

MQMD Expiry is always set to MQEI_UNLIMITED.

PutApplType, PutApplName, PutDate, PutTime, ApplOriginData and BackoutCount are not set so either default or are set by the queue manager.

If the queue was opened with MQOO_SET_IDENTITY_CONTEXT:

- All SendMessage calls specify MQPMO_SET_IDENTITY_CONTEXT
- MQMD UserIdentifier is set from EIService UserId
- MQMD AccountingToken is set to MQACT_NONE
- MQMD ApplIdentityData is set to blanks.

Otherwise MQMD UserIdentifier, AccountingToken and ApplIdentityData are left to be set by the queue manager according to MQSeries rules.

The EIMessage Format property is used to set the MQMD Format. An EIMessage Format of EIFMT_NONE translates to MQFMT_NONE. An EIMessage Format of EIFMT_STRING translates to MQFMT_STRING. Any other value is not translated.

Chapter 7 Programming for an IMS via MQSeries service

Whatever enterprise system your application communicates with, there are some specific aspects to consider.

Here the mapping of the MQEI properties to MQSeries and IMS attributes is covered, along with other aspects that you may need to consider when communicating with an IMS application via MQSeries.

IMS via MQSeries support

Creating an MQEI Service definition with a service type of IMS via MQSeries allows you to run IMS transactions on an IMS/ESA server. An IMS transaction is identified by a transid of up to eight characters.

The MQEI LSX uses MQSeries and the MQSeries-IMS bridge as the communication mechanism.

The MQSeries-IMS bridge allows an application program, in this case the MQEI LSX, to run an IMS transaction on an IMS/ESA server via the MQI. The MQEI LSX uses the MQI to access a local MQSeries queue manager, either directly or via an MQSeries client.

The Notes client or Domino server using the MQEI LSX needs to reside on the same machine as the local MQSeries queue manager or MQSeries client.

An MQSeries for MVS/ESA queue manager is required on the same MVS/ESA system as the IMS server. The MQSeries-IMS bridge, which must also be running on the same MVS/ESA system, runs the IMS transaction, and returns results via the MQI.

For more information, see the *MQSeries Application Programming Guide* manual.

The EIService object created from an MQEI Service definition of this type would typically represent a suite of related transactions on the IMS server.

Creating an MQEI Service definition

The key properties when creating an MQEI Service definition for an IMS via MQSeries service are ConnectionManager, OutboundConnection, InboundConnection, ServiceStep and ServiceContext:

- The ConnectionManager propertygives the name of the local MQSeries queue manager to which connection will be made. If the ConnectionManager property is left blank the default queue manager is selected.
- The OutboundConnection property gives the name of the MQSeries queue used to send messages to the MQSeries-IMS bridge.
- The InboundConnection property gives the name of the MQSeries queue used to receive reply messages back from the MQSeries-IMS bridge. Both InboundConnection and OutboundConnection must be specified, and they may not be the same queue.
- The ServiceStep property gives the name of the IMS transaction itself. It is possible to invoke several different IMS transactions using a single EIService object, so the ServiceStep property should initially be set to the first such transaction.
- The ServiceContext property optionally gives the name of a logical terminal (LTERM) to pass to the IMS transaction.

Creating MQEI Message definitions

An MQEI Message definition for an IMS via MQSeries service corresponds to the IMS message data passed to the IMS transaction when it issues a GET UNIQUE (GU) to the IOPCB, or sent as output by the IMS transaction when it issues an INSERT (ISRT) to the IOPCB. This is transmitted to the MQSeries-IMS bridge as the application message data part of an MQSeries message. The application message data can be either a single-segment or a multi-segment IMS message, the Segment property of each field in the MQEI Message definition giving the segment that the field resides in. The source for the MQEI Message definition would typically be an MFS map source file or the equivalent application data structure used by the IMS transaction (typically a copybook).

An MQSeries message consists of control information and application data. The control information part of the message is called the MQSeries Message Descriptor (MQMD). Additionally messages for the MQSeries-IMS bridge have some extra control information in a bridge header called the MQIIH, an embedded IMS trancode, and embedded IMS message <LLZZ> segment indicators. You do not need to worry about any of these, as they are all built automatically by the MQEI LSX and should **not** be part of your MQEI Message definition.

Connecting to MQSeries

The EIService Connect method call translates to an MQI call to connect to the MQSeries queue manager given by the ConnectionManager property, followed by MQI calls to open the MQSeries queues given by the InboundConnection and OutboundConnection properties.

If the EIService UserId property contains a userid at the time of the Connect call, the OutboundConnection queue is opened in a mode that allows the MQEI LSX to pass an IMS userid to the MQSeries-IMS bridge when a message is sent.

For more information, see "Security" later in this chapter.

Sending a message

Each EIService SendMessage method call translates to an MQI call to put an MQSeries message on the MQSeries queue given by the OutboundConnection property. The EIMessage parameter is used to provide the application message data part of the message, that is, the IMS message to be passed as input to the IMS transaction. The name of the IMS transaction is given by the ServiceStep property. This is automatically added to the MQSeries message by the MQEI LSX, as is the MQSeries-IMS bridge header, and any IMS message <LLZZ> segment indicators.

The EISendOptions parameter controls how the message is put on the queue. The EISendOptions properties of interest are Identifier and Priority:

- Priority property is used to set the priority of the message
- Identifier property can be specified to supply a unique identity to the message (the message identifier) and its associated reply. If no Identifier is specified, MQSeries generates one automatically and sets it in the Identifier property when the SendMessage call has completed.
- Delivery property is used to set both the persistence of the message and the IMS commit mode. Note that EIDEL_ASSURED must not be used if the IMS transaction is conversational or Fast Path.

Note The EISendOptions MessageType property is ignored by the MQEI LSX and is overridden internally to EIMT_REQUEST.

The EIMessage can also provide a named format for the IMS message from its Format property. This is used to set the name of the MFS map that gets passed to the IMS application to represent the IMS MOD. An EIMessage Format of EIFMT_NONE means no MFS map name is set in the message.

Receiving a message

Each ReceiveMessage method call translates to an MQI call to get an MQSeries message from the MQSeries reply queue given by the InboundConnection property .

The EIMessage parameter's fields are completed from the application message data, that is, the IMS message output by the IMS transaction, assuming they match. It is possible for the MQEI LSX to detect a mismatch if either the length of the EIMessage exceeds the length of the message data, or the EIMessage does not match the message segments, or the name of the MFS map representing the IMS MOD does not match the Format property of the EIMessage parameter. This latter check is only made if the message actually contains an MFS map name and EIMessage Format is not EIFMT_NONE. If the EIMessage does not match the data, an MQEI reason code of EIRC_WRONG_MESSAGE is returned and a different EIMessage should be tried.

For more information, see "Variant Messages" in Chapter 4.

Whether a match occurred or not, the MFS map name is returned in the EIReceiveOptions Format property when the call has completed.

When calling the ReceiveMessage method to receive a reply from the MQSeries-IMS bridge, the EIReceiveOptions Identifier property can be used to target a specific message, or can be used to receive the first message on the queue. The recommendation is to target a specific message, as this guarantees that the wrong message is not picked up in a multi-user environment. The Identifier to be used should be that from the EISendOptions object used on the corresponding SendMessage call. If no Identifier is specified in EIReceiveOptions, the message identifier of the message is returned in the Identifier property when the call has completed.

When calling ReceiveMessage to receive a reply, the EIReceiveOptions WaitType and WaitInterval properties can be used to make the call block or not. A blocking call waits for the specified WaitInterval. If no message arrives an MQEI reason code of EIRC_NO_MSG_AVAILABLE is returned.

The message type value from the received message is returned in the EIReceiveOptions MessageType property when the call has completed.

Disconnecting from MQSeries

An EIService Disconnect call, or destruction of the EIService, uses MQI calls to close any open queues and disconnect from the queue manager.

This does not lose outstanding messages from the MQSeries-IMS bridge reply queue given by the InboundConnection property. The existence of such messages is governed by the usual MQSeries rules regarding message persistence and recoverability. Messages can be recovered following a subsequent EIService Connect for the same EIService instance, or a new instance of the EIService, within the same, or a different, Notes client or Domino server agent.

Programming a conversation

An IMS conversation can consist of running either a single IMS transaction, or a suite of conversational IMS transactions.

During a SendMessage call, the IMS trancode is passed to IMS in the MQSeries message only at the beginning of a conversation. The LotusScript program can vary the IMS trancode by changing the ServiceStep property prior to making the SendMessage call. The trancode is not normally passed to IMS if in the middle of a conversation. In this case, the ServiceStep property is not used. The only exception to this is if the LotusScript program wishes to terminate the conversation prematurely, in which case the LotusScript program should set the ServiceStep property to /EXIT.

It is not possible to use a single EIService object to run several separate conversations in parallel. You must create an EIService for each parallel conversation you wish to hold.

A conversation can persist across destruction and re-creation of the EIService object only if a ReceiveMessage call is the first call made from the new EIService object after the Connect call. This is because the transaction instance id assigned to the conversation by IMS is lost by the MQEI LSX when the EIService is destroyed, and can only be recreated from an inbound reply message.

Errors

No specific action needs to be taken by the LotusScript program if a failure occurs when sending or receiving a message.

Unsupported MQSeries functions

Local commit and backout of messages is not supported by the MQEI LSX.

Security

The MQEI Service definition SystemName property can be used by the MQEI LSX to read an MQEI Security definition for the Notes user from the MQEI Security database. This definition should contain the IMS userid and password for the Notes user, and is used to set the EIService UserId and Authenticator properties. Alternatively the LotusScript program can prompt the user for this information and set EIService UserId and Authenticator itself.

If security is being used on the IMS server, the IMS userid and password need to be passed by the MQEI LSX to the MQSeries-IMS bridge on each SendMessage call so that the MQSeries-IMS bridge can authenticate the Notes user. For this to happen, the EIService UserId property **must** contain a userid when the EIService Connect call is made. If so, the OutboundConnection queue is opened with an additional option that permits a userid to be set in the control information of messages put on that queue. The userid in the control information is set from EIService UserId when each message is sent. This UserId may also be used by MQSeries to check queue access authority, if MQSeries is configured for this.

Depending on the setup of your MQSeries-IMS bridge, it may only be necessary to pass the IMS password on the first SendMessage call you make to IMS. If this is the case, and you wish to miminize the number of times the password is transmitted, the LotusScript program should set the EIService Authenticator property to blanks after the first SendMessage call.

If the EIService UserId property is empty when the EIService Connect call is made, the OutboundConnection queue is opened as normal, and the userid in the control information is left to be set by the queue manager according to MQSeries rules. This approach should only be used if security is not being used by the IMS server.

If authentication by the MQSeries-IMS bridge fails, no reply message will be returned. Failure to receive an expected reply message could therefore be an indication that authentication failed. If the IMS userid does not have sufficient authority to access resources used by the IMS transaction, a reply message consisting of a DFS1292E error message will be returned. In both cases the original message sent to IMS will be placed on the dead letter queue if one is defined.

Data conversion

Data conversion of MQSeries-IMS bridge messages **must** be performed by the MQEI LSX. This is because the EIMessage Format property is used to specify an IMS MFS map name and not a named message format (which is the control information used to locate an MQSeries data conversion exit).

Data conversion by the MQEI LSX is achieved by specifying the CharacterSet and Encoding properties in the MQEI Service definition. An outbound message is converted to the specified character set and numeric encoding from the local settings specified in the MQEI initialization file (mqei.ini). An inbound reply is converted to the local settings from the character set and encoding specified in its control information.

If you are familiar with the MQI

This section may prove helpful if you are familiar with the MQSeries MQI.

Mapping of MQEI properties to an IMS via MQSeries service

The following lists the properties of the EIService, EIMessage, EISendOptions and EIReceiveOptions classes that are relevant to the MQI.

MQEI Class and Property	MQI Equivalent
EIService ServiceStep	IMS trancode (in message data)
EIService Context	LTermOverride (MQIIH)
EIService CharacterSet	CodedCharSetId (MQMD)
EIService ConnectionManager	Queue Manager Name (MQCONN)
EIService Encoding	Encoding (MQMD)
EIService InboundConnection	Queue ObjectName (MQOPEN MQOD)
EIService OutboundConnection	Queue ObjectName (MQOPEN MQOD)
EIService SystemName	No equivalent
EIService UserId	UserIdentifier (MQMD) Note 1
EIService Authenticator	Authenticator (MQIIH)
EIMessage Format	MFSMapName (MQIIH)
EISendOptions Delivery	Persistence (MQMD) and CommitMode (MQIIH)
EISendOptions Identifier	MsgId and CorrelId (MQMD)
EISendOptions Priority	Priority (MQMD)
EIReceiveOptions Format	MFSMapName (MQIIH)
EIReceiveOptions Identifier	CorrelId (MQMD)
EIReceiveOptions MessageType	MsgType (MQMD)
EIReceiveOptions WaitInterval	WaitInterval (MQGMO)
EIReceiveOptions WaitType	Options (MQGMO)

Note 1: Only if OutboundConnection opened with set identity context authority.

The MQSeries MQI calls used are MQCONN, MQOPEN, MQPUT, MQGET, MQCLOSE, and MQDISC.

What happens during a Connect

The OutboundConnection queue is opened with options MQOO_OUTPUT and MQOO_FAIL_IF_QUIESCING.

The InboundConnection queue is opened with options MQOO_INPUT_AS_Q_DEF and MQOO_FAIL_IF_QUIESCING.

If the MQOPEN for the InboundConnection queue fails with MQRC_OPTION_NOT_VALID_FOR_TYPE the MQOPEN is automatically retried with MQOO_INPUT_EXCLUSIVE.

If the EIService UserId property contains a userid at the time of the Connect call, the OutboundConnection queue is additionally opened with MQOO_SET_IDENTITY_CONTEXT.

What happens during a SendMessage

MQMD MsgId and CorrelId are set from EISendOptions Identifier if specified. If no Identifier is specified, they are set to MQMI_NONE and MQCI_NONE respectively.

MQMD Priority is set from EISendOptions Priority, a value of EIPRI_DEFAULT being translated to MQPRI_PRIORITY_AS_Q_DEF.

MQMD Persistence and MQIIH CommitMode are set from EISendOptions Delivery.
EIDEL_ASSURED translates to MQPER_PERSISTENT and MQCIM_COMMIT_THEN_SEND.
EIDEL_EXPRESS translates to MQPER_NON_PERSISTENT and MQCIM_SEND_THEN_COMMIT.
EIDEL_DEFAULT translates to MQPER_PERSISTENT_AS_Q_DEF and MQCIM_SEND_THEN_COMMIT.

The following MQMD fields can not be influenced by the LotusScript program:

- MQMD MessageType is always set to MQMT_REQUEST.
- MQMD Report is always set to MQRO_NEW_MSG_ID and MQRO_COPY_MSG_ID_TO_CORREL_ID.
- MQMD ReplyToQ is always set from EIService InboundConnection
- MQMD ReplyToQMgr is always set to blanks.
- MQMD Expiry is always set to MQEI_UNLIMITED.
- MQMD Format is always set to MQFMT_IMS to indicate that an MQIIH bridge header follows.

- PutApplType, PutApplName, PutDate, PutTime, ApplOriginData and BackoutCount are not set so either default or are set by the queue manager.

If the queue was opened with MQOO_SET_IDENTITY_CONTEXT:

- All SendMessage calls specify MQPMO_SET_IDENTITY_CONTEXT
- MQMD UserIdentifier is set from EIService UserId
- MQMD AccountingToken is set to MQACT_NONE
- MQMD ApplIdentityData is set to blanks.

Otherwise MQMD UserIdentifier, AccountingToken and ApplIdentityData are left to be set by the queue manager according to MQSeries rules.

The EIMessage Format property is used to set the MQIIH MFSMapName that gets placed in the IOPCB to represent the IMS MOD. An EIMessage Format of EIFMT_NONE means no MFS map name is set. An EIMessage Format of EIFMT_STRING translates to MQFMT_IMS_VAR_STRING. Any other value is assumed to be a real MFS map name and is not translated.

The MQIIH Format and ReplyToFormat are always set to MQFMT_NONE. This means that data conversion of the IMS message data will not take place on the MQSeries for MVS/ESA queue manager.

The following MQIIH field can not be influenced by the LotusScript program.

- MQIIH SecurityScope is always set to MQISS_CHECK.

Chapter 8 Programming for a CICS DPL via MQSeries service

Whatever enterprise system your application communicates with, there are some specific aspects to consider.

Here the mapping of the MQEI properties to MQSeries and CICS attributes is covered, along with other aspects that you may need to consider when communicating with a CICS DPL application via MQSeries.

CICS DPL via MQSeries support

Creating an MQEI Service definition with a service type of CICS DPL via MQSeries allows you to run CICS Distributed Program Link (DPL) programs on a CICS for MVS/ESA server. A CICS DPL program is a CICS program that may be invoked using the CICS command EXEC CICS LINK but may not use CICS terminal or syncpoint facilities.

The MQEI LSX uses MQSeries and the MQSeries-CICS/ESA DPL bridge as the communication mechanism. The MQSeries-CICS/ESA DPL bridge allows an application program, in this case the MQEI LSX, to invoke a CICS DPL program on a CICS for MVS/ESA server via the MQI. The MQEI LSX uses the MQI to access a local MQSeries queue manager, either directly or via an MQSeries client. The Notes client or Domino server using the MQEI LSX needs to reside on the same machine as the local MQSeries queue manager or MQSeries client.

An MQSeries for MVS/ESA queue manager is required on the same MVS/ESA system as the CICS server. The MQSeries-CICS/ESA DPL bridge, which must be running on a CICS server in the same MVS/ESA system, invokes the DPL program, which may reside on any connected CICS server, and returns results via the MQI.

For more information, see the *MQSeries Application Programming Guide* manual and the *MQSeries-CICS/ESA DPL bridge User Guide*.

Creating an MQEI Service definition

The key properties when creating an MQEI Service definition for a CICS DPL via MQSeries service are ConnectionManager, OutboundConnection, InboundConnection and ServiceStep.

- The ConnectionManager property gives the name of the local MQSeries queue manager to which connection will be made. If the ConnectionManager property is left blank the default queue manager is selected.
- The OutboundConnection property gives the name of the MQSeries queue used to send messages to the MQSeries-CICS/ESA DPL bridge.
- The InboundConnection property gives the name of the MQSeries queue used to receive reply messages back from the MQSeries-CICS/ESA DPL bridge. Both InboundConnection and OutboundConnection must be specified, and they may not be the same queue.
- The ServiceStep property gives the name of the DPL program itself. It is possible to invoke several different DPL programs using a single EIService object, so the ServiceStep property should initially be set to the first such program.
- The ServiceContext property gives the CICS transaction identifier that the DPL program will run under.

Creating MQEI Message definitions

An MQEI Message definition for a CICS DPL via MQSeries service corresponds to the CICS COMMAREA passed as a parameter to the DPL program by the EXEC CICS LINK command. This is transmitted to the MQSeries-CICS/ESA DPL bridge as the application message data part of an MQSeries message. The source for the MQEI Message definition would typically be a copybook or header file containing the definition of the COMMAREA data structure in an appropriate language.

Often the format of the COMMAREA is the same when passing data into a given DPL program as it is when receiving data back. In this case a single MQEI Message definition will be needed per DPL program. However, sometimes the format of the COMMAREA varies between input and output, so two MQEI Message definitions will be needed, one for the input format and one for the output format. If the output format is longer than the input format, you must pad the MQEI Message definition for the input format so its length matches the output format length, otherwise a failure may occur on the CICS server.

An MQSeries message consists of control information and application data. The control information part of the message is called the MQSeries Message Descriptor (MQMD). Additionally messages for the MQSeries-CICS/ESA DPL bridge have some extra control information in a bridge header called the MQCIH, and an embedded DPL program name. You do not need to worry about any of these, as they are all built automatically by the MQEI LSX and should **not** be part of your MQEI Message definition.

Connecting to MQSeries

The EIService Connect method call translates to an MQI call to connect to the MQSeries queue manager given by the ConnectionManager property, followed by MQI calls to open the MQSeries queues given by the InboundConnection and OutboundConnection properties.

If the EIService UserId property contains a userid at the time of the Connect call, the OutboundConnection queue is opened in a mode that allows the MQEI LSX to pass a CICS userid to the MQSeries-CICS/ESA DPL bridge when a message is sent.

For more information see "Security" later in this chapter.

Sending a message

Each EIService SendMessage method call translates to an MQI call to put an MQSeries message on the MQSeries queue given by the OutboundConnection property. The EIMessage parameter's fields are used to provide the application message data part of the message, that is, the COMMAREA to be passed as input to the DPL program. The name of the DPL program is given by the ServiceStep property. This is automatically added to the message by the MQEI LSX, as is the MQSeries-CICS/ESA DPL bridge header. The CICS transid that the DPL program will run under is given by theServiceContext property (if this is the start of a new CICS logical unit of work(UOW)).

The EISendOptions parameter controls how the message is put on the queue and how the DPL program is invoked. The EISendOptions properties of interest are UnitOfWork and Identifier:

- Identifier property can be specified to supply a unique identity to the message (the message identifier) and its associated reply. If no Identifier is specified, MQSeries generates one automatically and sets it in the Identifier property when the SendMessage call has completed.
- Delivery property is used to set the persistence of the message.
- UnitOfWork property specifies how this DPL program relates to the CICS logical unit of work (UOW) that it will run under.

For more information see "Programming a conversation" later in this chapter.

Note that the EISendOptions MessageType and Priority properties are ignored. The MQEI LSX overrides MessageType internally to EIMT_REQUEST. Priority ordered queues should not be used with the MQSeries-CICS/ESA DPL bridge.

Note EIMessage Format property is ignored

For more information, see "Data Conversion" later in this chapter.

Receiving a message

Each SendMessage method call generates exactly one reply from the MQSeries-CICS/ESA DPL bridge. A ReceiveMessage method call translates to an MQI call to get an MQSeries message from the reply queue given by the InboundConnection property. The EIMessage parameter's fields are completed from the application message data, that is, the COMMAREA, assuming they match. Because a COMMAREA is just an unstructured buffer, it is only possible for the MQEI LSX to detect a mismatch if the length of the EIMessage exceeds the length of the data. If the EIMessage does not exactly match the data, an MQEI reason code of EIRC_WRONG_MESSAGE or EIRC_INSUFFICIENT_DATA is returned and a different EIMessage should be tried.

For more information see "Variant Messages" in Chapter 4.

When trying different EIMessages, start with the longest message first.

When calling the ReceiveMessage method to receive a reply from the MQSeries-CICS/ESA DPL bridge, the EIReceiveOptions Identifier property can be used to target a specific message, or can be used to receive the first message on the queue. The recommendation is to target a specific message, as this guarantees that the wrong message is not picked up in a multi-user environment. The Identifier to be used should be that from the EISendOptions object used on the corresponding SendMessage call. If no Identifier is specified in EIReceiveOptions the message identifier of the message is returned in the Identifier property when the call has completed.

When calling ReceiveMessage to receive a reply, the EIReceiveOptions WaitType and WaitInterval properties can be used to make the call block or not. A blocking call will wait for the specified WaitInterval. If no message arrives an MQEI reason code of EIRC_NO_MSG_AVAILABLE is returned.

The message type value from the received message is returned in the EIReceiveOptions MessageType property when the call has completed.

Disconnecting from MQSeries

An EIService Disconnect call l, or destruction of the EIService, uses MQI calls to close any open queues and disconnect from the queue manager.

This does not lose outstanding messages from the MQSeries-CICS/ESA DPL bridge reply queue given by the InboundConnection property. The existence of such messages is governed by the usual MQSeries rules regarding message persistence and recoverability. Messages can be recovered following a subsequent EIService Connect for the same EIService instance, or a new instance of the EIService, within the same, or a different, Notes client or Domino server agent.

Programming a conversation

A CICS DPL via MQSeries conversation can consist of a single call to a DPL program, or several calls to the same DPL program, or several calls to different DPL programs. Varying the DPL program name may be achieved by changing the EIService ServiceStep property from the LotusScript program prior to making a SendMessage call. You have the choice to run each DPL program in its own CICS logical unit of work (UOW) or to group programs together in an extended UOW. This is controlled by the EISendOptions UnitOfWork property.

When running an extended UOW, the EISendOptions Identifier property is used as the link between the different calls that make up the UOW. On the first SendMessage call of the UOW, the LotusScript program can either provide its own, unique, Identifier or let the MQSeries queue manager provide a unique one on its behalf. If this Identifier is not unique, an MQEI reason code of EIRC_IDENTIFIER_ERROR will be returned when the ReceiveMessage call is made. This same Identifier **must** be present on all SendMessage calls that comprise the UOW. Failure to do this will result in the MQSeries-CICS/ESA DPL bridge not finding messages and timing out the unit of work.

When running an extended UOW, it is only when the last reply message is received that the LotusScript program can be sure that the UOW has been committed or backed out. This last reply message is distinguished by having an EIReceiveOptions MessageType of EIMT_REPLY, whereas intermediate reply messages will have a MessageType of EIMT_REQUEST.

It is possible to use a single EIService object to run several separate UOWs in parallel. An example would be a query application, where three queries needed to be made to the same DPL program but with different parameters. Three SendMessage calls could be made one after the other. The management of the Identifiers is the responsibility of the LotusScript program.

Errors

No specific action needs to be taken by the LotusScript program if a failure occurs when sending or receiving a message.

If a failure has occurred in the MQSeries-CICS/ESA DPL bridge, or there was a problem with a message it receives, the CICS bridge task will have backed out any changes made to recoverable resources on the CICS server.

If a failure has occurred in MQSeries, or there was a problem with a message, which prevents an expected message arriving at the bridge, the CICS bridge task will timeout the unit of work and again back out changes to recoverable resources.

Unsupported MQSeries functions

Local commit and backout of messages is not supported by the MQEI LSX.

Security

The MQEI Service definition SystemName property can be used by the MQEI LSX to read an MQEI Security definition for the Notes user from the MQEI Security database. This definition should contain the CICS userid and password for the Notes user, and is used to set the EIService UserId and Authenticator properties. Alternatively the LotusScript program can prompt the user for this information and set EIService UserId and Authenticator itself.

If security is being used on the CICS server, the CICS userid and password need to be passed by the MQEI LSX to the MQSeries-CICS/ESA DPL bridge on each SendMessage call so that CICS can authenticate the Notes user.
For this to happen, the EIService UserId property **must** contain a userid when the EIService Connect call is made. If so, the OutboundConnection queue is opened with an additional option that permits a userid to be set in the control information of messages put on that queue. The userid in the control information is set from EIService UserId when each message is sent. This UserId may also be used by MQSeries to check queue access authority, if MQSeries is configured for this.

If the EIService UserId property is empty when the EIService Connect call is made, the OutboundConnection queue is opened as normal, and the userid in the control information is left to be set by the queue manager according to MQSeries rules. This approach should only be used if security is not being used by the CICS server.

If authentication fails, or if the CICS userid does not have sufficient authority to access resources used by the DPL program, an MQEI reason code of EIRC_SECURITY_FAILURE will be returned by the ReceiveMessage call.

Data conversion

Data conversion of MQSeries-CICS/ESA DPL bridge messages **must** be performed by the MQEI LSX. This is because there is no facility on MQSeries queue managers to data convert the MQCIH bridge header, which also prevents the data conversion of the COMMAREA.

Data conversion by the MQEI LSX is achieved by specifying the CharacterSet and Encoding properties in the MQEI Service definition. An outbound message is converted to the specified character set and numeric encoding from the local settings specified in the MQEI initialization file (mqei.ini). An inbound reply is converted to the local settings from the character set and encoding specified in its control information.

Because data conversion must not take place on the MQSeries queue managers, the EIMessage Format property is not used to set a named message format in an outbound message. Otherwise an MQSeries data conversion exit may be called which would fail for the reason given above.

If you are familiar with the MQI

This section may prove helpful if you are familiar with the MQSeries MQI.

Mapping of MQEI properties to a CICS DPL via MQSeries service

The following lists the properties of the EIService, EIMessage, EISendOptions and EIReceiveOptions classes that are relevant to the MQI.

MQEI Class and Property	MQI Equivalent
EIService AbendCode	AbendCode (MQCIH)
EIService ServiceStep	DPL program name (in message data)
EIService ServiceContext	TransactionId (MQCIH)
EIService CharacterSet	CodedCharSetId (MQMD)
EIService ConnectionManager	Queue Manager Name (MQCONN)
EIService Encoding	Encoding (MQMD)
EIService InboundConnection	Queue ObjectName (MQOPEN MQOD)
EIService OutboundConnection	Queue ObjectName (MQOPEN MQOD)
EIService SystemName	No equivalent
EIService UserId	UserIdentifier (MQMD) Note 1
EIService Authenticator	Authenticator (MQCIH)
EISendOptions Delivery	Persistence (MQMD)
EISendOptions Identifier	MsgId and CorrelId (MQMD)
EISendOptions UnitOfWork	UOWControl (MQCIH)
EIReceiveOptions Identifier	CorrelId (MQMD)
EIReceiveOptions MessageType	MsgType (MQMD)
EIReceiveOptions WaitInterval	WaitInterval (MQGMO)
EIReceiveOptions WaitType	Options (MQGMO)

Note 1: Only if OutboundConnection opened with set identity context authority.

The MQSeries MQI calls used are MQCONN, MQOPEN, MQPUT, MQGET, MQCLOSE, and MQDISC.

What happens during a Connect

The OutboundConnection queue is opened with options MQOO_OUTPUT and MQOO_FAIL_IF_QUIESCING.

The InboundConnection queue is opened with options MQOO_INPUT_AS_Q_DEF and MQOO_FAIL_IF_QUIESCING. If the MQOPEN for the InboundConnection queue fails with MQRC_OPTION_NOT_VALID_FOR_TYPE the MQOPEN is automatically retried with MQOO_INPUT_EXCLUSIVE.

If the EIService UserId property contains a userid at the time of the Connect call, the OutboundConnection queue is additionally opened with MQOO_SET_IDENTITY_CONTEXT.

What happens during a SendMessage

MQMD Persistence is set from EISendOptions Delivery.
EIDEL_ASSURED translates to MQPER_PERSISTENT.
EIDEL_EXPRESS translates to MQPER_NON_PERSISTENT.
EIDEL_DEFAULT translates to MQPER_PERSISTENT_AS_Q_DEF.

If EISendOptions UnitOfWork is EIUOW_FIRST or EIUOW_ONLY, the following occurs:

- MQMD MsgId is set from EISendOptions Identifier if specified. If no Identifier is specified, it is set to MQMI_NONE which causes MQSeries to generate a unique MsgId for the message.
- MQMD CorrelId is set to MQCI_NEW_SESSION.

If EISendOptions UnitOfWork is anything else:

- MQMD MsgId is set from EISendOptions Identifier if specified. If no Identifier is specified, it is set to MQMI_NONE which causes MQSeries to generate a unique MsgId for the message
- MQMD CorrelId is set from the EISendOptions Identifier property in the same manner as MQMD MsgId.

The following MQMD fields can not be influenced by the LotusScript program:

- MQMD MessageType is always set to MQMT_REQUEST.
- MQMD Priority is always set to MQPRI_PRIORITY_AS_Q_DEF as priority is not applicable for the MQSeries-CICS/ESA DPL bridge.
- MQMD Report is always set to MQRO_NEW_MSG_ID and MQRO_COPY_MSG_ID_TO_CORREL_ID.
- MQMD ReplyToQ is always set from EIService InboundConnection
- MQMD ReplyToQMgr is always set to blanks.

- MQMD Expiry is always set to MQEI_UNLIMITED.
- MQMD Format is always set to MQFMT_CICS to indicate that an MQCIH bridge header follows.
- PutApplType, PutApplName, PutDate, PutTime, ApplOriginData and BackoutCount are not set so either default or are set by the queue manager.

If the queue was opened with MQOO_SET_IDENTITY_CONTEXT:

- All SendMessage calls specify MQPMO_SET_IDENTITY_CONTEXT
- MQMD UserIdentifier is set from EIService UserId
- MQMD AccountingToken is set to MQACT_NONE
- MQMD ApplIdentityData is set to blanks.

Otherwise MQMD UserIdentifier, AccountingToken and ApplIdentityData are left to be set by the queue manager according to MQSeries rules.

Note EIMessage Format property is ignored and MQCIH Format and ReplyToFormat are always set to MQFMT_NONE. This ensures that data conversion of the COMMAREA will not take place on the MQSeries for MVS/ESA queue manager

Chapter 9 Programming for a CICS DPL direct service

Whatever enterprise system your application communicates with, there are some specific aspects to consider.

Here the mapping of the MQEI properties to CICS attributes is covered, along with other aspects that you may need to consider when communicating with a CICS DPL application via the CICS client.

MQEI CICS DPL direct support

Creating an MQEI Service definition with a service type of CICS DPL direct allows you to run CICS Distributed Program Link (DPL) programs on a CICS server. A CICS DPL program is a CICS program that may be invoked using the CICS command EXEC CICS LINK but may not use CICS terminal or syncpoint facilities.

The MQEI LSX uses a CICS client interface called the External Call Interface (ECI) as the communication mechanism. The CICS ECI allows access to the full range of servers in the CICS family via a CICS client. Also, some members of the CICS family can be accessed without a CICS client as they have a built-in client.

The Notes client or Domino server using the MQEI LSX needs to reside on the same machine as the CICS client (or CICS server with the built-in client).

For more information about the CICS ECI, see the *IBM CICS Family: Client/Server Programming* manual.

Note The CICS DPL direct service type is not supported on UNIX systems.

Creating an MQEI Service definition

The key properties when creating an MQEI Service definition for a CICS DPL direct service are ConnectionManager and ServiceStep.

- The ConnectionManager property gives the name of the CICS server to which connection will be made. It is the name of the CICS server as defined in your CICS client initialization file ServerName property.
  If a CICS server with a built-in client is being used, the ConnectionManager property must always be set to 'CICSMRO' (this is a CICS restriction).
  If the ConnectionManager property is left blank the default CICS server for the CICS client is selected.
- The ServiceStep property gives the name of the DPL program itself. It is possible to invoke several different DPL programs using a single EIService object, so the ServiceStep property should initially be set to the first such program.
- The ServiceContext property gives the CICS transaction identifier that the DPL program will run under.

Creating MQEI Message definitions

An MQEI Message definition for a CICS DPL direct service corresponds to the CICS COMMAREA passed as a parameter to the DPL program by the EXEC CICS LINK command. The source for the MQEI Message definition would typically be a copybook or header file containing the definition of the COMMAREA data structure in an appropriate language.

Often the format of the COMMAREA is the same when passing data into a given DPL program as it is when receiving data back. In this case a single MQEI Message definition will be needed per DPL program. However, sometimes the format of the COMMAREA varies between input and output, so two MQEI Message definitions are needed, one for the input format and one for the output format.

If the output format is longer than the input format, you must pad the MQEI Message definition for the input format so its length matches the output format length, otherwise a storage violation may occur on the CICS server.

Connecting to CICS

The EIService Connect method call uses the CICS ECI to check that the CICS server given by the ConnectionManager property is up and running, and if a CICS client is being used, that the CICS client is up as well. The Connect call will fail if this is not the case. No attempt is made to signon at this point.

Sending a message

Each EIService SendMessage method call translates to a CICS ECI call to invoke the DPL program, named by the ServiceStep property on the CICS server. The EIMessage parameter fields are used to provide the COMMAREA passed as input to the DPL program.

Note that the DPL program need not actually run on the CICS server given by the ConnectionManager property. The CICS definition for the DPL program may specify it as remote in which case the DPL program call is routed to a different CICS server using CICS function shipping.

The CICS transid that the DPL program will run under is given by the ServiceContext property (if this is the start of a new CICS logical unit of work(UOW)).

The EISendOptions parameter controls how the DPL program is invoked. The Identifier property can be specified to supply a unique identity to the invocation and resulting reply. If none is specified, the MQEI LSX generates one automatically and sets it in the Identifier property. The UnitOfWork property specifies how this DPL program relates to the CICS logical unit of work (UOW) that it will run under.

For more information see "Programming a conversation" later in this chapter.

Receiving a message

Each SendMessage method call will always generate exactly one reply from CICS. A ReceiveMessage method call translates to a CICS ECI call to get the COMMAREA returned by the DPL program. The EIMessage parameter's fields are completed from the COMMAREA assuming these match. Because a COMMAREA is just an unstructured buffer, it is only possible for the MQEI LSX to detect a mismatch if the length of the EIMessage exceeds the length of the data. If the EIMessage does not match the data, an MQEI reason code of EIRC_WRONG_MESSAGE is returned and a different EIMessage should be tried.

For more information, see "Variant Messages" in Chapter 4.

When trying different EIMessages, start with the longest message first.

When calling the ReceiveMessage method to receive the reply back from CICS, the EIReceiveOptions Identifier property can be used to target a specific reply, or can be used to receive the first reply. The recommendation is to target a specific reply, as this guarantees that the wrong reply is not picked up in a multi-user environment. If no Identifier is specified the identifier of the reply is returned in the Identifier property.

When calling the ReceiveMessage method to receive a reply, the EIReceiveOptions WaitType and WaitInterval properties can be used to make the call block or not. Due to the way timeout is supported by the CICS ECI, a blocking call will only wait for the specified time if there is a reply outstanding, otherwise it will return immediately with a warning and an MQEI reason code of EIRC_NO_MSG_AVAILABLE. The CICS ECI does not support unlimited timeout.

Disconnecting from CICS

An EIService Disconnect call, or destruction of the EIService, does not lose outstanding replies nor terminate in-progress extended UOWs. Replies can be recovered, and UOWs resumed, following a subsequent EIService Connect for the same EIService instance, or a new instance of the EIService, within the same Notes client or Domino server agent. It is only when the Notes client or Domino agent terminates that the CICS ECI discards all outstanding replies, and abends in-progress UOWs.

It is the responsibility of the LotusScript program to know the state of UOWs it has started. The MQEI LSX provides no facility to interrogate this.

Programming a conversation

A CICS DPL conversation can consist of a single call to a DPL program, or several calls to the same DPL program, or several calls to different DPL programs. Varying the DPL program name may be achieved by changing the EIService ServiceStep property from the LotusScript program prior to making a SendMessage call.

You have the choice to run each DPL program in its own CICS logical unit of work (UOW) or to group programs together in an extended UOW. This is controlled by the EISendOptions UnitOfWork property.

When running an extended UOW, each SendMessage for the UOW must be followed by a ReceiveMessage in strict order. Failure to do this will result in an MQEI reason code of EIRC_CALL_SEQUENCE_ERROR.

When running an extended UOW, the EISendOptions Identifier property is used as the link between the different calls that make up the UOW. On the first SendMessage of the UOW, the LotusScript program can either provide its own, unique, Identifier or let the MQEI LSX generate a unique one on his behalf. This same Identifier **must** be present on all SendMessage calls that comprise the UOW. Failure to do this, or failure to provide a unique UOW on the first SendMessage call, will result in an MQEI reason code of EIRC_IDENTIFIER_ERROR.

It is possible to use a single EIService object to run several separate UOWs in parallel. An example would be a query application, where three queries needed to be made to the same DPL program but with different parameters. Three SendMessage calls could be made one after the other. The management of the Identifiers is the responsibility of the LotusScript program.

Do **not** leave replies outstanding, even if the reply is of no interest to the application. Each new UOW generates a unique CICS ECI message qualifier, destroyed when a reply indicates the UOW has completed. Failure to receive replies may exhaust the range of unique ECI message qualifiers.

Errors

If a failure occurs from ReceiveMessage when receiving a reply that is part of an extended UOW, the normal action the program should take is to issue a SendMessage call with an EISendOptions UnitOfWork property of EIUOW_BACKOUT. This should backout any changes made to recoverable resources on the CICS server.

The backout call is not necessary if it is known the UOW has completed, indicated by MQEI reason codes EIRC_SERVICE_SYSABEND, EIRC_CONNECTION_BROKEN, EIRC_UOW_ROLLED_BACK, and EIRC_UOW_IN_DOUBT. The last of these indicates that the MQEI LSX can not determine the UOW state, and that manual recovery may be required. If a backout call is made, and the UOW has already been backed out, an MQEI reason code of EIRC_CALL_SEQUENCE_ERROR is returned, which can be ignored.

Security

If security is being used on the CICS server, there are two options for performing authentication of users:

1. To specify the CICS userid and password via the /u and /p options of the CICSCLI command when the CICS client is started, outside the Notes environment.
2. To let the MQEI LSX pass the CICS userid and password automatically to the CICS ECI on each SendMessage call.

For the first option,

```
CICSCLI /c=servername /u=userid /p=password
```

must be issued when the CICS client is started. The CICS server uses the userid and password so specified. No other authentication is necessary and the MQEI Security database is not required. This option can not be used if you are connected directly to a CICS server with a built-in client.

Note If you are using a CICS client running on a Domino server, the CICS userid is fixed for all users of all your Notes agents.

For the second option, the CICS userid and password are obtained from the EIService UserId and Authenticator properties, and passed to the CICS ECI on each SendMessage call. If authentication fails, or if the CICS userid does not have sufficient authority to access resources used by the DPL program, an MQEI reason code of EIRC_SECURITY_FAILURE is returned by the ReceiveMessage call.

The MQEI Service definition SystemName property can be used by the MQEI LSX to read an MQEI Security definition for the Notes user from the MQEI Security database. This definition should contain the CICS userid

and password for the Notes user, and is used to set the EIService UserId and Authenticator properties. Alternatively the LotusScript program can prompt the user for this information and set EIService UserId and Authenticator itself.

Data conversion

It is recommended that data conversion of COMMAREAs is performed by the MQEI LSX, in which case CICS data conversion should be turned off.

This is achieved by specifying the CharacterSet and Encoding properties in the MQEI Service definition. The outbound COMMAREA is converted to the specified character set and numeric encoding from the local settings specified in the MQEI initialization file (mqei.ini), and vice versa for the inbound COMMAREA

Note If the DPL program call is function shipped to a different CICS server, the CharacterSet and Encoding properties must be those of the CICS server on which the DPL program actually runs.

Alternatively data conversion can be carried out on the CICS server, by providing CICS DFHCNV macros in a CICS data conversion exit. If this approach is adopted, any MQEI LSX data conversion can be turned off by specifying a CharacterSet and an Encoding in the MQEI Service definition that match those in the MQEI initialization file.

If you are familiar with the CICS ECI

This section may prove helpful if you are familiar with the CICS ECI.

Mapping of MQEI properties to a CICS DPL direct service

The following lists the properties of the EIService, EIMessage, EISendOptions and EIReceiveOptions classes that are relevant to the CICS ECI.

MQEI Class and Property	ECI Equivalent
EIService AbendCode	eci_abend_code (ECI_PARMS)
EIService ServiceStep	eci_program_name (ECI_PARMS)
EIService ServiceContext	eci_transid (ECI_PARMS)
EIService CharacterSet	No equivalent
EIService ConnectionManager	eci_system_name (ECI_PARMS)
EIService Encoding	No equivalent
EIService SystemName	No equivalent
EIService Userid	eci_userid (ECI_PARMS)
EIService Authenticator	eci_password (ECI_PARMS)
EISendOptions Identifier	eci_message_qualifier (ECI_PARMS) Note 1
EISendOptions UnitOfWork	eci_luw_token, eci_extend_mode (ECI_PARMS)
EIReceiveOptions Identifier	eci_message_qualifier (ECI_PARMS) Note 1
EIReceiveOptions WaitInterval	eci_timeout (ECI_PARMS)
EIReceiveOptions WaitType	eci_call_type (ECI_PARMS)

Note 1: Because eci_message_qualifier is only two bytes long, an algorithm is used to associate 24 byte Identifiers to eci_message_qualifiers.

The CICS ECI calls used are ECI_STATE_SYNC (for Connect), ECI_ASYNC (for SendMessage), and various flavors of ECI_GET_REPLY (for ReceiveMessage).

Chapter 10 Programming for a CICS 3270 direct service

Whatever enterprise system your application communicates with, there are some specific aspects to consider.

Here the mapping of the MQEI properties to CICS attributes is covered, along with other aspects that you may need to consider when communicating with a CICS 3270 application via the CICS client.

MQEI CICS 3270 direct support

Creating an MQEI Service definition with a service type of CICS 3270 direct allows you to run CICS 3270 transactions on a CICS server. A CICS 3270 transaction is a CICS transaction that is normally invoked from a 3270 type terminal, and is identified by a transid (transaction identifier) of up to four characters (e.g. CESN).

The MQEI LSX uses a CICS client interface called the External Presentation Interface (EPI) as the communication mechanism. The CICS EPI works by allowing an application program, in this case the MQEI LSX, to emulate a CICS terminal and send and receive 3270 data streams to and from CICS. The CICS EPI allows access to most of the servers in the CICS family via a CICS client. Also, some members of the CICS family can be accessed without a CICS client as they have a built-in client. The Notes client or Domino server using the MQEI LSX needs to reside on the same machine as the CICS client (or CICS server with the built-in client).

Note A 3270 data stream is a data stream used to control the actions of, send data to, and receive data from, a 3270 terminal. As the CICS EPI emulates a 3270 terminal, it uses 3270 data streams to communicate with a CICS Server. You do not need to understand the content of 3270 data streams, only the CICS BMS maps from which they are derived.

For more information about the CICS EPI, see the *IBM CICS Family: Client/Server Programming* manual.

For more information about CICS BMS maps, see the *IBM CICS Application Programming Guide*.

Note The CICS 3270 direct service type is not supported under Windows 3.1 or UNIX systems.

The EIService object created from an MQEI Service definition of this type would typically represent a suite of related transactions on the CICS server (a CICS conversation). The LotusScript program is effectively mimicking the actions of a CICS 3270 terminal user, and must therefore be sensitive to the sorts of things that the CICS transaction expects the terminal user to do (such as positioning the cursor and pressing function keys).

The EISendOptions class is used to define these features, as it controls how a message is sent. Internally, the EIService object maintains an image of the CICS terminal screen.

Creating an MQEI Service definition

The key properties when creating an MQEI Service definition for a CICS 3270 direct service are ConnectionManager, OutboundConnection and ServiceStep.

- The ConnectionManager property gives the name of the CICS server to which connection will be made. It is the name of the CICS server as defined in your CICS client initialization file ServerName property. If a CICS server with a built-in client is being used, the ConnectionManager property must always be set to 'CICSMRO' (this is a CICS restriction). If the ConnectionManager property is left blank the default CICS server for the CICS client is selected (this may not be supported with older CICS clients).
- The OutboundConnection property gives the name of the CICS model terminal to use as a template for the terminal the MQEI LSX is emulating. If the OutboundConnection property is left blank, the model terminal defined in your CICS client initialization file ModelTerm property is used if specified, otherwise the default model terminal for the CICS server is used (note that this varies between servers).
- The ServiceStep property gives the transid of the CICS transaction itself. It is possible to invoke several different transactions during the course of a conversation, so the ServiceStep property should initially be set to the first transaction to run.

Creating MQEI Message definitions

An MQEI Message definition for a CICS 3270 direct service corresponds either to the start data entered along with a CICS transid on the CICS command line, or to the data contained in a CICS BMS map. The source for the MQEI Message definition would typically be a BMS mapset file containing the definitions for one or more BMS maps.

If the MQEI Message definition is for start data, the message should consist of just one field, being of data type STRING at row 1, column 1 with length 80.

If the MQEI Message definition is for a BMS map, only those fields that are of interest to the LotusScript program need be defined. Each field is character based, so must be of data type string, authenticator or system_authenticator, and a row and column must be provided. Such MQEI Message definitions can be created automatically from BMS map source using the MQEI BMS map utility program.

For more information, see BMS map conversion utility later in this chapter.

Connecting to CICS

The EIService Connect method call initializes the MQEI LSX's use of the CICS EPI if this is the first EIService to use the EPI, then uses the EPI to add a terminal on the CICS server for the exclusive use of the EIService object. The name of the CICS server is given by the EIService ConnectionManager property. The name of the model terminal to use is given by the EIService OutboundConnection property.

Sending a message

Each EIService SendMessage method call translates to a CICS EPI call either to start a new transaction or to pass data to satisfy an outstanding EXEC CICS RECEIVE or CONVERSE command issued by a running transaction. When starting a new transaction, the transid to use may be implicit if in the middle of a conversation, in which case it has been specified by the CICS application either via EXEC CICS RETURN TRANSID, or via a protected field on the BMS map, or the transid to use is taken from the EIService ServiceStep property if at the start of a new conversation and the terminal screen is completely empty.

See "Programming a conversation" later in this chapter.

Note The transaction need not actually run on the CICS server given by the ConnectionManager property. The CICS definition for the transid may specify it as remote in which case the transaction is routed to a different CICS server using CICS transaction routing.

An EISendOptions object controls how the built 3270 data stream is sent to CICS. The AttentionId property specifies the function key to be passed to CICS along with the data from the EIMessage. The allowable values are:

- EIAI_ENTER
- EIAI_CLEAR
- EIAI_PA1 to EIAI_PA3
- EIAI_F1 to EIAI_F24

The default is EIAI_ENTER.

Note EIAI_CLEAR and EIAI_PA1 to EIAI_PA3 cause the EIMessage and any transid to be ignored, and a 3270 data stream containing only the function key to be sent to CICS.

The SelectedField property allows the screen cursor to be positioned at a particular named field in the EIMessage. If no cursor position is specified, the cursor is left where it is on the image of the screen kept by the EIService object.

Receiving a message

Each ReceiveMessage method call corresponds to receiving reply data transmitted by an EXEC CICS SEND or CONVERSE command. The EIMessage parameter's fields are completed from the received 3270 data stream assuming they match. If the EIMessage does not match the data, an MQEI reason code of EIRC_WRONG_MESSAGE is returned and a different EIMessage should be tried.

For more information, see "Variant Messages" in Chapter 4.

The MQEI LSX detects a mismatch if the row and column properties of a field in the EIMessage do not correspond to any field in the screen modeled by the 3270 data stream. When trying different EIMessages it is advisable to start with the most complicated BMS map first and work down to the simplest BMS map last. This is because when matching the EIMessage to the screen, the MQEI LSX allows the EIMessage to be a subset of the screen, and it is therefore possible that a simple BMS map may match the screen corresponding to a more complicated BMS map.

It is important to be aware that CICS itself can send a data stream as well as the transaction. Examples of this are the error messages sent to the error line if the transid of the transaction is not known, or if the transaction abends. The location of the error line is usually the last row (typically row 24), although error messages that span lines will start on an earlier row (both the examples mentioned above start on row 23, for example). Note also that such error messages normally overlay the existing screen contents, and do not erase the screen. If your EIMessage contains a field at row 23 column 1 or row 24 column 1 it is a good idea to examine it for the presence of a CICS error message. If your EIMessage does not contain such fields, it is a good idea to try and match an EIMessage that maps row 23 or row 24 to see if a CICS error message has been sent, before trying the real EIMessage. See the CICS 3270 direct sample code for an example of the former.

When calling ReceiveMessage to receive a reply, the EIReceiveOptions WaitType property can be used to make the call block or not. Timeout is not supported by the EPI, so the EIReceiveOptions WaitInterval property has no effect, and a blocking call will wait indefinitely if there is no reply outstanding. For this reason, use of blocking calls is not recommended, and the program should instead poll for a reply, an MQEI reason code of EIRC_NO_MSG_AVAILABLE being returned if the reply has not yet arrived.

When ReceiveMessage is completing the EIMessage parameter, if a field in the received 3270 data stream was transmitted with screen attributes, the color, intensity, highlight and protection attributes of the field are set in the Color, Intensity, HighLight and Protection properties of the corresponding

field in the EIMessage object. These may be interrogated by EIMessage methods. This is useful, for example, where the CICS transaction has highlighted any fields containing invalid input. The LotusScript program could identify such fields and inform the user accordingly.

Note An attempt by the LotusScript program to set a value in a field that is protected will return a reason code of EIRC_PROTECTED_FIELD.

Certain EXEC CICS SEND commands send control characters instead of data. These should be received by specifying an EIMessage parameter of **Nothing** on the ReceiveMessage call.

Disconnecting from CICS

An EIService Disconnect call, or destruction of the EIService object, deletes the terminal on the CICS server.

Note An EIService Disconnect call will fail if there is a transaction still running at the terminal or there are outstanding replies still to be received from CICS. If the EIService is destroyed under these circumstances, any outstanding replies are lost and it is no longer possible to communicate with any transaction still running. An orphan terminal is effectively created on the CICS server which will only be deleted when the last EIService that uses the CICS EPI has disconnected.

Programming a conversation

A CICS 3270 conversation can consist of running either a single transaction or a suite of related transactions.

In CICS terms, this typically corresponds to the end user entering a transid and any start data on the CICS command line to start a transaction, then the transaction sending a screen of data to the end user which the end user processes and sends back to CICS. There may be several such screens involved. The conversation ends when the last screen of data is sent to the end user. The LotusScript program must emulate this, by making an initial SendMessage call to start a transaction, followed by a sequence of ReceiveMessage and SendMessage pairs, and a final ReceiveMessage to get the last screen.

The exact scenario depends on the model terminal being used, but typically the terminal screen would be empty after the EIService Connect method call. The LotusScript program is at the start of a conversation. The first SendMessage method call will build a 3270 data stream from the transid given by the ServiceStep property and any start data described by the EIMessage parameter. The LotusScript program is now in conversation, and must react in accordance with the wishes of the CICS transaction on the CICS server. This would typically be a sequence of EIService ReceiveMessage and SendMessage calls. When in conversation, the EIMessage returned by a ReceiveMessage method call should be passed on the subsequent SendMessage method call. This ensures that the 3270 modified data tags (MDTs) are correctly set and the correct datastream is built.

During a SendMessage call, the transid is explicitly set in the 3270 data stream from the EIService ServiceStep property only at the beginning of a conversation **and** the image of the screen is empty. The LotusScript program can change the ServiceStep property prior to making the SendMessage call. A non-empty screen is taken to mean that a conversation is still in progress. In this case the transid is implicitly known by CICS either because EXEC CICS RETURN TRANSID was used by the CICS transaction or because the transid is contained in a protected field in the screen. In both cases the ServiceStep property is not used.

It is recommended that a SendMessage call to send just the CLEAR key is performed at the end of each conversation. When CLEAR is sent at conversation end in this manner the EIService internal image of the screen is cleared, and CICS does not generate a reply, so a ReceiveMessage method call is not necessary. Because the screen is empty, the next SendMessage method call will take the transid from the ServiceStep property (as discussed above) and start a new conversation. Failure to send the CLEAR key in this manner may cause the MQEI LSX to think a

conversation is still in progress, and omit to explicitly set the transid from the ServiceStep property.

When a SendMessage or ReceiveMessage method call is made, the MQEI LSX validates that this is the correct action for the state of the conversation. If the action is incorrect an MQEI reason code of EIRC_CALL_SEQUENCE_ERROR is returned.

It is not possible to use a single EIService object to run several separate conversations in parallel.

Errors

If an unexpected error occurs, the recommended action is to issue an EIService Disconnect call ignoring any errors and delete the service. This may leave an orphan terminal.

For more information, see "Disconnecting from CICS" earlier in this chapter.

Security

If security is being used on the CICS server, there are two options for performing authentication of users.

- The first is to specify the CICS userid and password via the /u and /p options of the CICSCLI command when the CICS client is started outside the Lotus Notes environment.
- The second is to perform authentication explicitly from the LotusScript program by running the CICS signon transaction CESN (or your local equivalent). It is not possible for the MQEI LSX to do this automatically.

The first option must be used when your CICS client is attached to a CICS for MVS/ESA server. If the CONNECTION definition for the CICS client-CICS server link on the CICS for MVS/ESA server specifies ATTACHSEC(VERIFY), then CICSCLI /c=servername /u=userid /p=password must be issued when the CICS client is started. The CICS server uses the userid and password so specified. No other authentication is necessary.

Note The use of CESN or any other CICS signon facilities is not allowed when connected to CICS for MVS/ESA. The MQEI Security database is not required, and the EIService UserId property and Authenticator properties are not used. Note that if you are running the CICS client on a Domino server, the CICS userid is fixed for all users of all your Notes agents.

The second option must be used when your CICS client is attached to any other CICS server, or if you are connected directly to a CICS server with a built-in client. You must provide a LotusScript program that runs the CESN

transaction or your local equivalent. The CICS 3270 direct sample code shows how CESN can be run, and also provides examples of an MQEI Service definition for the CESN transaction and an MQEI Message definition for the CESN BMS map.

The MQEI Service definition SystemName property can be used by the MQEI LSX to read an MQEI Security definition for the Notes user from the MQEI Security database. This definition should contain the CICS userid and password for the Notes user, and is used to set the EIService UserId and Authenticator properties. Alternatively the LotusScript program can prompt the user for this information and set EIService UserId and Authenticator itself.

Once the EIService UserId and Authenticator have been set, it is the responsibility of the LotusScript program to set the userid and password fields of the EIMessage representing the CESN BMS map. The userid field may be set by the program copying the EIService UserId property. The password field will be set automatically by the MQEI LSX from the EIService Authenticator property during SendMessage if it is defined with a data type of System_Authenticator.

For more Information, See "Security" In Chapter 5.

If you use CESN to signon to CICS, you are signed off either when your LotusScript program issues a Disconnect call or the EIService object is destroyed, or when your LotusScript program runs the CICS signoff transaction CESF or another instance of CESN (or your local equivalent) from the same EIService object.

Data conversion

Data conversion of 3270 data streams is not performed by the MQEI LSX. Because a 3270 data stream is self-defining, CICS automatically performs any data conversion that may be necessary.

Unsupported CICS functions

EXEC CICS RECEIVE BUFFER, which results in a 3270 Read Buffer command being received by the MQEI LSX, is not supported. Any CICS transaction using this command must be modified if it is to work with the MQEI LSX.

Transactions automatically started by CICS at the terminal using EXEC CICS START TERMID, known as ATI transactions, are not supported. When the terminal is added by the EIService Connect method call, ATI is disabled by default. Any transactions started by CICS at the terminal in this manner will be queued indefinitely and will never run.

If you are familiar with the CICS EPI

This section may prove helpful if you are familiar with the CICS EPI.

Mapping of MQEI properties to a CICS 3270 direct service

The following lists the properties of the EIService, EIMessage, EISendOptions and EIReceiveOptions classes that are relevant to the CICS EPI.

MQEI Class and Property	EPI Equivalent
EIService ServiceStep	Transid (CICS_EpiStartTran)
EIService ConnectionManager	System (CICS_EpiAddTerminal)
EIService OutboundConnection	DevType (CICS_EpiAddTerminal)
EIService SystemName	No equivalent
EIService Userid	No equivalent
EIService Authenticator	No equivalent
EISendOptions AttentionId	AID (in 3270 datastream)
EISendOptions SelectedField	Cursor Position (in 3270 datastream)
EIReceiveOptions WaitType	Wait (CICS_EPIGetEvent)

The CICS EPI calls used are CICS_EpiInitialize, CICS_EpiAddTerminal, CICS_EpiGetEvent, CICS_EpiReply, CICS_EpiStartTran, and CICS_EpiDelTerminal.

BMS maps

About Basic Mapping Support (BMS)

Basic Mapping Support (BMS) is an application programming interface between CICS programs and terminal devices (such as printers or displays). Many CICS applications use BMS when communicating with 3270 display terminals.

BMS works by taking data from a program and displaying it in a predetermined format. This predetermined format is defined by a BMS map in a BMS mapset file. BMS merges variable data supplied by the CICS program with constant data (such as the position of the text, field labels and default values for the fields). This constant data is contained within the BMS map definition. From this information, BMS builds a 3270 data stream for the terminal to which the CICS program is communicating.

BMS provides three macros for defining maps:

DFHMSD - Map set definition macro

The DFHMSD macro groups single maps into a map set. A BMS mapset file can contain zero, one or several map definitions.

DFHMDI - Map definition macro

The DFHMDI macro defines a map within the map set defined by the previous DFHMSD macro. A map contains zero or more fields.

DFHMDF - Field definition macro

The DFHMDF macro defines an individual field within a map defined by the previous DFHMDI macro.

How the BMS map conversion utility works

The MQSeries Enterprise Integrator provides a utility program to automatically create MQEI Message definitions from BMS maps:

1. The BMS map conversion utility reads the MQEI initialization file (mqei.ini) and opens the specified MQEI Definition database.
2. The BMS map conversion utility scans the mapset file for a DFHMDI statement, ignoring any other comments or statements it finds on the way.
3. When the BMS map conversion utility finds a DFHMDI statement, it extracts the map name from the statement label.
4. It then creates a new MQEI Message definition on the target MQEI Definition database (that is defined in the mqei.ini file).
5. The BMS map conversion utility then scans down the file for DFHMDF statements within the DFHMDI statement.
6. When the BMS map conversion utility finds a DFHMDF statement, it extracts the field name from the statement label.
7. It then creates a Field definition on the target MQEI Definition database (that is defined in the mqei.ini file).

Note Any statement other than a SPACE directive or a comment, including # macro directives, will terminate the scan for DFHMDF statements and end the processing for the current map. The BMS map conversion utility will resume scanning down the file for next DFHMDI statement as described in step 1.

Ignored statements

If the BMS map conversion utility cannot find a label, it ignores the DFHMDF statement. This is because the BMS map conversion utility assumes that the field is of no interest to the MQEI programmer. It also ignores any comments and SPACE directives that may be present in the BMS mapset file.

Syntax checking

The BMS map conversion utility performs very limited syntax checking when scanning the BMS mapset file. It is assumed that any mapset file presented to the utility has been successfully processed by CICS beforehand. If this is not the case, the results from the BMS map conversion utility are unpredictable and any MQEI definitions generated are not guaranteed to match the BMS maps from which they are derived.

Before running the BMS map conversion utility

Before you use the BMS map conversion utility, you must ensure that:

- The Notes executable directory is included in the PATH environment variable.
- The directory containing the code page files shipped with Notes is included in the PATH environment variable. Notes code page files have an extension of .cls.
- On AIX, HP-UX, and Sun Solaris, the Notes data directory is included in the the PATH environment variable.
- On AIX, HP-UX, and Sun Solaris, the MQEI_INI_PATH, MQEI_XLAT_PATH, and Notes_ExecDirectory environment variables are correctly set (for other platforms these are set as necessary by the installation program).
- On AIX, HP-UX, and Sun Solaris, the Notes_ISOLATION environment variable is set to 1 as described in the Lotus Notes Release Notes section for UNIX Platforms and Multiple Notes Clients.

  For example, using the Korn shell:

  ```
  Notes_ISOLATION=1;export Notes_ISOLATION
  ```

Failure to do the above may result in Notes error or warning messages or segmentation faults.

Running the BMS map conversion utility

To convert a BMS map file into an MQEI message definition follow the instructions described below.

Note This is not available on Windows 3.1, Windows for Workgroups, and WIN-OS/2.

Enter:

```
mqeibms [-r] [-?] mapname.bms
```

mqeibms starts the BMS map conversion utility.

The -r option specifies that an MQEI Message definition on the database, with the same name as a map in the BMS map file, is to be replaced along with all its associated Field definitions.

The -? option displays a line describing the syntax of the MQEIBMS command.

Note If the -r option is not specified and an MQEI Message definition with the same name as a map in the BMS map file is present in the MQEI Definition database, an MQEI reason code of EIRC_DEFN_ALREADY_EXISTS is returned and the definition is not replaced.

Prompting for Notes password

When you run the MQEIBMS utility, you may be prompted to enter your Lotus Notes password. If this happens the first time, it happens every time you run MQEIBMS. Because MQEIBMS only deals with one mapset file at a time this may be inconvenient if you are converting a number of mapsets.

You can avoid this problem by changing the settings of your Lotus Notes client:

1. Make sure that your Notes client is running on the same workstation as MQEIBMS.
2. From the Notes client, select File - Tools - User ID...
3. When prompted, enter your Notes password.
4. Select the box marked, "Share password with Notes add-ins".

Reason codes

When you run MQEIBMS and an error occurs, an error message will be displayed that contains an MQEI reason code.

These are explained in Chapter 11.

In the addition, if the reason code is EIRC_DEFN_DB_SYSERROR, a Notes error code and accompanying diagnostic text (if available) is displayed.

After running the BMS map conversion utility

Having successfully created an MQEI Message definition, you must build the MQEI message.

When an MQEI message has been built, it can be used when communicating with a CICS 3270 direct service in the same way as any other MQEI Message definition.

For more information, see "Building MQEI Message definitions" in Chapter 3.

Chapter 11 Troubleshooting

This chapter explains:

- How to check the level of code you are running
- MQEI LSX dynamic loading
- Data Conversion
- How to use the trace facility
- Reason codes

Code level tool

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

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

From the command prompt, change to the directory containing the MQEI LSX library (eilsx.dll or libeilsx.*) or add the full path name and enter:

```
mqeilev yyyyy > xxxxx.xxx
```

where yyyyy is the name of the shared library (e.g. eilsx.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.

This is a sample extract of what you could see:

```
COPYRIGHT IBM CORP. 1996, 1997 ALL RIGHTS RESERVED, LICENSED
MATERIALS-PROPERTY OF IBM

@(!)      ***** Code Level is 1.0.0 *****

@(#) lib/mqlsx/xmqtrca.c, mqlsx, lnk000, lnk000 L970401  1.60
97/04/01 12:04:08

@(#) lib/mqlsx/xmqxlat.c, mqlsx, lnk000, lnk000 L970401  1.40
97/04/01 12:04:13

@(#) lib/mqlsx/xmqfdca.c, mqlsx, lnk000, lnk000 L970327  1.32
97/03/05 16:09:24

@(#) lib/mqlsx/xmqutila.c, mqlsx, lnk000, lnk000 L970327 1.26
97/01/15 14:42:07

@(#) lib/mqea/gmqadyn0.c, mqlsx, lnk000, lnk000 L970327  1.14
97/03/13 15:12:43

@(#) lib/mqea/gmqadyn1.c, mqea, lnk000, lnk000 L970327   1.16
97/03/18 08:21:25

@(#) lib/mqlsx/xmqcsa.c, mqlsx, lnk000, lnk000 L970401   1.19
97/04/01 12:04:04

@(#) lib/mqea/gmqxbase.cpp, mqea, lnk000, lnk000 L970327 1.10
97/03/24 11:06:47

.......

.......
```

```
@(#) lib/mqea/gmqxlsx.cpp, mqea, lnk000                  1.17
97/04/02 09:50:58

@(#) lib/imqi/imqobj.cpp, imqi, lnk000, lnk000 L970327
1.25.2.1  96/10/04 10:40:34

@(#) lib/imqi/imqpmo.cpp, imqi, lnk000, lnk000 L970327

@(#) lib/imqi/imqpro.cpp, imqi, lnk000, lnk000 L970327

@(#) lib/imqi/imqque.cpp, imqi, lnk000, lnk000 L970327
```

Dynamic loading and the MQEI LSX

The shared libraries (for MQSeries or CICS) are dynamically loaded when your application calls the EIService Connect method. If you have a problem on the Connect call (typically returning the reason code EIRC_DYNAMIC_LOAD_ERROR) the following information may help.

General

When you specify:

```
Uselsx "eilsx"
```

in your LotusScript program, Notes uses standard system services (different on each platform) to locate your LSX.

For example, on Windows 95 it looks for an object called eilsx.dll in the current working directory or on a directory in the search path (the exact mechanism for establishing the search path differs according to the platform).

See Lotus documentation for more information.

Tip One way of identifying whether or not your search path is in error is to change the Uselsx to include the full path name.

The MQEI LSX generates at various points within its code, standard API calls to MQSeries, CICS ECI, and CICS EPI (MQCONN, CICS_ExternalCall etc.). These are "trapped" within the MQEI LSX code at entrypoints with the same names as the MQSeries, CICS ECI, or CICS EPI entry points.

If this is the **first call** to either MQSeries or CICS within the application **,** the MQEI LSX code tries to dynamically load the system object containing the real MQSeries, CICS ECI, or CICS EPI code.

- On Windows and OS/2 these objects are called DLLs (Dynamic Link Library).
- On Unix systems these objects are called 'shared libraries'.

If the necessary object is found, the MQEI LSX detects and remembers the entry points of the real MQSeries, CICS ECI, and CICS EPI functions. The "trapped" call is passed to the real function entry point.

After the initial call, subsequent calls to the service are "trapped" in the MQEI LSX code and immediately passed to the remembered entry points.

The system routines called by the MQEI LSX to provide this functionality are:

On Windows 3.1, Windows 95, and Windows NT

The mechanism used is LoadLibrary coupled with GetProcAddress to find the appropriate dll (mqm.dll or mqic.dll for MQSeries, faacicnt.dll or cclwin32.dll for CICS ECI / EPI) and determine the real entry addresses. The search for the dlls instigated by the LoadLibrary call uses the normal Windows mechanisms, i.e. looks first in the current working directory and then in the libraries that are in your PATH.

Note If you override the dll to be loaded, by using the MQEI_xxx_LIB environment variable, do not forget that this dll picks up other dlls, which need to be on the PATH.

On OS/2

The mechanism used is DosLoadModule coupled with DosQueryProcAddr to find the appropriate dll (mqm.dll or mqic.dll for MQSeries, faacic32.dll or cclos232.dll for CICS ECI/EPI) and determine the real entry addresses. The search for the dlls instigated by the DosLoadModule call uses the normal OS/2 mechanisms, i.e. looks first in the current working directory and then in the libraries that are in your LIBPATH.

On Unix Platforms

On the Unix platforms (AIX,Sun Solaris and HP-UX) the mechanism for detecting and loading the MQSeries shared libraries is different to that on non-Unix. This support depends upon the code in the MQEI LSX shared libraries (libeilsx.a on AIX, libeilsx.so on Solaris, libeilsx.sl on HP) finding and successfully loading a stub called either eilsxmqm or eilsxmqic. These stubs are themselves shared libraries linked with the appropriate MQSeries base or MQSeries client libraries supporting the MQ API calls. If these stubs cannot be found and successfully loaded then EIRC_DYNAMIC_LOAD_ERROR is produced.

Note The stubs **are not** LotusScript Extensions - they **are not** loadable via Uselsx.

In order to find these stubs the MQEI LSX uses the standard system shared library mechanisms on each platform.

- On AIX the “load” mechanism is used.
- On Sun Solaris “dlopen” coupled to “dlsym” is used.
- On HP-UX “shl_load” and “shl_findsym” is used.

The default on each platform is to first look for eilsxmqm and then (if eilsxmqm cannot be found) look for eilsxmqic. This search order may be over-ridden by use of the MQEI_MQ_LIB environment variable. The primary intention of MQEI_MQ_LIB is to allow eilsxmqic to be used in preference to eilsxmqm when both the MQ client (libmqic) and MQ server libraries are available - however it can also be used if you encounter problems with the eilsx being able to locate the stubs (such problems are usually due to system setup problems).

Note The MQEI_MQ_LIB environment variable *must* be used on HP-UX.

As noted before the stubs are themselves linked with the standard MQSeries API shared libraries and standard system libraries and therefore require (just like any other MQSeries application) that the normal MQSeries and system libaries are available. The normal method of ensuring that all the required code is accessible (the stubs, the MQSeries libraries, the system libraries) is by putting the shared libaries in the /lib or /usr/lib directories directly or (more commonly) by putting soft links to these libraries into /lib or /usr/lib. MQEI adds such soft links as part of the install process.

The use of the MQEI_xxx_LIB environment variable

There are 3 environment variables available, where xxx is set to MQ, ECI, or EPI.

The use of these is primarily in a development environment. In a production environment you would not normally need to set these as the MQEI LSX finds the appropriate MQSeries and CICS shared libraries automatically via the standard system dynamic load mechanisms (unless you are running MQEI on HP-UX, when you must specify the library explicitly).

They enable you to have multiple copies of the MQSeries and CICS products installed (either at different levels, under test, or where you want to develop more than one application) on the same hardware. Setting these environment variables enables you to bypass the normal search routes.

You may also want to set the MQEI_MQ_LIB environment variable when you want to force the call to MQSeries to use the MQSeries client even though you have the MQSeries server dll in you system path:

For example (on Windows NT or OS/2):

```
set MQEI_MQ_LIB=C:\MQM\BINMQIC.DLL
```

EIRC_DYNAMIC_LOAD_ERROR

If the dynamic load fails (using the EIService Connect method), the reason code returned to your LotusScript program is EIRC_DYNAMIC_LOAD_ERROR. Look at the EIService SystemErrorText property for the name of the dll that MQEI LSX failed to load.

MQEI databases not displaying text

If text is not displayed or is truncated within the MQEI databases, the problem is related to the font size you have selected. Select a smaller font size in either or both, Notes and your operating system.

Data conversion

Different systems often have different rules on the interpretation of internal data, which becomes an important factor in exchanging data between systems. The MQEI LSX takes account of this, and via appropriate use of the Encoding and CharacterSet values for the [Base] stanza in the mqei.ini file and the EIService (in the MQEI definition database), converts your messages for you.

Encoding translation (such as Big-Endian to Little-Endian) takes place internally within the MQEI LSX code for those message fields defined with a data type of long or short that require conversion.

For CharacterSet conversion the situation is more complicated due to the proliferation of many different code pages even on the same system type. For example one machine may be using code page 437 and another code page 850 or code page 819. Many characters in these code sets are compatible (for example, alphanumerics between the 437, 850, and 819 code page are represented by the same code points), but some are not (typically currency symbols and characters that are specific to a national language).

The conversion that happens within the MQEI LSX for a field with data type of STRING (or indeed AUTHENTICATOR or SYSTEM_AUTHENTICATOR) is best illustrated by this example, where the mqei.ini file specifies:

```
[Base]
CharacterSet = Local
```

and the EIService specifies:

```
CharacterSet:500
```

Suppose the machine that the MQEI LSX is operating under is running OS/2. When the EISession is created, because the CharacterSet in the [Base] stanza of the mqei.ini file is Local, the MQEI LSX will determine the system code page from the operating system. Typically for an OS/2 machine this would be 850.

The EIService in this example would have a CharacterSet value of 500 (typical of an MVS system). When a SendMessage method for the EIService is executed, any character based fields in the message would need conversion from 850 (ASCII) representation into 500 (EBCDIC). The code within the MQEI LSX tries to dynamically load a table for this conversion - the table is 256 bytes in size and maps the 256 code points in code page 850 to those in code page 500, at the same time another table is loaded that enables the reverse conversion.

Additional Notes

On Windows NT (Japanese), when the operating system reports that the code page is 932, the MQEI LSX treats this as code page 943. This allows data originating in a Windows NT environment using 932(MS) to be successfully converted to 932 (IBM) as used on AIX systems. This also means that if you wish to send a message with conversion to an AIX system supporting 932 then you need to set the character set to 932 in the Service definition - you cannot let it default to local.

Your questions answered

1. How does the MQEI LSX locate these tables?

Answer: It uses the MQEI_XLAT_PATH environment variable.

2. What are these tables called?

Answer: The tables take the form aaaabbbb.tbl, where aaaa is the 4 hex digit representation of the from code page and bbbb is that of the to code page.

Thus the two tables associated with 500 (= X'01F4') to 850 (= X'0352') and vice versa are 01F40352.tbl and 035201F4.tbl. The tables generally come in pairs and generally support 'round trip conversion' which means that a string converted from one code page to another code page going out, will (if echoed back), be converted coming back to the exact same characters.

3. Are there any other files associated with this conversion?

Answer: Yes, one other. Its called mqeiccs.tbl and sits in the same place as the other data conversion tables. As part of the MQEI LSX initialization this file is loaded (xxxInitialize - the first thing you see in an MQEI trace) into memory. It contains additional details needed to support a particular code set, such things as whether it's ASCII or whether it's EBCDIC, and whether the code page maps directly to another one. The basic rule is that if a code page does not have an entry in this table it is not supported and conversion will fail, giving rise to an EIRC_DATA_CONVERSION_ERROR.

Note The MQEI LSX only supplies a subset of the vast number of possible conversions. Consult the file readme.ccs that can be found in the conv directory of your install package for a list of this subset.

4. What should I do if I get EIRC_DATA_CONVERSION_ERROR?

Answer: Firstly check that the MQEI_XLAT_PATH environment variable has been set correctly and that the directory it points to contains a mqeiccs.tbl and the necessary conversion tables for your particular conversion. If the MQEI_XLAT_PATH is correct, check that the [Base] and EIService CharacterSet values are correct.
If these are both set to Local and the problem is occurring on a ReceiveMessage (for an MQSeries based service) , one possibility is that the MQSeries system that sent the incoming message is using a character set that is not in the ccsid.tbl. If these checks do not solve your problem then you should start trace, probably with MQEI_TRACE_LEVEL set to 9, this should provide trace showing the conversion that was attempted, something like:

```
-------------->EADriver :: convertString

! Converting from code page nnn to mmm
```

where nnn is the 'from' code page and mmm is the 'to' code page.

You should check that these values are what you expect them to be. Check for their existence in both the ccsid.tbl and as a conversion table aaaabbbb.tbl (where aaaa is the hex representation of nnn and bbbb that of mmm).

For more information, refer to the trace for an entry similar to:

```
---->ObtainSystemCP

! Code page is xyz
```

where xyz is the code page of the Local System (which may be overridden in the [Base] stanza of the mqei.ini file).

5. On Windows NT and Windows 95, I seem to be losing National Language characters and currency symbols when sending and receiving messages?

Answer: This arises due to an inconsistency between the character set used by Notes (taken from the ACP) and that used by MQEI (taken from the OEMCP). The suggested way to overcome this is to set the Character Set value in your mqei.ini file to the same value as the ACP. You can determine this value using the Windows registry editor and looking under key:

```
HKEY_LOCAL_MACHINE
\SYSTEM\CurrentControlSet\Control\Nls\CodePage\ACP
```

Typically it will be a value like 125x where x ranges from 0-7.

Subsystem error logging

If the MQEI LSX raises an error that implies an MQSeries or CICS error, there may be additional information on the MQSeries or CICS error logs.

For more information on diagnosing MQSeries errors, see the *MQSeries System Administration* guide.

For more information about diagnosing CICS errors, see the *CICS Problem Determination* guide for your system.

For more information about diagnosing IMS errors, see the *IMS/ESA Diagnosis Guide and Reference*.

Using trace

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

The environment variables that you set to control trace:

- MQEI_TRACE
  Use this to turn tracing on and off.
- MQEI_TRACE_PATH
  Use this to point to the directory to hold the trace files.
- MQEI_TRACE_LEVEL
  Use this to set the level of detail you want recorded in the trace file.

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 MQEI LSX, but it will not affect the performance of your Notes, CICS, or MQSeries environments. When you no longer need a trace file, it is your responsibility to delete it.

You must stop and restart Notes for any change to the status of the MQEI_TRACE variable to take effect.

Note The MQEI LSX 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 MQEI LSX, 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 MQEInnnn.trc, where nnnn 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 MQEI_TRACE_PATH=drive:\directory	Sets the trace directory where the trace file will be written
SET MQEI_TRACE_PATH=	Removes the MQEI_TRACE_PATH environment variable, the trace file is written to the current working directory (when Notes is started)
SET MQEI_TRACE_PATH	Displays the current setting of the trace directory path on OS/2, Windows for WorkGroups, and Windows 3.1
ECHO %MQEI_TRACE_PATH%	Displays the current setting of the trace directory path
SET MQEI_TRACE=xxxxxxxx	This sets tracing ON. You switch tracing on by putting one or more characters after the '=' sign For example: SET MQEI_TRACE=yes or SET MQEI_TRACE=no In both of these examples, tracing will be set ON
SET MQEI_TRACE=	Sets tracing OFF
SET MQEI_TRACE	Displays the contents of the environment variable on OS/2, Windows 3.1 and Windows for WorkGroups
ECHO %MQEI_TRACE%	Displays the contents of the environment variable
SET	Displays the contents of all the environment variables on OS/2, Windows 3.1,Windows for WorkGroups, and Windows NT

Commands on AIX, HP-UX and Sun Solaris

Command	Effect
export MQEI_TRACE_PATH=/directory	Sets the trace directory where the trace file will be written
unset MQEI_TRACE_PATH	Removes the MQEI_TRACE_PATH environment variable, the trace file is written to the current working directory (when Notes is started)
echo $MQEI_TRACE_PATH	Displays the current setting of the trace directory path
export MQEI_TRACE=xxxxxxxx	This sets tracing ON. You switch tracing on by putting one or more characters after the '=' sign For example: export MQEI_TRACE=yes or export MQEI_TRACE=no In both of these examples, tracing will be set ON
unset MQEI_TRACE	Sets tracing off
echo $MQEI_TRACE	Displays the contents of the environment variable
set	Displays all the settings for all the environment variables for the session

Trace level

The environment variable MQEI_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 MQEI_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.

The format of the trace output may differ slightly on different platforms (this example is from Windows NT):

```
Trace for program d:\notes\NLNOTES.EXE(Enterprise Integrator)
 started at Tue Mar 11 09:09:59 1997

************************************************************
  The first line of the trace shows the start date and time,

  and the program that produced the trace. This is followed

  (see trace lines below) by information about the code level
  and the date the code was built on. These first few lines

  result from the trace initialization done by the eilsx code
  loaded as a result of the Uselsx "eilsx" statement in your

  LotusScript.


************************************************************
@(!)      ***** Code Level is 1.0.0 *****
   ! BuildDate Mar 11 1997
   ! Trace Level is 2

************************************************************
  The above 3 lines of trace give the code level and the date
  that the code was built together with a trace level. The

  trace level defaults to 2 but may be overridden by use of

  the MQEI_TRACE_LEVEL environment variable that enables more
  or less trace data to be generated. With a trace level of
```

```
  2 only the first 2 levels into the eilsx code is traced.

************************************************************
(00205)@09:09:59.560
************************************************************
  After 40 entries there is a time stamp. The number at the
  front of the timestamp (205 in this case) is the thread id.
  This timestamp may be useful in comparing this trace with
  traces from other sources (such as an MQSeries trace) or
  with traces from another machine (although you need to be
  careful to establish that the times on both machines are in
  step!)

************************************************************
   -->xxxInitialize
************************************************************
   Trace entries such as the one above show entry into a code
   function (In this case xxxInitialize). The number of --
   signs prior to the arrow (>) shows the depth within the
   code. Each entry line (having a >) should at some later
   point have an exit line (<) with the same number of --
   signs.
************************************************************
*
```

```
   ---->ObtainSystemCP
   ! Code page is 437
   <----ObtainSystemCP (rc= OK)
   <--xxxInitialize (rc= OK)

************************************************************
   The above shows the function exit point corresponding to

   the trace entry 5 lines above. The text within the
brackets
   (rc= OK) shows the return value from the function. In most
   cases (BUT NOT ALL!) OK shows that things are working as

   expected.


************************************************************
   -->LSX: MainEntryPoint
   ! LSX: Expecting LotusScript Interpreter Version 2.0
   <--LSX: MainEntryPoint (rc= OK)
   -->LSX: EALSX_MessageProc
   ! LSX: LSX_MSG_SETPATH received; library loaded from
          d:\notes\eilsx.DLL

************************************************************
   Entries in the trace beginning with ! are informational.

   They have often been inserted in the code by the
programmer
   to display useful information. In the case above for

   example the entry shows where Lotus Notes has loaded the

   eilsx dll from.
```

```
************************************************************
   <--LSX: EALSX_MessageProc (rc= OK)
   -->LSX: EALSX_MessageProc
   ! LSX: LSX_MSG_INITIALIZE received
   ! LSX: SUCCESS on ClassRegistration of EISession
   ! LSX: SUCCESS on ClassRegistration of EIService
   ! LSX: SUCCESS on ClassRegistration of EIMessage
   ! LSX: SUCCESS on ClassRegistration of EISendOptions
   ! LSX: SUCCESS on ClassRegistration of EIReceiveOptions
   ---->RegisterEAConstants
   <----RegisterEAConstants (rc= OK)
   <--LSX: EALSX_MessageProc (rc= OK)
   -->LSX: Class entry point
   ! LSX: LSI_ADTMSG_CREATE received for class:EISession
   ! LSX: LotusScript >>>        Set [X] = new EISession
   ! LSX: allocating unique EISession
   ! LSX: >>> MEM >>> new: 0x1a805c4
   ---->EASession::EASession()
   <----EASession::EASession() (rc= OK)
   <--LSX: Class entry point (rc= OK)
   -->LSX: ClassControl
   ! LSX: LSI_ADTMSG_ADDREF received for class:EISession
refCount = 1
   <--LSX: ClassControl (rc= OK)
   -->LSX: ClassControl
   ! LSX: LSI_ADTMSG_PROP_GET for class:EISession;
property:ReasonCode[2]
   ! LSX: LotusScript >>>        [X] = EISession.reasonCode
   <--LSX: ClassControl (rc= OK)

************************************************************
   -->LSX: ClassControl
```

```
   ! LSX: LSI_ADTMSG_METHOD received for class:EISession;
method:CreateService[3]

   ! LSX: >>> MEM >>> new: 0x1a8066c

   ---->EAService::EAService

   ! >>> Adding EAService reference at 0x1a8066c >>>

   ! Defaulting to LOCAL encoding

   ! Security database not available...continuing

   <----EAService::EAService (rc= OK)

   ! LSX: LotusScript >>>        Set [X] =
EISession.createService("MQServ")  [0x1a8066c]

   ---->EAError :: translateCodes( const EA_LONG )

   <----EAError :: translateCodes( const EA_LONG ) (rc= OK)

   <--LSX: ClassControl (rc= OK)

************************************************************

   The above 12 trace lines show the start and end of the
   trace entries you would see as a result of running a
   LotusScript line that used the createService method
   established by using the eilsx. These (LSX:ClassControl)
   entries are the ones that are output when Notes code calls
   eilsx code and as such can be used to determine what part
   of your LotusScript may have given rise to an error. Note
   there is a comment line (Set [x] = .....) that should help
   tie the LotusScript line into the trace output.
 ************************************************************

   -->LSX: ClassControl

   ! LSX: LSI_ADTMSG_ADDREF received for class:EIService
refCount = 1

   <--LSX: ClassControl (rc= OK)

   -->LSX: ClassControl

   ! LSX: LSI_ADTMSG_EVENT_REG received for class:EIService
event = EIERROR
```

```
   ! LSX: LotusScript >>>        On Event EIERROR From
EIService Call [X]
   <--LSX: ClassControl (rc= OK)
  ..............................................
  ....... Lines removed for clarity ..............
  ..............................................
   -->LSX: ClassControl
   ! LSX: LSI_ADTMSG_DELETE received for class:EIService
   ---->EAService::~EAService
   ! >>> Removing EAService reference at 0x1a8066c >>>
(00205)@09:10:34.441
   <----EAService::~EAService (rc= OK)
   ! LSX: >>> MEM >>> delete: 0x1a8066c
   <--LSX: ClassControl (rc= OK)

************************************************************
   The above 8 trace lines show the start and end of the

   trace entries resulting from the deletion (after issuing a
   notes shut down) of the EIService created above.


************************************************************
   -->LSX: ClassControl
   ! LSX: LSI_ADTMSG_DELETE received for class:EISession ...
ignoring
   <--LSX: ClassControl (rc= OK)
   -->LSX: EALSX_MessageProc
   ! LSX: LSX_MSG_TERMINATE received
   ! LSX: Deleting EISession
   ---->EASession::~EASession
   <----EASession::~EASession (rc= OK)
   ! LSX: >>> MEM >>> delete: 0x1a805c4
```

```
   ---->xmqTermCommonServices
   ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   ! Trace termination - closing trace file
   ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

************************************************************
*
   The final lines from a normal trace run should look like
   the above. Note there is a trace entry into the function
   xmqTermCommonServices but no trace exit as this function

   shuts down trace.

************************************************************
*
```

In addition to the sort of trace entries shown above you can also obtain trace entries showing more detail of the interface between the eilsx and the various drivers (MQ, ECI and EPI). In order to obtain these then the MQEI_TRACE_LEVEL environment variable needs to be set to an appropriate value (8 or 9 recommended). Note this will increase quite dramatically the amount of trace data produced and you will see entries of the type:

```
   -->LSX: ClassControl
   ! LSX: LSI_ADTMSG_METHOD received for class:EIService;
method:Connect[2]
   ! LSX: LotusScript >>> Call EIService.connect()
[EIService = "MQServ"]
   ---->EAService::connect
   ------>EAError :: resetCodes()
   <------EAError :: resetCodes() (rc= OK)
   ------>EAMQNativeDriver::connect
   -------->EAMQDriver::connect
   ---------->EAMQDriver::connectToMQ
   ------------>ImqQueueManager::connect
   -------------->gmqadyn0:MQCONN
```

```
   ! >>>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 ---
   ! gmqadyn0 : About to try and find a dynamic library
   ---------------->EstablishMQEPs
   <----------------EstablishMQEPs (rc= 1)
   ! gmqadyn0: About to go off to real MQCONN
   ! gmqadyn0: Back from real MQCONN
   ! <<<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 ---
   ! <<<HConn...
   0000  4C 2B B7 01                                      :
L+..
   ! <<<Completion Code...
   0000  00 00 00 00                                      :
....
   ! <<<Reason Code...
   0000  00 00 00 00                                      :
....
   <--------------gmqadyn0:MQCONN (rc= OK)
   <------------ImqQueueManager::connect (rc= OK)
   ! EAMQDriver::connectToMQ (MQCONN) - CC 0 , Reason 0
   !  ... for Queue Manager freddy .
   <----------EAMQDriver::connectToMQ (rc= OK)
   ! Outbound and inbound queues are the same
   ---------->EAMQDriver::openMQApplicationQ
   ------------>EAMQDriver :: openQueue
```

```
   -------------->ImqObject::open

   ---------------->gmqadyn0:MQOPEN

   ! >>>HConn...

         4C 2B B7 01                                       :
L+..

   ! >>>MQOD...

   0000  4F 44 20 20 01 00 00 00 01 00 00 00 53 59 53 54  :
OD  ........SYST

   0010  45 4D 2E 44 45 46 41 55 4C 54 2E 4C 4F 43 41 4C  :
EM.DEFAULT.LOCAL

   0020  2E 51 55 45 55 45 00 00 00 00 00 00 00 00 00 00  :
.QUEUE..........

   0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :
................

         --- 2 lines identical to above ---

   0060  00 00 00 00 00 00 00 00 00 00 00 00 41 4D 51 2E  :
............AMQ.

   0070  2A 00 00 00 49 6D 71 4F 62 6A 65 63 74 3A 3A 63  :
*...ImqObject::c

   0080  6C 6F 73 65 00 00 00 00 49 6D 71 4F 62 6A 65 63  :
lose....ImqObjec

   0090  74 3A 3A 63 6C 6F 73 65 20 28 65 72 00 00 00 00  :
t::close (er....

   0100  00 00 00 00 00 00 00 00                          :
........

   ! >>>Options...

   0000  11 20 00 00                                       : .
..

   ! >>>Hobj...

   0000  00 00 00 00                                       :
....

   ! <<<HConn...

   0000  4C 2B B7 01                                       :
L+..

   ! <<<MQOD...
```

```
   0000  4F 44 20 20 01 00 00 00 01 00 00 00 53 59 53 54  :
OD  ........SYST

   0010  45 4D 2E 44 45 46 41 55 4C 54 2E 4C 4F 43 41 4C  :
EM.DEFAULT.LOCAL

   0020  2E 51 55 45 55 45 00 00 00 00 00 00 00 00 00 00  :
.QUEUE..........

   0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :
................

         --- 2 lines identical to above ---

   0060  00 00 00 00 00 00 00 00 00 00 00 00 41 4D 51 2E  :
............AMQ.

   0070  2A 00 00 00 49 6D 71 4F 62 6A 65 63 74 3A 3A 63  :
*...ImqObject::c

   0080  6C 6F 73 65 00 00 00 00 49 6D 71 4F 62 6A 65 63  :
lose....ImqObjec

   0090  74 3A 3A 63 6C 6F 73 65 20 28 65 72 00 00 00 00  :
t::close (er....

   0100  00 00 00 00 00 00 00 00                          :
........

   ! <<<Options...

   0000  11 20 00 00                                      : .
..

   ! <<<Hobj...

   0000  58 84 4B 00                                      :
X.K.

   ! <<<Completion Code...

   0000  00 00 00 00                                      :
....

   ! <<<Reason Code...

   0000  00 00 00 00                                      :
....

   <----------------gmqadyn0:MQOPEN (rc= OK)

   <--------------ImqObject::open (rc= OK)

   ! EAMQDriver::openQueue (MQOPEN) - CC 0 , Reason 0

   <------------EAMQDriver :: openQueue (rc= OK)

   <----------EAMQDriver::openMQApplicationQ (rc= OK)
```

```
   <--------EAMQDriver::connect (rc= OK)
   <------EAMQNativeDriver::connect (rc= OK)
(00265)@09:54:00.036
   ------>EAError :: translateCodes( const EA_LONG )
   -------->EAError :: resetCodes()
   <--------EAError :: resetCodes() (rc= OK)
   <------EAError :: translateCodes( const EA_LONG ) (rc= OK)
   <----EAService::connect (rc= OK)
   <--LSX: ClassControl (rc= OK)

************************************************************
*
   The lines above show the trace entries associated with the
   use of the Connect method against a previously created
service.
   In this case it's a service using MQSeries and shows the
   dynamic loading of the mqm dll together with the data
blocks
   passing from the eilsx code into the MQ code. Similar
levels
   of data would be shown for ECI or EPI calls.


************************************************************
*
```

Reason Codes

This lists the external reason codes provided by the MQEI LSX. An MQEI LSX object's ReasonCode property is set when an error occurs, or to warn of a potential error, accessing a method or property of the object concerned.

Reason codes may also be returned by the MQEI utility programs.

The ReasonCode property is dependent on the setting of the accompanying CompletionCode property.

ReasonCode	Explanation / Action
EIRC_NONE (0)	Success. No action

The following reason codes can be encountered by your LotusScript program.

Reason codes 1 - 129

ReasonCode	Explanation / Action
EIRC_NO_MEMORY (105)	The MQEI LSX is unable to allocate enough memory. Method/Property: Various **Action:**Close any unwanted processes and programs are retry the operation.
EIRC_NULL_POINTER (107)	A string or object passed in an MQEI method /property call is NULL where this is not allowed. Method/Property: EIMessage methods **Action:**Correct your LotusScript program, make sure all objects and strings contain valid data.
EIRC_UNEXPECTED_ERROR (108)	An error has occurred within the MQEI LSX. Method/Property: Various **Action:**Contact your IBM/Lotus service representative.
EIRC_INVALID_UNIT_OF_WORK (110)	The UnitOfWork property in the EISendOptions object is set to a value other than one of those listed as legal values. Method/Property: EISendOptions UnitOfWork EIService SendMessage **Action:**Set the UnitOfWork property to a legal value.
EIRC_INVALID_RECEIVE_TYPE (111)	The ReceiveType property in the EIReceiveOptions object is set to a value other than one of those listed as legal values. Method/Property: EIReceiveOptionsReceiveType EIService ReceiveMessage **Action:**Set the ReceiveType property to a legal value.

ReasonCode	Explanation / Action
EIRC_INVALID_WAIT_TYPE (112)	The WaitType property in the EIReceiveOptions object is set to a value other than one of those listed as legal values. Method/Property: EIReceiveOptionsWaitType EIService ReceiveMessage **Action:**Set the WaitType property to a legal value.
EIRC_INVALID_WAIT_INTERVAL (113)	The WaitInterval property in the EIReceiveOptions object is set to a value other than one of those listed as legal values. Method/Property: EIReceiveOptions WaitInterval EIService ReceiveMessage**Action:**Set the WaitInterval to a positive numeric value or EIWI_UNLIMITED.
EIRC_INVALID_PRIORITY (115)	The value assigned to the Priority property in the EISendOptions object is invalid. Method/Property: EISendOptions Priority EIService SendMessage **Action:**Set the Priority to a positive numeric value not exceeding EIService MaxPriority or EIPRI_DEFAULT.

ReasonCode	Explanation / Action
EIRC_INVALID_DATATYPE (116)	The MQEI LSX has detected an unexpected data type for an EIMessage field. An example of such an error would be to assign a numeric value to an EIMessage field which is defined in the MQEI Definition database as data type String. This error can also occur when an EIMessage is sent or received, and the data type of a field in the EIMessage is not one supported by the type of service. For example Long, Short and Byte are not supported by a CICS 3270 direct service as all fields must be character based. Method/Property: EIMessage CreateMessage EIMessage SetFieldValue EIService SendMessage EIService ReceiveMessage **Action:**Ensure that the LotusScript program is using variables of the correct data type when assigning or accessing EIMessage field values. Ensure that the Field definitions for an EIMessage destined for a CICS 3270 direct service all have character based data types.
EIRC_INVALID_FIELD_NAME (117)	The name of the EIMessage field specified is either more than 32 characters in length or contains invalid characters. Valid characters in a name are: a-z, A-Z, 0-9, and underscore. Method/Property: EISession CreateMessage **Action:**Change the name of the EIMessage field to conform to the specified standards. If this error was generated by a utility program, this may involve changing the file used as input to the utility.

ReasonCode	Explanation / Action
EIRC_INVALID_OBJECT_NAME (118)	The name of the MQEI object does not conform to standards: An EIMessage name must not exceed 16 characters An EIService name must not exceed 16 characters Valid characters in a name are: a-z, A-Z, 0-9, - (hyphen), $ (dollar), % (percentage), * (asterisk), # (hash), & (ampersand), @ (atsign), ? (question mark), ! (exclamation mark) and _ (underscore). Method/Property: EISession CreateService EISession CreateMessage **Action:**Change the name of the object to conform to the standards.
EIRC_INVALID_LENGTH (119)	The length specified for an EIMessage field does not match its data type: Data Type Long must be 4 bytes long Data Type Short must be 2 bytes long Data Type Byte must be 1 byte long. Method/Property: EISession CreateMessage **Action:**Correct the field definition and build the new message document.
EIRC_DEFN_ALREADY_EXISTS (124)	The MQEI Message definition cannot be written to the MQEI Definition database because one with the same name already exists. Note that the name of a definition is **not** case sensitive. Method/Property: Used by utilities only **Action:**Select the replace option if the definitions are for the same message. Otherwise either change the name of the existing definition on the database, or the label of the DFHMDI macro in the BMS map source.

ReasonCode	Explanation / Action
EIRC_DEFN_NOT_FOUND (125)	The definition requested from the MQEI Definition database, to build an EIService or EIMessage object, cannot be found. Method/Property: EISession CreateService EISession CreateMessage **Action:**Correct the name of the parameter in the method call or create a definition on the MQEI Definition database.
EIRC_DATA_OVERFLOW (127)	A field in a message has been set outside the range allowed by the data type of the field. Method/Property: EIMessage Set properties EIMessage SetFieldValue method **Action:**Check the MQEI Definition database for the data type of the field and check your LotusScript is correct.
EIRC_FIELD_TRUNCATED (128)	Warning. Your LotusScript program has set a field in an EIMessage where the field is defined to be shorter than the length of the data supplied. Method/Property: EIMessage methods **Action:**Ensure your LotusScript program caters for the defined length of the field in the EIMessage.
EIRC_INVALID_DB_NAME (129)	The length of the name of the path, together with the name of the MQEI Definition database, or MQEI Security database, is too long. Method/Property: new EISession **Action:**Ensure that the length of the database name plus the path does not exceed 48 characters.

Reason codes 130 - 999

ReasonCode	Explanation / Action
EIRC_DEFN_DB_NOT_OPEN (132)	The EIMessage or EIService object cannot be created because the MQEI Definition database is closed. Method/Property: EISession CreateService • EISession CreateMessage **Action:** Your LotusScript program has continued after an error during the creation of the EISession object. Check your LotusScript program. It is important to check the CompletionCode and ReasonCode properties of the EISession and only continue when the EISession object has been created successfully. One common error giving rise to this reason code is that your mqei.ini file could not be opened. Check that your MQEI_INI_PATH is correct and that there is an mqei.ini file in this directory.

ReasonCode	Explanation / Action
EIRC_INVALID_CONNECTION (134)	The value assigned to the InboundConnection or OutboundConnection property in the EIService object is invalid. Alternatively, the InboundConnection or OutboundConnection was omitted, or was not unique for those service types that require separate inbound and outbound connections. This error can also arise when the LotusScript program sends a message with a message type of EIMT_REPLY, and the inbound EIMT_REQUEST message to which this is replying contained an MQSeries reply to queue name that does not match the EIService OutboundConnection property. Method/Property: EISession CreateService EIService Connect EIService SendMessage **Action:**Ensure a valid string of the correct length is specified in the MQEI Service definition, held in the MQEI Definition database. Ensure the connection names are specified, and unique when necessary. The maximum length for the EIService you are using is held in the ConnectionLength property in the EIService object. Ensure that a Native MQSeries service running on your enterprise system does not specify a reply to queue name that differs from the EIService OutboundConnection, when it sends a message to the MQEI application.

ReasonCode	Explanation / Action
EIRC_INVALID_IDENTIFIER (135)	The value assigned to the Identifier property in the EISendOptions or EIReceiveOptions object is invalid. Method/Property: EISendOptions Identifier Property EIReceiveOptions Identifier Property EIService SendMessage EIService ReceiveMessage **Action:**Ensure a valid hexadecimal string of the correct length is specified. The maximum length for the EIService you are using is held in the EIService IdentifierLength property.
EIRC_INVALID_ATTENTION_ID (136)	The value assigned to the AttentionId property in the EISendOptions object is not included in the legal values list. Method/Property: EISendOptions AttentionId EIService SendMessage **Action:**Set the AttentionId property to a legal value.
EIRC_INVALID_SELECTED_FIELD (137)	The value assigned to the SelectedField property in the EISendOptions object does not match any field name in the message. Method/Property: EISendOptions SelectedField EIService SendMessage **Action:**Set the SelectedField property to the name of a field in the message. If you do not know the name of the field, but you do know the position of the field, use the EIMessage GetFieldName method to obtain the name.

ReasonCode	Explanation / Action
EIRC_DEFN_DB_SYSERROR (138)	An error was signalled by Notes when opening, closing, reading from, or writing to, the MQEI Definition database. The error was such that MQEI does not or is not able to translate it into a separate MQEI reason code. More information on the error can be found in the EISession PrimarySystemErrorCode property and SystemErrorText property. Method/Property: new EISession EISession CreateService EISession CreateMessage **Action:**Establish the cause of the error with the help of the SystemErrorText and PrimarySystemErrorCode, correct the error and retry the operation. The Notes error codes are defined by the Lotus Notes C API and each has an associated message that is retrieved from Notes (using the OSLOadString Notes C API function) by the MQEI LSX, and recorded in SystemErrorText and in the MQEI LSX trace file.
EIRC_PROTECTED_FIELD (139)	An attempt to set a value in an EIMessage field has failed because the field is protected. This occurs if the enterprise system has designated fields as protected, or if the field has a data type of SYSTEM_AUTHENTICATOR. Method/Property: EIMessage SetFieldValue EIService SendMessage **Action:**Change your LotusScript program so that it doesn't set protected fields. To check if a field is protected use the EIMessage GetProtection method. If EIPRO_PROTECTED is returned, the field is protected. Also ensure that where a message maps to a terminal screen, the EIMessage object sent is the same as the EIMessage object that was last received to ensure field protection designated by the enterprise is honoured.

ReasonCode	Explanation / Action
EIRC_INVALID_SERVICE_TYPE (140)	An invalid value is specified for the ServiceType property in the EIService object. Method/Property: EISession CreateService **Action:**Correct the MQEI Service definition in the MQEI Definition database.
EIRC_NOT_SUPPORTED (141)	There are several possible reasons for this. • An attempt was made to connect to a connection manager that is not at a sufficient level to support the MQEI. • An attempt was made to create an EIService object for a service type that is not supported from the environment you are running Notes under. For example, CICS 3270 direct services are not supported under Windows 3.1 or UNIX systems, and CICS DPL direct services are not supported under UNIX systems. • It is not possible to create an EIService object for a CICS 3270 direct service and an EIService object for a CICS DPL direct service at the same time in the same Notes client or Domino 4.5 Server agent. This is a CICS restriction. • A message was received that makes use of one or more features in the target service that are not supported by the current version of MQEI. An example of this is receiving a Read Buffer message from a CICS 3270 direct service.

ReasonCode	Explanation / Action
	Method/Property: new EISession EISession CreateService EIService Connect EIServiceReceiveMessage **Action:**Depending on the cause, either upgrade the connection manager to the required level, modify the target service so that it only uses features that are supported by the MQEI. Ensure that the correct service types are being specified. Do not run CICS 3270 direct and CICS DPL direct services in the same Notes client or Domino Server agent at the same time, or use a platform that supports the type of service you are creating.
EIRC_INVALID_SERVICE_STEP (142)	An invalid value was specified for the EIService ServiceStep property. Method/Property: EISession CreateService EIService ServiceStep EIService SendMessage **Action:**Ensure a valid string of the correct length is specified in the MQEI Service definition in the MQEI Definition database, or in the LotusScript program. The maximum length for the EIService in question is given by the EIService ServiceStepLength property. Additionally, some drivers require that a ServiceStep must always be present and a value of EISS_NOT_APPLICABLE is not allowed.
EIRC_INVALID_MESSAGE_TYPE (143)	The value assigned to the MessageType property in the EISendOptions object is not included in the legal values list. Method/Property: EISendOptions MessageType EIService SendMessage **Action:**Set the MessageType property to a legal value.

ReasonCode	Explanation / Action
EIRC_INVALID_ENCODING (144)	An invalid value was specified for the system encoding in the MQEI initialization file or for the EIService Encoding property. Method/Property: new EISession EISession CreateService **Action:**Ensure a valid value is specified in the mqei.ini file or in the MQEI Service definition in the MQEI Definition database.
EIRC_INVALID_ALIGNMENT (145)	An invalid value was specified for the Alignment property of an EIMessage field. Method/Property: EISession CreateMessage **Action:**Ensure a valid value is specified for Alignment in the MQEI Definition database and that the MQEI Message definition concerned is rebuilt.
EIRC_INVALID_SEGMENT (146)	The segment number specified for an EIMessage field is invalid. The segment number of the first field in the message must be 1. The segment number of a subsequent field must equal, or be exactly 1 greater than, that of the immediately preceding field. Method/Property: EISession CreateMessage **Action:**Ensure that the rules for segment numbers are obeyed for all fields in the message and that the MQEI Message definition concerned is rebuilt.
EIRC_INI_OPEN_ERROR (147)	An error occurred opening the MQEI initialization file, mqei.ini Method/Property: new EISession **Action:**Ensure that the mqei.ini file exists, is not read protected, and is pointed at by environment variable MQEI_INI_PATH.

ReasonCode	Explanation / Action
EIRC_INVALID_SERVICE_CONTEXT (148)	An invalid value was specified for the EIService ServiceContext property. Method/Property: EISession CreateService EIService ServiceContext EIService SendMessage **Action:**Ensure a valid string of the correct length is specified in the MQEI Service definition in the MQEI Definition database, or in the LotusScript program. The maximum length for the EIService in question is given by the EIService ServiceContextLength property.
EIRC_INVALID_DELIVERY (149)	The value assigned to the Delivery property in the EISendOptions object is either not included in the legal values list, or is not valid for the outbound connection being used. Method/Property: EISendOptions Delivery EIService SendMessage **Action:**Set the Delivery property to a legal value that is acceptable to the outbound connection being used. One cause of this reason code is specifying a Delivery of EIDEL_ASSURED for a message destined for an MQSeries temporary dynamic queue. Such a queue can not contain assured delivery (that is, persistent) messages.

Reason codes 1000 - 13999

ReasonCode	Explanation / Action
EIRC_INI_SECTION_NOT_FOUND (1004)	An error has occurred locating a mandatory section in the MQEI initialization file (mqei.ini). Method/Property: new EISession **Action:**Ensure the mqei.ini follows the required syntax, all sections are present, and all section names are spelt correctly.
EIRC_INI_KEYWORD_NOT_FOUND (1005)	An error has occurred locating a mandatory keyword in the MQEI initialization file (mqei.ini). Method/Property: new EISession **Action:**Ensure the mqei.ini follows the required syntax, all keywords are present, and all keywords are spelt correctly.
EIRC_INI_VALUE_NOT_SPECIFIED (1006)	An error has occurred locating a mandatory keyword value in the MQEI initialization file (mqei.ini). Method/Property: new EISession **Action:**Ensure the mqei.ini follows the required syntax and all keyword values are present.
EIRC_INVALID_DEFN_VALUE (5004)	When reading the MQEI Definition database to build an EIService or EIMessage object, the requested definition contained an invalid property value. Method/Property: EISession CreateService EISession CreateMessage **Action:**Ensure that all values in the appropriate MQEI Service definition, MQEI Message definition and MQEI Security definition are valid.

ReasonCode	Explanation / Action
EIRC_DEFN_PROPERTY_NOT_FOUND (5009)	When reading the MQEI Definition database to build an EIService or EIMessage object, the requested definition did not contain a required property. Method/Property: EISession CreateService EISession CreateMessage **Action:**Ensure that all properties in the appropriate MQEI Service, MQEI Message, or MQEI Security definition are present.
EIRC_INVALID_HAS_ATTRIBUTES (5018)	An invalid value was specified for the HasAttributes property of an EIMessage field. Method/Property: EISession CreateMessage **Action:**Ensure a valid value is specified in the definition for the field in the MQEI Definition database and that the MQEI Message definition concerned is rebuilt.
EIRC_INVALID_PAD_CHARACTER (5019)	An invalid value was specified for the PadCharacter property of an EIMessage field. Method/Property: EISession CreateMessage **Action:**Ensure a valid value is specified in the definition for the field in the MQEI Definition database and that the MQEI Message definition concerned is rebuilt.
EIRC_DEFN_DB_UNKNOWN (5020)	The MQEI Definition database named in the MQEI initialization file (mqei.ini) is not known to Notes. Method/Property: new EISession **Action:**Ensure that the database exists and the name specified in the mqei.ini file is correct. Check that the name of any server you need to access is correctly specified in the mqei.ini file.

ReasonCode	Explanation / Action
EIRC_DEFN_DB_SECURITY_FAILURE (5021)	The user is not authorized to access the MQEI Definition database named in the mqei.ini file. Method/Property: new EISession EISession CreateService **Action:**Ensure the user (if your MQEI application runs under a Notes client) or agent owner (if your MQEI application runs under an agent on a Domino server) has sufficient authority to access the database,the server, and the definitions on the database.
EIRC_DUPLICATE_DEFN (5022)	Two or more definitions with an identical name were detected while accessing the MQEI Definition database. All definitions of the same type must have a unique name. Method/Property: EISession CreateService EISession CreateMessage EIService SendMessage **Action:**Ensure that each MQEI Service definition within the MQEI Definition database has a unique Service Name. Ensure that each Message definition within the MQEI Definition database has a unique Message Name and that all messages are rebuilt.
EIRC_DEFN_DB_INCOMPATIBLE (5024)	The versions of the Notes MQEI Definition database and MQEI software you are using are not compatible. Method/Property: EISession CreateService EISession CreateMessage **Action:**Update your database design to the level that matches your MQEI software.

ReasonCode	Explanation / Action
EIRC_DEFN_DB_UNAVAILABLE (5025)	The MQEI Definition database server is not responding. Method/Property: new EISession **Action:**Ensure the database server is started and can can be accessed from the client.
EIRC_SEC_DB_NOT_OPEN (13003)	The EIService object cannot be created because the MQEI Security database is closed. Method/Property: EISession CreateService **Action:**Your LotusScript program has continued after an error during the creation of the EISession object. Check your LotusScript program. It is important to check the CompletionCode and ReasonCode properties of the EISession and only continue when the EISession object has been created successfully.

ReasonCode	Explanation / Action
EIRC_SEC_DB_SYSERROR (13005)	An error was signalled by Notes when opening, closing, or reading from the MQEI Security database. The error was such that MQEI does not or is not able to translate it into a separate MQEI reason code. More information on the error can be found in the EISession PrimarySystemErrorCode property and SystemErrorText property. Method/Property: new EISession EISession CreateService **Action:**Establish the cause of the error with the help of SystemErrorText and PrimarySystemErrorCode, correct the error and retry the operation. The Notes error codes are defined by the Lotus Notes C API and each has an associated message that is retrieved from Notes (using the OSLOadString Notes C API function) by the MQEI LSX, and recorded in SystemErrorText and an MQEI LSX trace file.
EIRC_SEC_DB_UNKNOWN (13006)	The MQEI Security database named in the MQEI initialization file (mqei.ini) is not known to Notes. Method/Property: new EISession **Action:**Ensure that the database exists and the name specified in the mqei.ini file is correct. Check that the name of any server you need to access is correctly specified in the mqei.ini file.

ReasonCode	Explanation / Action
EIRC_SEC_DB_SECURITY_FAILURE (13007)	The user is not authorized to access the MQEI Security database named in the mqei.ini file. Method/Property: new EISession EISession CreateService **Action:**Ensure the user (if your MQEI application runs under a Notes client) or agent owner (if your MQEI application runs under an agent on a Domino server) has sufficient authority to access the database, server and the definitions on the database. This error may also occur if decryption of the MQEI Security definition fails. Check the correct Notes user.id file is being used.
EIRC_SEC_DB_INCOMPATIBLE (13009)	The versions of the MQEI Security database and MQEI software you are using are not compatible. Method/Property: EISession CreateService **Action:**Update your database design to the level that matches your MQEI software.
EIRC_SEC_DB_UNAVAILABLE (13010)	The MQEI Security database server is not responding. Method/Property: new EISession **Action:** Ensure the database server is started and can be accessed from the client.

Reason codes 14000 - 24999

ReasonCode	Explanation / Action
EIRC_CONN_MANAGER_UNKNOWN (16003)	The attempt to connect to the connection manager failed. The name in the EIService ConnectionManager property is not known to CICS or MQSeries. Method/Property: EIService Connect **Action:**Ensure the MQEI Service definition in the MQEI Definition database includes the name of a known MQSeries queue manager or CICS server.
EIRC_CONN_MANAGER_UNAVAILABLE (16004)	The attempt to connect to the connection manager named in the ConnectionManager property of the EIService object has failed. The connection manager is not currently up and running. Method/Property: EIService Connect **Action:**Ensure the appropriate MQSeries or CICS system is up and running. If you are using an MQSeries client, check the client is up and running, and the queue manager is running. If you are not using an MQSeries client, check the queue manager is running. If you are using a CICS client, check that the client is up and running and that the CICS server is up and running. If you are not using a CICS client, check the server is running.

ReasonCode	Explanation / Action
EIRC_SERVICE_SYSERROR (16005)	An attempt to connect, disconnect, send an EIMessage, or receive a message has failed. The connection manager has identified that there is a problem. More information can be found in the PrimarySystemErrorCode property, SecondarySystemErrorCode property, and SystemErrorText property of the object in which the error occurred. Method/Property: EIService Connect EIService Disconnect EIService SendMessage EIService ReceiveMessage **Action:**For a CICS direct service (where MQSeries is not used), refer to the CICS Family Client/Server Programming manual. For CICS DPL direct, PrimarySystemErrorCode will contain a CICS ECI return code, and SecondarySystemErrorCode will contain an ECI_PARMS eci_sys_return_code. For CICS 3270 direct, PrimarySystemErrorCode will contain a CICS EPI return code, and SecondarySystemErrorCode will contain a CICS_EpiSysError_t Cause.

ReasonCode	Explanation / Action
	For an MQSeries service, refer to the MQSeries Application Programming Reference manual or MQSeries-CICS/ESA DPL bridge User Guide, as appropriate. For Native MQSeries, PrimarySystemErrorCode will contain an MQSeries reason code, and SecondarySystemErrorCode will not be set. For IMS via MQSeries, PrimarySystemErrorCode will contain an MQSeries reason code or an IMS bridge specific MQSeries feedback code, and SecondarySystemErrorCode will not be set. For CICS DPL via MQSeries, PrimarySystemErrorCode will contain an MQSeries reason code or a CICS DPL bridge specific return code. If the latter, SecondarySystemErrorCode will contain either an MQSeries reason code, a CICS DPL bridge specific MQSeries feedback code, or a CICS DFHRESP code, depending on the error. Otherwise SecondarySystemErrorCode will not be set.

ReasonCode	Explanation / Action
EIRC_CALL_SEQUENCE_ERROR (16006)	An attempt was made to send or receive a message, or disconnect from a service, but the operation is not in the correct sequence for the target service. There are several possible reasons for this. • With certain services, eg, CICS 3270 direct and CICS DPL direct, messages must be sent and received in a particular sequence. Your LotusScript program must understand the order in which messages are sent and received by the service and issue EIService SendMessage and ReceiveMessage calls accordingly. • For CICS 3270 direct, a disconnect is not allowed while unreceived messages are outstanding or the target CICS transaction is still running. • A SendMessage that specifies an EISendOptions MessageType of EIMT_REPLY must be preceded by a ReceiveMessage that obtained a message with an EIReceiveOptions MessageType of EIMT_REQUEST.

ReasonCode	Explanation / Action
	• A ReceiveMessage that specifies an EIReceiveOptions ReceiveType of EIRT_RECEIVE must precede a ReceiveMessage with an EIReceiveOptions ReceiveType property of EIRT_RETURN. Method/Property: EIService SendMessage EIService ReceiveMessage EIService Disconnect **Action:**Correct your program, ensuring the rules of the conversation are followed, and that: A ReceiveMessage with a ReceiveType of EIRT_RECEIVE always precedes one with EIRT_RETURN. A message with a MessageType of EIMT_REQUEST must have been received before a SendMessage specifying a MessageType of EIMT_REPLY can be issued.
EIRC_MESSAGE_TOO_LONG (16007)	An attempt to send an EIMessage has failed because the message is too long for the connection. This may be an MQSeries queue or queue manager restriction, an IMS message segment length restriction, or a CICS commarea restriction. Method/Property: EIService SendMessage **Action:**If possible, increase the size of message allowed by the particular connection. To do this on MQSeries, change the attribute, on either or both, for the queue or the queue manager. Alternatively, reduce the size of the message or split the message to meet the size restrictions.

ReasonCode	Explanation / Action
EIRC_WRONG_IDENTIFIER (16008)	An attempt made to send or receive an EIMessage has failed because there is a problem with the Identifier. The Identifier is used by some types of services to link messages that make up a conversation. When the Identifier supplied does not match the conversation, where this can be detected, this error occurs. Method/Property: EIService SendMessage EIService ReceiveMessage **Action:**Ensure the Identifier returned in the EISendOptions after the successful completion of the first EIService SendMessage call in a conversation is used on subsequent SendMessage calls. Where specific messages need to be received, use the same Identifier in EIReceiveOptions on the ReceiveMessage calls. If a single EISendOptions object is being used for several concurrent conversations, remember to reset the Identifier property in the EISendOptions at the start of each conversation.

ReasonCode	Explanation / Action
EIRC_NO_MESSAGE_AVAILABLE (16009)	Warning. Your LotusScript program has tried to receive an enterprise message, but there is no message there. The target enterprise system may not have sent the message yet, or the target system may have encountered an error or be queuing work. Method/Property: EIService ReceiveMessage **Action:**Delay for a short period of time and retry the ReceiveMessage call. Check for any problems with the target system if the message still does not arrive. Check that your LotusScript program has specified the correct EIReceiveOptions properties, particularly the Identifier property. Also check that any message sent earlier had the correct EISendOptions Identifier and UnitofWork properties.
EIRC_SERVICE_SYSABEND (16010)	A message received indicates the last service step invoked terminated with an abend. The abend code is held in the EIService AbendCode property. This applies to CICS DPL direct, CICS DPL via MQSeries, and CICS 3270 direct services only. Method/Property: EIService SendMessage EIService ReceiveMessage EIService Disconnect **Action:**Establish the cause of the abend. Refer to the Messages and Codes manual for your CICS system to lookup abend codes. Fix the error and retry.

ReasonCode	Explanation / Action
EIRC_SERVICE_SECURITY_FAILURE (16011)	You do not have sufficient authority. The action causing the error is one of connecting to a connection manager, opening an inbound or outbound connection, authentication on the target system, or accessing resources on the target system. Method/Property: EIService Connect EIService SendMessage EIService ReceiveMessage **Action:**Ensure you have the authority to connect to the connection manager, open connections and access resources on the target system. Check your UserId and Authenticator. They may be wrong on the MQEI Security database, or you may have entered them incorrectly when you ran your application.If you are using the MQEI Security database, check the correct definition exists. If your MQEI application runs under a Notes client, there should be a definition with the EIService SystemName of your Notes user name in it. If your MQEI application runs under an agent on a Domino server, there should be a defintion with the EIService SystemName and the server's name.
EIRC_UOW_ROLLED_BACK (16012)	The CICS server has been unable to process an EIMessage sent via the CICS DPL direct service. The CICS server terminated abnormally or could not commit the current unit of work. The CICS system has backed out all the changes to recoverable resources. Method/Property: EIService ReceiveMessage **Action:**Restart the CICS server and retry.

ReasonCode	Explanation / Action
EIRC_UOW_IN_DOUBT (16013)	The CICS server has been unable to process an EIMessage sent via the CICS DPL direct service. The CICS server terminated abnormally in an indoubt state. The MQEI is unable to determine whether or not the current unit of work has been committed or backed out. Method/Property: EIService ReceiveMessage **Action:**Contact your CICS System programmer and initiate procedures to resolve the indoubt situation before restarting the CICS server. If the unit of work was backed out, retry after restarting the CICS server.
EIRC_UNRECOGNIZED_REPLY (16018)	The MQEI has detected that the message received is not part of the current conversation. This is likely to be caused where an inbound connection is shared by several users or applications and an incorrect EIReceiveOptions Identifier property has been specified, perhaps set to EIID_NONE and you are getting the first message on the inbound connection, which is not part of the conversation. Method/Property: EIService ReceiveMessage **Action:**Ensure that where connections are shared, a unique Identifier is used for each conversation that may be in progress. Avoid the use of an EIReceiveOptions Identifier of EIID_NONE that will normally return the first or highest priority message on the inbound connection. Instead specify the Identifier returned in EISendOptions by the EIService SendMessage call that started the conversation.

ReasonCode	Explanation / Action
EIRC_DATA_CONVERSION_ERROR (16019)	An error occurred converting character data in the EIMessage fields to the code page required. The required code page could not be located. Method/Property: EIService ReceiveMessage EIService SendMessage **Action:**Ensure the code page for the system is present. If you do not have the code page you need, contact your IBM/Lotus service representative.
EI_CONNECTION_DOWN (16022)	The connection to the connection manager is not available. It is no longer possible to send a message, receive a message, or disconnect from the service. Method/Property: EIServiceSendMessage EIServiceReceiveMessage EIService Disconnect **Action:**Check the inbound and outbound connections, you may need to restart the connection manager. Retry the operation.
EIRC_CONNECTION_UNKNOWN (16023)	The attempt to connect to the connection manager failed. The EIService InboundConnection or OutboundConnection property does not contain the name of a known connection. Method/Property: EIService Connect **Action:** Ensure the MQEI Service definition in the MQEI Definition database includes the names of known MQSeries queues or CICS terminal models.

ReasonCode	Explanation / Action
EIRC_WRONG_MESSAGE (16026)	Warning. The received message does not match the EIMessage object passed as the parameter on the EIService ReceiveMessage call. Method/Property: EIService ReceiveMessage **Action:**If the message could be one of several formats, retry the ReceiveMessage call with an alternative EIMessage object until a match is made, ensuring EIReceiveOptions ReceiveType is set to EIRT_RETURN. See Variant Messages for more information. Alternatively, check that the MQEI Message definition on the database is correct, paying particular attention to field lengths and positions.
EIRC_SERVICE_STEP_UNKNOWN (16029)	A message sent to a service cannot be processed. The program or transaction specified by the ServiceStep property in the EIService object is not known to the target system. Method/Property: EIService ReceiveMessage **Action:**Ensure the ServiceStep property is correct in the MQEI Service definition in the MQEI Definition database, or that your LotusScript program is setting it correctly. Ensure the program name or transaction is known to the system receiving the message.

ReasonCode	Explanation / Action
EIRC_WRONG_DATA_CONVERSION (16048)	An incorrect value was supplied for the EIService CharacterSet property or Encoding property or both. The MQEI LSX converted an outbound message into the CharacterSet and Encoding specified by the EIService object, but upon receipt of the message the enterprise service detected that these were not the CharacterSet and Encoding expected. Method/Property: EIService ReceiveMessage **Action:**Ensure the correct values are specified for the Encoding and CharacterSet properties of the enterprise service in the MQEI Definition database.
EIRC_WRONG_MESSAGE_SENT (16049)	An attempt was made to send a message to an enterprise service, but the EIMessage parameter passed on the SendMessage call was not correct. This can occur for a CICS 3270 direct service when the EIMessage parameter does not match the internal image of the terminal screen maintained by the EIService object. Method/Property: EIService SendMessage **Action:**Ensure the correct EIMessage parameter is passed. If the terminal screen is not empty, the EIMessage should be that used on the last ReceiveMessage call. If the EIMessage being passed is that intended, check that the corresponding MQEI Message definition is correct, paying particular attention to field lengths and positions.

ReasonCode	Explanation / Action
EIRC_INSUFFICIENT_DATA (16051)	Warning. A message was received that matches the EIMessage object passed as a parameter on the EIService ReceiveMessage call, except that the length of the message data is insufficient to fully complete the last EIMessage field. All EIMessage fields except the last are fully completed, but the last field is only partly completed from the available data. Method/Property: EIService ReceiveMessage **Action:**If the message you are receiving is expected to contain a varying length string at the end where the length is unknown, this reason code indicates that the message has been successfully received. It is your responsibility in your LotusScript program to interpret the last field in the returned EIMessage object. Note this warning can be avoided if you define the last field in the message as a variable length string instaed of a fixed length string, in the MQEI Definition database. For more information, see "Varying Length Messages" in Chapter 4. If the message you are receiving has a fixed format, then this reason code probably indicates that the wrong EIMessage object has been passed on the ReceiveMessage call. You should treat this reason code exactly as you would treat EIRC_WRONG_MESSAGE, namely retry the ReceiveMessage call with a alternative EIMessage object until a match is made, ensuring that EIReceiveOptions ReceiveType property is set to EIRT_RETURN.

ReasonCode	Explanation / Action
EIRC_SERVICE_CONTEXT_UNKNOWN (16053)	A message sent to a service can not be processed. The service context information specified by the EIService ServiceContext property is not understood by the target system. Method/Property: EIService ReceiveMessage **Action:**Ensure that the ServiceContext property is correct in the MQEI Service definition in the MQEI Definition database, or that your LotusScript program is setting it correctly. Ensure the service context information is known on the system that receives the message. For a service of type CICS DPL direct or CICS DPL via MQSeries, this involves checking that the CICS transid specified by the ServiceContext is defined on the target CICS system.

Reason codes 25000 - 45000

ReasonCode	Explanation / Action
EIRC_BMS_INVALID_FILENAME (25001)	When running the BMS map conversion utility MQEIBMS, the BMS filename specifed as a parameter did not have a file extension of **.**bms Method/Property Used by utilities only **Action:**Specify a BMS filename with an extension of **.**bms when running the MQEIBMS utility.
EIRC_BMS_NO_FILENAME (25002)	When running the BMS map conversion utility MQEIBMS, the BMS filename parameter was omitted. Method/Property Used by utilities only **Action:**Specify a BMS filename with an extension of **.**bms when running the MQEIBMS utility.
EIRC_BMS_OPEN_ERROR (25003)	When running the BMS map conversion utility MQEIBMS, an error occurred opening the BMS file specified as a parameter of the utility. Method/Property Used by utilities only **Action:**Ensure the file exists in the correct directory, and that the filename is spelt correctly when you run the MQEIBMS utility.
EIRC_BMS_UNNAMED_MESSAGE (25007)	When running the BMS map conversion utility MQEIBMS, a BMS map was encountered that does not have a name. This is because no label was specified for the DFHMDI macro for the map in question. An MQEI Message definition can not be created as each MQEI Message definition must have a unique name, and the name is obtained from the DFHMDI macro label. Method/Property: Used by utilities only **Action:**Edit the BMS mapset file, locate the DFHMDI macro for the map in question, and supply a label.

ReasonCode	Explanation / Action
EIRC_INVALID_DB_TYPE (41008)	An invalid database type was specified for either the MQEI Definition database or the MQEI Security database in the MQEI initialization file (mqei.ini). Method/Property: new EISession **Action:**Correct the mqei.ini file, using a legal value for database type.
EIRC_INVALID_CHARACTER_SET (41011)	An invalid value was specified for the system character set in the MQEI initialization file (mqei.ini) or for the EIService CharacterSet property. Method/Property: new EISession EISession CreateService **Action:**.Ensure a valid value is specified in the mqei.ini file or in the MQEI Service definition in the MQEI Definition database.
EIRC_CONNECTED (42002)	Warning. An attempt to connect to a service has been rejected because the service is already connected. The connect is ignored and your LotusScript program continues. Method/Property: EIService Connect Method **Action:**No action if you intended this to happen, however you may have a logic error in your LotusScript program.

ReasonCode	Explanation / Action
EIRC_NOT_CONNECTED (42003)	A service is not connected and your application has tried to send a message, receive a message, or disconnect from a service. Method/Property: EIService SendMessage EIService ReceiveMessage EIService Disconnect **Action:**Check your LotusScript program. An EIService Connect call must be made and be successful before your program can send or receive messages, or disconnect from a service. It is important to check CompletionCode and ReasonCode properties after the Connect call.
EIRC_INVALID_AUTHENTICATOR (42006)	An invalid value was specified for the EIService Authenticator property. Method/Property: EISession CreateService EIService Authenticator **Action:**Ensure a valid string of the correct length is specified in the MQEI Security definition in the MQEI Security database, or in the LotusScript program. The maximum length for the EIService in question is given by the EIService AuthenticatorLength property.
EIRC_INVALID_USERID (42007)	An invalid value was specified for the EIService UserId property. Method/Property: EISession CreateService EIService UserId **Action:**Ensure a valid string of the correct length is specified in the MQEI Security definition in the MQEI Security database, or in the LotusScript program. The maximum length for the EIService in question is given by the EIService UserIdLength property.

ReasonCode	Explanation / Action
EIRC_INVALID_SYSTEM_NAME (42016)	An invalid value was specified for the EIService SystemName property. Method/Property: EISession CreateService **Action:**Ensure a valid string of the correct length is specified in the MQEI Service definition in the MQEI Definition database. The maximum length for the EIService in question is given by the EIService SystemNameLength property.
EIRC_INVALID_CONN_MANAGER (42017)	An invalid value was specified for the EIService ConnectionManager property. Method/Property: EISession CreateService **Action:**Ensure a valid string of the correct length is specified in the MQEI Service definition in the MQEI Definition database. The maximum length for the EIService in question is given by the EIService ConnectionManagerLength property.
EIRC_DYNAMIC_LOAD_ERROR (42019)	The MQEI LSX has tried to connect to a service and been unable to load the dynamic load library (or shared library) required to support the type of service. This error normally occurs due to a system setup problem (such as default system search path not being correct) or by the search path over-ride environment variable (MQEI_xxx_LIB, where xxx is MQ,ECI or EPI) having been incorrectly specified. The SystemErrorText property of the EIService will contain details on the name of the library that could not be found. Method/Property: EIService Connect Method **Action:**Ensure that the necessary library is available either via the normal search path or via the MQEI_xxx_LIB environment variable.

ReasonCode	Explanation / Action
EIRC_FIELD_NOT_FOUND (43002)	The EIMessage field specified could not be found. Method/Property: All EIMessage methods and properties **Action:**Check the name of the EIMessage field used in your LotusScript program exists as a Field definition for the message in the MQEI Definition database. Check that the MQEI Message definition has been built.
EIRC_DUPLICATE_FIELD (43004)	The EIMessage has two fields that have the same name. Method/Property: EISession CreateMessage **Action:**Check the MQEI Message definition. Note that the names of the fields are case insensitive, e.g."FieldA" is considered to be the same as "FIELDA". Rename one of the offending fields.
EIRC_EMPTY_MESSAGE (43010)	A field in an EIMessage cannot be accessed because the EIMessage does not contain any fields. Method/Property: All EIMessage methods and properties **Action:**Check the FieldCount property of an EIMessage before accessing the fields in the message.
EIRC_INVALID_INDEX (43017)	The index used to reference a field in a EIMessage is either less than zero or greater than the number of fields in the EIMessage. Method/Property: EIMessage GetFieldName **Action:**Check the FieldCount property for the EIMessage and use a value in this range in the GetFieldName call.

ReasonCode	Explanation / Action
EIRC_INVALID_MESSAGE_FORMAT (43021)	The value assigned to the Format property in the EIMessage object is not valid. Method/Property: EIMessage Format **Action:**Ensure that a legal value, or a user-defined string of not more than 8 characters, is specified in the MQEI Message definition or in the LotusScript program.
EIRC_LICENSE_EXPIRED (99999)	You have an evaluation copy of MQEI installed, and the evaluation period has expired. Method/Property: new EISession **Action:** Obtain a fully licensed copy of MQEI.

Chapter 12 MQEI LSX Reference

This chapter describes the classes of the MQSeries Enterprise Integrator LotusScript Extension (MQEI LSX), developed for Lotus Notes Release 4.5. The classes enable you to write Notes applications that can access other applications running in your non-Notes environments, using MQSeries, CICS, IMS, or MQSeries.

Constants

All constants used by the MQEI LSX are available to your LotusScript program, displayed in the browser under Notes:Constants. There are constants for every MQEI reason code and completion code, and for many of the legal values of MQEI properties.

All MQEI reason codes start with EIRC_, for example EIRC_CONNECTION_DOWN.

All MQEI completion codes start with EICC_, for example EICC_OK.

All constants associated with a legal value for a property start with EIx_ where x is a two or three character abbreviation for the property. For example EIST_MQ, where ST is the abbreviation for ServiceType and in this case the ServiceType is MQSeries.

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 your program supplies a fixed length string of the wrong length, as an input parameter or a return value, then the value is truncated or padded as required.
- If your program supplies a variable length string of the wrong length as an input parameter, then the value is truncated or padded. Trailing NULLS are the default padding character, you change it by modifying the message format in the MQEI Definition database.
- If your program 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).

Errors

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

The MQEI LSX 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, the owning object's ReasonCode and CompletionCode fields are set to EIRC_NONE and EICC_OK respectively.

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

For more information, see "Error handling" in Chapter 4.

EISession Class

This is the root class for the MQEI LSX.

There must always be one and only one EISession object per LotusScript instance. If you attempt to create another EISession object, you are returned a reference to the original object. If you delete the EISession, the reference to the object is deleted and you have no further access to it. However, the object persists, so if you need to access it again, create a new object reference.

If you are using multi-threaded Domino Web agents, however, a new EISession object is created for each new thread.

The EISession object controls access to all the MQEI resources. It is responsible for access to, and creation of the objects that are created by the LotusScript instance:

- EIService
- EIMessage
- EIReceiveOptions
- EISendOptions

The EISession object is also responsible for opening and closing the MQEI Definition database and the MQEI Security database.

When creating an EIService or EIMessage object, that object is 'tied' to the corresponding enterprise service or message format definition on the MQEI Definition database. This definition is used to initialize the properties of that object with default values, some of which may be later overridden in a LotusScript program.

The EISession reads an initialization file (mqei.ini) which must be present. This is a simple text file that contains environment specific information such as the name of the MQEI Definition database. The EISession finds the path of the initialization file from the environment variable MQEI_INI_PATH.

Properties:

- CharacterSet property
- CompletionCode property
- DefinitionDBName property
- PrimarySystemErrorCode property
- ReasonCode property
- SecondarySystemErrorCode property
- SecurityDBName property
- SystemErrorText property

Methods:

- ClearErrorCodes method
- CreateMessage method
- CreateReceiveOptions method
- CreateSendOptions method
- CreateService method

LotusScript Events:

- EIError
- EIWarning

Creation:

New - creates a new EISession object reference.

Reason codes:
EIRC_INVALID_DB_NAME

EIRC_DEFN_DB_SYSERROR

EIRC_NOT_SUPPORTED

EIRC_INI_OPEN_ERROR

EIRC_INI_SECTION_NOT_FOUND

EIRC_INI_KEYWORD_NOT_FOUND

EIRC_INI_VALUE_NOT_SPECIFIED

EIRC_DEFN_DB_UNKNOWN

EIRC_DEFN_DB_SECURITY_FAILURE

EIRC_SEC_DB_SYSERROR

EIRC_SEC_DB_UNKNOWN

EIRC_SEC_DB_SECURITY_FAILURE

EIRC_INVALID_DB_TYPE

EIRC_INVALID_ENCODING

EIRC_INVALID_CHARACTER_SET

Syntax:
Dim *eisess* **As New EISession** or

Set *eisess* = **New EISession**

CharacterSet Property

Read-only. Specifies the character set (code page) of the local system on which the MQEI LSX is running. The MQEI LSX automatically converts character data to, or from, the indicated setting when data is received from, or sent to, the enterprise service. The value is set from the mqei.ini file when the EISession object is created.

Defined in:
EISession Class

Data Type:
Long

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

CompletionCode Property

Read-only. Returns the MQEI LSX completion code set by the most recent method or property access issued against the object. It is reset to EICC_OK when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EISession Class

Data Type:
Long

Legal Values:
- EICC_OK
- EICC_WARNING
- EICC_FAILED

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

DefinitionDBName Property

Read-only. Returns the name of the MQEI Definition database. The value for this property is obtained from the mqei.ini file.

Defined in:
EISession Class

Data Type:
String of 48 characters

Syntax:
To get: *definitiondbname$ = EISession*.**DefinitionDBName**

PrimarySystemErrorCode Property

Read-only. Returns any primary external error code set by the most recent method or property access issued against the object. It has a meaning only when the CompletionCode property is set to EICC_FAILED and the ReasonCode property is set to EIRC_DEFN_DB_SYSERROR or EIRC_SEC_DB_SYSERROR, indicating an error occurred accessing the MQEI Definition database or MQEI Security database.

An example of a PrimarySystemErrorCode is a Notes C API error code.

Defined in:
EISession Class

Data Type:
Long

Legal Values:
Refer to the appropriate Notes documentation, but note that a text description may also be held in the SystemErrorText property .

Syntax:
To get: *primarysystemerrorcode& = EISession*.**PrimarySystemErrorCode**

ReasonCode Property

Read-only. Returns the MQEI LSX reason code set by the most recent method or property access issued against the object. It is reset to EIRC_NONE when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes Method is called.

Defined in:
EISession Class

Data Type:
Long

Legal Values:
See the list of possible reason codes listed under Reason Codes in Chapter 11.

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

SecondarySystemErrorCode Property

Read-only. Returns any secondary enterprise system error code set by the most recent method or property access issued against any MQEI LSX object. It has a meaning only when the PrimarySystemErrorCode property is such that a secondary error code is relevant.

Defined in:
EISession Class

Data Type:
Long

Legal Values:
Refer to the appropriate Notes documentation.

Syntax:
To get: *secondarysystemerrorcode& = EISession*.**SecondarySystemErrorCode**

SecurityDBName Property

Read-only. Returns the name of the MQEI Security database. The value for this property is obtained from the mqei.ini file.

Defined in:
EISession Class

Data Type:
String of 48 characters

Syntax:
To get: *securitydbname$ = EISession*.**SecurityDBName**

SystemErrorText Property

Read-only. Returns any explanatory error text set by the most recent method or property access issued against the object. It has a meaning only when the CompletionCode property is set to EICC_FAILED and the ReasonCode property is set to EIRC_DEFN_DB_SYSERROR or EIRC_SEC_DB_SYSERROR, indicating an error occurred accessing the MQEI Definition database or MQEI Security database.

Defined in:
EISession Class

Data Type:
String of varying length

Syntax:
To get: *systemerrortext$ = EISession*.**SystemErrorText**

ClearErrorCodes Method

Resets CompletionCode property to EICC_OK, ReasonCode property to EIRC_NONE, and PrimarySystemErrorCode property and SecondarySystemErrorCode property to zero, and clears SystemErrorText property.

Defined in:
EISession Class

Syntax:
Call *EISession*.**ClearErrorCodes**

CreateMessage Method

Creates a new EIMessage object based on an MQEI Message definition on the MQEI Definition database. This is the only means by which an EIMessage object can be created.

If the named definition cannot be found or there is an error reading the MQEI Definition database, or there is insufficient memory, the object is not created and ReasonCode and CompletionCode are set.

Note Events are only raised if you have an event handler registered.

Defined in:
EISession Class

Syntax:
Set *message* = *EISession*.**CreateMessage**(*name$*)

Parameter:
name$ String. The name of the message on the MQEI Definition database.

Reason codes:
EIRC_INVALID_FIELD_NAME

EIRC_INVALID_OBJECT_NAME

EIRC_INVALID_LENGTH

EIRC_DEFN_NOT_FOUND

EIRC_INVALID_DEFN_VALUE

EIRC_DEFN_DB_NOT_OPEN

EIRC_DEFN_DB_SYSERROR

EIRC_DEFN_PROPERTY_NOT_FOUND

EIRC_INVALID_HAS_ATTRIBUTES

EIRC_INVALID_PAD_CHARACTER

EIRC_DUPLICATE_FIELD

CreateReceiveOptions Method

Creates a new EIReceiveOptions object and fills it with default values. This is the only means by which an EIReceiveOptions object can be created.

If there is a problem, an event is raised, the ReasonCode and CompletionCode are set and the object is not created.

Note Events are only raised if you have an event handler registered.

Defined in:
EISession Class

Syntax:
Set *rcvoptions = EISession*.**CreateReceiveOptions**

CreateSendOptions Method

Creates a new EISendOptions object and fills it with default values. This is the only means by which an EISendOptions object can be created.

If there is a problem, an error event is raised, the ReasonCode and CompletionCode are set and the object is not created.

Note Events are only raised if you have an event handler registered.

Defined in:
EISession Class

Syntax:
Set *sendoptions = EISession*.**CreateSendOptions**

CreateService Method

Creates a new EIService object based on an MQEI Service definition on the MQEI Definition database and, if one is being used, accesses the MQEI Security database to obtain any userid and authenticator for the target service.

This is the only means by which an EIService object can be created.

If the named definition can not be found or there is an error reading the MQEI Definition database, or there is insufficient memory, an error event is raised, the object is not created and ReasonCode and CompletionCode are set.

Note Events are only raised if you have an event handler registered.

Defined in:
EISession Class

Syntax:
Set *service* = *EISession*.**CreateService**(*name$*)

Parameter:
name$ String. The name of the enterprise service on the MQEI Definition database.

Reason codes:
EIRC_DEFN_NOT_FOUND

EIRC_INVALID_DEFN_VALUE

EIRC_DEFN_DB_NOT_OPEN

EIRC_INVALID_CONNECTION

EIRC_DEFN_DB_SYSERROR

EIRC_DUPLICATE_DEFN

EIRC_INVALID_SERVICE_STEP

EIRC_DEFN_PROPERTY_NOT_FOUND

EIRC_DEFN_DB_SECURITY_FAILURE

EIRC_SEC_DB_NOT_OPEN

EIRC_SEC_DB_SYSERROR

EIRC_SEC_DB_SECURITY_FAILURE

EIRC_INVALID_ENCODING

EIRC_INVALID_CHARACTER_SET

EIRC_INVALID_SERVICE_TYPE

EIRC_INVALID_AUTHENTICATOR

EIRC_INVALID_USERID

EIRC_INVALID_SYSTEM_NAME

EIRC_INVALID_CONN_MANAGER

EIService Class

This represents an enterprise service to which messages will be sent and received.

An EIService allows your LotusScript program to communicate with a non-Notes application or service residing on the same or a remote server. The EIService provides a common API regardless of the nature of the enterprise service involved.
An enterprise service can be:

- Native MQSeries
- IMS via MQSeries (using the MQSeries IMS bridge)
- CICS DPL via MQSeries (using the MQSeries-CICS/ESA DPL bridge)
- CICS DPL direct via CICS client
- CICS 3270 direct via CICS client

An instance of an EIService corresponds to a single enterprise service, however, a single service can consist of several programs or transactions. The EIService is responsible for:

- Establishing the connection to the enterprise system on which the service resides
- Performing any necessary authentication if supported
- Transmitting EIMessage objects of the appropriate format and receiving any replies
- Performing any data conversion
- Closing the connection

There is no limit on the number of messages that can be transmitted, this depends entirely on the enterprise service.

Your LotusScript program controls how the EIMessage is sent and received by using the EISendOptions Class and the EIReceiveOptions Class.

The EIService, EISendOptions and EIReceiveOptions classes provide the transmission functionality of the MQEI API.

An EIService can only be created via the CreateService method of the EISession object. This reads the MQEI Definition database which provides the corresponding enterprise service definition that is used to assign values to the EIService properties. You can override some of these properties at run-time.

Properties:

- AbendCode property
- Authenticator property
- AuthenticatorLength property
- CharacterSet property
- CompletionCode property
- ConnectionLength property
- ConnectionManager property
- ConnectionManagerLength property
- IdentifierLength property
- InboundConnection property
- MaxPriority property
- Name property
- OutboundConnection property
- PrimarySystemErrorCode property
- ReasonCode property
- SecondarySystemErrorCode property
- ServiceStep property
- ServiceStepLength property
- ServiceType property
- SystemErrorText property
- SystemName property
- SystemNameLength property
- UserId property
- UserIdLength property

Methods:

- ClearErrorCodes method
- Connect method
- Disconnect method
- ReceiveMessage method
- SendMessage method

LotusScript Events:

- EIError
- EIWarning

Creation:

Use the CreateService method from the EISession class.

Property Access:

When assigning values to properties, errors can occur. These cause an error event to be raised if an event handler has been registered, the CompletionCode property to be set to EICC_FAILED and an appropriate value to be set in the ReasonCode property for the EIService object.

AbendCode Property

Read-only. Returns the enterprise service abend code set by the most recent method access issued against the object. It has a meaning only when the CompletionCode property is set to EICC_FAILED and the ReasonCode property is set to EIRC_SERVICE_SYSABEND, indicating an error occurred communicating with the target enterprise service such that the executable step abended.

An example of an AbendCode is a CICS transaction abend code, such as 'ASRA'.

Defined in:
EIService Class

Data Type:
String of maximum 4 characters

Legal Values:
Refer to the appropriate product documentation for the enterprise system on which the error occurred.

Syntax:
To get: *abendcode$ = EIService*.**AbendCode**

Authenticator Property

Write-only. Specifies the password or ticket for the userid given by the UserId property. Its value is set from the MQEI Security definition for the enterprise system, if a definition is available, when the EIService object is created.

The value may be overwritten by a LotusScript program, either by prompting the person running the application or by information from some other source.

Any attempt to read the contents of this property returns the value EIAUT_HIDDEN.

An example of an authenticator is a CICS or an IMS password.

Defined in:
EIService Class

Data Type:
String of varying length, maximum length given by the AuthenticatorLength property .

Syntax:
To set: *EIService*.**Authenticator =** *authenticator$*

Reason codes:
EIRC_INVALID_AUTHENTICATOR

AuthenticatorLength Property

Read-only. Specifies the maximum length allowable for the Authenticator property, if this property is relevant to the type of enterprise service, otherwise EIAL_NOT_APPLICABLE.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *authenticatorlength& = EIService*.**AuthenticatorLength**

CharacterSet Property

Read-only. Specifies the character set (code page number) in which character data is passed to the enterprise service. The MQEI LSX automatically converts character data to the indicated setting when data is sent to the enterprise service. When data is received from the enterprise service, the MQEI LSX automatically converts character data from the indicated setting if the received message does not contain its own character set. If it does, its own character set is used.

The value is set from the MQEI Definition database when the EIService is created.

Note No character set conversion is performed if ServiceType is CICS 3270 direct as this is done automatically by CICS.

Defined in:
EIService Class

Data Type:
Long

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

CompletionCode Property

Read-only. Returns the MQEI LSX completion code set by the most recent method or property access issued against the object. It is reset to EICC_OK when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EIService Class

Data Type:
Long

Legal Values:
- EICC_OK
- EICC_WARNING
- EICC_FAILED

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

ConnectionLength Property

Read-only. Specifies the maximum length allowable for both the InboundConnection property and OutboundConnection property, if one or both of these properties are relevant to the type of enterprise service, or EICL_NOT_APPLICABLE otherwise.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *connectionlength& = EIService*.**ConnectionLength**

ConnectionManager Property

Read-only. Specifies the name of the connection manager being used to communicate with the enterprise service. This value is set from the MQEI Definition database when the EIService is created. If this property is not relevant to the enterprise service, or a default connection manager is being used, its value is EICM_NOT_APPLICABLE.

An example of a ConnectionManager is an MQSeries queue manager or the name of a CICS server as known by a CICS client.

Defined in:
EIService Class

Data Type:
String of varying length, maximum length given by the ConnectionManagerLength property

Syntax:
To get: *connectionmanager$ = EIService*.**ConnectionManager**

ConnectionManagerLength Property

Read-only. Specifies the maximum length allowable for the ConnectionManager property, if that property is relevant to the type of enterprise service, or EICML_NOT_APPLICABLE otherwise.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *connectionmanagerlength& = EIService*.**ConnectionManagerLength**

IdentifierLength Property

Read-only. Specifies the maximum length allowable for the Identifier property of the EISendOptions class and for the Identifier property of the EIReceiveOptions class, if the property is relevant to the type of enterprise service, or EIIL_NOT_APPLICABLE otherwise.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *identifierlength& = EIService*.**IdentifierLength**

InboundConnection Property

Read-only. Specifies the name of the connection being used for transmission of inbound messages, that is, from the enterprise service. This value is set from the MQEI Definition database when the EIService is created. If this property is not relevant to the enterprise service then its value is EIIC_NOT_APPLICABLE.

An example of an InboundConnection is an MQSeries reply queue.

Defined in:
EIService Class

Data Type:
String of varying length, maximum length given by the ConnectionLength property

Syntax:
To get: *inboundconnection$ = EIService*.**InboundConnection**

MaxPriority Property

Read-only. Specifies the maximum value allowable for the Priority property of the EISendOptions class, if this property is relevant to the type of enterprise service, or EIMP_NOT_APPLICABLE otherwise.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *maxpriority& = EIService*.**MaxPriority**

Name Property

Read-only. Returns the name of the enterprise service on the MQEI Definition database that the object represents.

Defined in:
EIService Class

Data Type:
String of maximum 16 characters

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

OutboundConnection Property

Read-only. Specifies the name of the connection being used for transmission of outbound messages, that is, to the enterprise service. This value is set from the MQEI Definition database when the EIService is created. If this property is not relevant to the enterprise service, or a default connection is being used, its value is EIOC_NOT_APPLICABLE.

An example of an OutboundConnection is an MQSeries application queue, or a CICS 3270 terminal model name.

Defined in:
EIService Class

Data Type:
String of varying length, maximum length given by the ConnectionLength property

Syntax:
To get: *outboundconnection$ = EIService*.**OutboundConnection**

PrimarySystemErrorCode Property

Read-only. Returns any primary enterprise system error code set by the most recent method or property access issued against the object. It has a meaning only when the CompletionCode property is set to EICC_FAILED and the ReasonCode property is set to EIRC_SERVICE_SYSERROR, indicating an error occurred communicating with the target enterprise service.

An example of a PrimarySystemErrorCode is an MQSeries reason code, a CICS ECI or EPI return code, an MQSeries IMS bridge feedback code, or an MQSeries CICS/ESA DPL bridge return code.

Defined in:
EIService Class

Data Type:
Long

Legal Values:
Refer to the appropriate product documentation for the enterprise system on which the error occurred.

Syntax:
To get: *primarysystemerrorcode& = EIService*.**PrimarySystemErrorCode**

ReasonCode Property

Read-only. Returns the MQEI LSX reason code set by the most recent method or property access issued against the object. It is reset to EIRC_NONE when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes Method is called.

Defined in:
EIService Class

Data Type:
Long

Legal Values:
See the list of possible reason codes listed under *Reason codes* in Chapter 11.

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

SecondarySystemErrorCode Property

Read-only. Returns any secondary enterprise system error code set by the most recent method or property access issued against the object. It has a meaning only when the PrimarySystemErrorCode property is such that a secondary error code is relevant.

An example of a SecondarySystemErrorCode is a CICS ECI system return code, a CICS EPI system error cause value, an MQSeries reason code, an MQSeries CICS/ESA DPL bridge feedback code, or a CICS DFHRESP code.

Defined in:
EIService Class

Data Type:
Long

Legal Values:
Refer to the appropriate product documentation for the enterprise system on which the error occurred.

Syntax:
To get: *secondarysystemerrorcode& = EIService*.**SecondarySystemErrorCode**

ServiceStep Property

Read-write. Specifies the name of the next executable step of the enterprise service, as known by the enterprise system. The value of the first step is set from the MQEI Definition database when the EIService is created. If the enterprise service comprises several such steps, this property must be updated within your LotusScript program before each SendMessage method call. If this property is not relevant to the enterprise service then its value is EISS_NOT_APPLICABLE.

An example of a ServiceStep is an IMS trancode, a CICS program name, or a CICS 3270 transid.

Defined in:
EIService Class

Data Type:
String of varying length, maximum length given by the ServiceStepLength property

Syntax:
To get: *servicestep$* = *EIService*.**ServiceStep**

To set: *EIService*.**ServiceStep** = *servicestep$*

Reason codes:
EIRC_INVALID_SERVICE_STEP

ServiceContext Property

Read-write. Specifies, for the next executable step of the enterprise service, the name of any context that the service step will execute with. The value of the first service context is set from the Definition Database when the EIService is created. If the enterprise service comprises several service steps, and each service step requires a different service context, this property must be updated before each SendMessage method call. This is the responsibility of the programmer. If this property is not relevant to the type of enterprise service, or no context is being used, then its value is EISC_NOT_APPLICABLE.

An example of a service context is the CICS transid assigned to a transaction running a CICS DPL program. In this case the ServiceStep is the CICS DPL program name, and the ServiceContext is the CICS transid. Another example of a service context is the name of a logical terminal (LTERM) passed to an IMS transaction.

Defined in:
EIService Class

Data Type:
String of varying length, maximum length given by the ServiceContextLength property

Syntax:
To get: *servicecontext$* **=** *EIService.***ServiceContext**

To set: *EIService.***ServiceContext =** *servicecontext$*

ServiceContextLength Property

Read-only. Specifies the maximum length allowable for the ServiceContext property, if that property is relevant to the type of enterprise service, or EISCL_NOT_APPLICABLE otherwise.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *servicecontextlength&* **=** *EIService.***ServiceContextLength**

ServiceStepLength Property

Read-only. Specifies the maximum length allowable for the ServiceStep property , if that property is relevant to the type of enterprise service, or EISSL_NOT_APPLICABLE otherwise.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *servicesteplength& = EIService*.**ServiceStepLength**

ServiceType Property

Read-only. Specifies the type or style of enterprise service that this EIService represents. This value is set from the MQEI Definition database when the EIService is created.

Defined in:
EIService Class

Data Type:
Long

Legal Values:

- EIST_MQ
- EIST_IMS_VIA_MQ
- EIST_CICS_DPL_VIA_MQ
- EIST_CICS_DPL_DIRECT
- EIST_CICS_3270_DIRECT

Syntax:
To get: *servicetype& = EIService*.**ServiceType**

SystemErrorText Property

Read-only. Returns any explanatory error text set by the most recent method or property access issued against the object. It has a meaning only when the CompletionCode property is set to EICC_FAILED and the ReasonCode property is set to EIRC_SERVICE_SYSERROR or EIRC_DYNAMIC_LOAD_ERROR, indicating an error occurred communicating with the target enterprise service.

Defined in:
EIService Class

Data Type:
String of varying length

Syntax:
To get: *systemerrortext$ = EISession*.**SystemErrorText**

SystemName Property

Read-only. Specifies the name of the enterprise system upon which the enterprise service resides and upon which authentication takes place.

This property is used solely to locate an MQEI Security definition in the MQEI Security database. Its value is set from the MQEI Definition database when the EIService is created. If this property is not relevant to the enterprise service then its value is EISN_NOT_APPLICABLE.

Note The SystemName is used as the key to access the MQEI Security database when the EIService is created in order to obtain any enterprise userid and authenticator for the user of the service.
You are recommended to use a system name that is understood by those who need to use it.

An example of a SystemName for CICS is a system name, applid or sysid. An example of a SystemName for IMS is an applid.

Defined in:
EIService Class

Data Type:
String of any 8 characters.

Syntax:
To get: *systemname$ = EIService*.**SystemName**

SystemNameLength Property

Read-only. Specifies the maximum length allowable for the SystemName property, if that property is relevant to the type of enterprise service, or EISNL_NOT_APPLICABLE otherwise. It is currently always set to 8.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *systemnamelength&* = *EIService*.**SystemNameLength**

UserId Property

Read-Write. Specifies the userid of the user as known by the enterprise system on which the enterprise service resides.

Its value is set from the MQEI Security definition for the enterprise system to which the connection is to be made, if a definition is available, when the EIService object is created. The authenticator associated with the userid is also held on the MQEI Security database.

If there is no MQEI Security definition, or no MQEI Security database is being used, the value is EIUI_NOT_APPLICABLE.

The value may be overwritten by a LotusScript program either by prompting the person running the application or by information from some other source.

If the value is **not** EIUI_NOT_APPLICABLE, the userid and associated authenticator are flowed along with the message by the SendMessage method when appropriate.

An example of a UserId is an IMS userid or a CICS userid.

Defined in:
EIService Class

Data Type:
String of varying length, maximum length given by the UserIdLength property

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

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

Reason codes:
EIRC_INVALID_USER_ID

UserIdLength Property

Read-only. Specifies the maximum length allowable for the UserId property, if that property is relevant to the type of enterprise service, or EIUIL_NOT_APPLICABLE otherwise.

Defined in:
EIService Class

Data Type:
Long

Syntax:
To get: *useridlength& = EIService*.**UserIdLength**

ClearErrorCodes Method

Resets CompletionCode property to EICC_OK, ReasonCode property to EIRC_NONE, and PrimarySystemErrorCode property and SecondarySystemErrorCode property to zero, and clears SystemErrorText property for this object only.

Defined in:
EIService Class

Syntax:
Call *EIService*.**ClearErrorCodes**

Connect Method

The Connect method makes a connection to the target enterprise system via the specified connection manager. Exactly what takes place depends on the enterprise service. Any required inbound and outbound connection resources are opened.

This method must succeed before the SendMessage method and ReceiveMessage method can be called.

If there is an error establishing the connection an error event is raised, and ReasonCode and CompletionCode are set.

If the Connect method has already been called, a warning event is raised, and ReasonCode and CompletionCode are set.

Note Events are only raised if you have an event handler registered.

Defined in:
EIService Class

Syntax:
Call *EIService*.**Connect**

Reason codes:
EIRC_CONN_MANAGER_UNAVAILABLE

EIRC_CONN_MANAGER_UNKNOWN

EIRC_CONNECTED

EIRC_CONNECTION_UNKNOWN

EIRC_DYNAMIC_LOAD_ERROR

EIRC_NOT_SUPPORTED

EIRC_SERVICE_SECURITY_FAILURE

EIRC_SERVICE_SYERROR

Disconnect Method

The Disconnect method destroys the connection to the enterprise service. This involves disconnecting from the connection manager, and closing any connection resources.

If there is an error destroying the connection or the Connect method has not yet been called, an error event is raised, and the ReasonCode and CompletionCode are set for this object .

Note Events are only raised if you have an event handler registered.

Defined in:
EIService Class

Syntax:
Call *EIService*.**Disconnect**

Reason codes:
EIRC_CALL_SEQUENCE_ERROR

EIRC_CONNECTION_DOWN

EIRC_NOT_CONNECTED

EIRC_SERVICE_SYERROR

ReceiveMessage Method

The ReceiveMessage method receives and returns a message, described by the EIMessage parameter, from the enterprise service. The EIReceiveOptions parameter controls exactly how the message is received and returned. If no *rcvoptions* are included in the call to this method, the default values are used.

The ReceiveMessage method can also be used when you do not want any user data returned to the program by using the reserved keyword **Nothing** instead of the message object reference. This is useful when receiving the reply from a service that contains no user application data, such as a CICS DPL commit or backout call, or when the contents of the received message are not of any interest.

An error event is raised and ReasonCode and CompletionCode are set for this object if there is an error receiving the message, the Connect method has not yet been called, the EIMessage parameter does not match the message data, or there is a data conversion problem.

If the ReceiveMessage method timed out or no message was available, a warning event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIService Class

Syntax:
Call *EIService*.**ReceiveMessage**(*message*, *rcvoptions*)

Call *EIService*.**ReceiveMessage**(*message*)

Call *EIService*.**ReceiveMessage**(**Nothing**, *rcvoptions*)

Call *EIService*.**ReceiveMessage**(**Nothing**)

Call *EIService*.**ReceiveMessage**()

Parameters:
message Optional. Object reference. The EIMessage to receive.

Nothing Optional. Null object reference. Used when no message is to be returned.

rcvoptions Optional. Object reference. The EIReceiveOptions to control how the message is received.

Reason codes:

EIRC_CALL_SEQUENCE_ERROR

EIRC_CONNECTION_DOWN

EIRC_DATA_CONVERSION_ERROR

EIRC_INSUFFICIENT_DATA

EIRC_INVALID_DATATYPE

EIRC_INVALID_IDENTIFIER

EIRC_INVALID_PRIORITY

EIRC_INVALID_RECEIVE_TYPE

EIRC_INVALID_WAIT_INTERVAL

EIRC_INVALID_WAIT_TYPE

EIRC_NO_MESSAGE_AVAILABLE

EIRC_NOT_CONNECTED

EIRC_NOT_SUPPORTED

EIRC_NOT_SUPPORTED

EIRC_SERVICE_STEP_UNKNOWN

EIRC_SERVICE_SYSABEND

EIRC_SERVICE_SYSERROR

EIRC_UNRECOGNIZED

EIRC_UOW_IN_DOUBT

EIRC_UOW_ROLLED_BACK

EIRC_WRONG_DATA_CONVERSION

EIRC_WRONG_IDENTIFIER

EIRC_WRONG_MESSAGE

SendMessage Method

The SendMessage method sends a message, described by the EIMessage parameter, to the enterprise service. The EISendOptions parameter controls exactly how the message is delivered. If no *sendoptions* are included in the call to this method, the default values are used.

The SendMessage method can also be used when you do not want to send any user data by using the reserved keyword **Nothing** instead of the message object reference. This is useful for sending a message that contains no user application data, such as a CICS DPL commit or backout call, or a CICS 3270 start transaction call with no start data, for example.

If there is an error sending the message, or the Connect method has not yet been called, or there is a data conversion problem, or there is a conflict of send options, an error event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIService Class

Syntax:
Call *EIService*.**SendMessage**(*message, sendoptions*)

Call *EIService*.**SendMessage**(*message*)

Call *EIService*.**SendMessage**(**Nothing**, *sendoptions*)

Call *EIService*.**SendMessage**(**Nothing***)*

Call *EIService*.**SendMessage**()

Parameters:
message Optional. Object reference. The EIMessage to send.

Nothing Optional. Null object reference. Used when no user data is to be sent.

sendoptions Optional. Object reference. The EISendOptions to control how the message is sent.

Reason codes:

EIRC_CALL_SEQUENCE_ERROR

EIRC_CONNECTION_DOWN

EIRC_DATA_CONVERSION_ERROR

EIRC_INVALID_ATTENTION_ID

EIRC_INVALID_DATATYPE

EIRC_INVALID_IDENTIFIER

EIRC_INVALID_MESSAGE_TYPE

EIRC_INVALID_SELECTED_FIELD

EIRC_INVALID_UOW

EIRC_MESSAGE_TOO_LONG

EIRC_NOT_CONNECTED

EIRC_PROTECTED_FIELD

EIRC_SERVICE_SECURITY_FAILURE

EIRC_SERVICE_SYSERROR

EIRC_WRONG_IDENTIFIER

EIRC_WRONG_MESSAGE_SENT

EIMessage Class

An EIMessage object represents the message that you send to an enterprise service, or receive from an enterprise service.

The EIMessage object allows your LotusScript program to build and interpret the message on a field by field basis using field names. An EIMessage object contains only user application data, not message headers such as those required by MQSeries bridges, and as such are enterprise service independent. This means, for example, that a given EIMessage object could represent a message that is sent to more than one enterprise service, assuming that the format of the user application data is the same.

An EIMessage might correspond to one of the following:

- MQSeries message data
- CICS DPL commarea
- CICS 3270 BMS map
- IMS transaction data

Each field in the message is represented by a property of the EIMessage object, and may be accessed or assigned as follows:

```
msg1.my_address="Home Farm"

customer_address$=msg1.my_address
```

Your LotusScript program is unaware of the offset of the field in the actual message as each field is accessed by name. When an EIMessage object is initialized, it is dynamically mapped to the corresponding message format. When your program references a field by name for the first time, the EIMessage object registers the field name as a property of the object. The property can then be examined or set in the normal manner.

If you do not know the name of the field you need to access, but you do know its position in the message, you can use the GetFieldName method of this class to extract the name of the field at the specified position prior to accessing the field by name.

Properties:

- CompletionCode Property
- FieldCount property
- Format property
- Name property
- ReasonCode property

Methods:

- ClearErrorCodes method
- GetColor method
- GetDataType method
- GetFieldName method
- GetFieldValue method
- GetHighLight method
- GetIntensity method
- GetLength method
- GetProtection method
- GetSegment method
- SetFieldValue method

LotusScript Events:

- EIError
- EIWarning

Creation:

Use the CreateMessage method from the EISession class.

Property Access:
As well as the properties listed , each user-defined field in the message is a LotusScript expanded property. You access and assign values to these in the same way as normal properties.
For example, a field called Person representing a person's name:

To get: *person$ = EIMessage*.**Person** or
person$=EIMessage.**GetFieldValue**(**"Person"**)

To set: *EIMessage*.**Person** = *person$* or **Call**
EIMessage.**SetFieldValue**(**"Person"**,*person$*)

Further, attributes of the field may be accessed (but not assigned).
For example, the data type attribute of the Person field, would be accessed as follows:

datatype% = EIMessage.**GetDataType**(**"Person"**)

The supported field attributes are:

- DataType
- Length
- Color
- Intensity
- HighLight
- Protection
- Segment

When accessing or assigning values to properties, errors can occur. If appropriate event handlers are registered, an event is raised, the CompletionCode property is set to EICC_FAILED and an appropriate value is set in the ReasonCode property for the EIMessage object.

An attempt to read a field of data type EIDT_AUTHENTICATOR or EIDT_SYSTEM_AUTHENTICATOR, meaning that the field contains a password, will return the value EIAUT_HIDDEN.

An attempt to assign a value to a field of data type EIDT_SYSTEM_AUTHENTICATOR will fail with reason code EIRC_PROTECTED_FIELD and completion code of EICC_FAILED as a field of this type is automatically set from the authenticator that is extracted from the EIService object Authenticator property.

An attempt to assign a value to a field of data type EIDT_AUTHENTICATOR will succeed. You are responsible for the value put in any field of type EIDT_AUTHENTICATOR.

CompletionCode Property

Read-only. Returns the MQEI LSX completion code set by the most recent method or property access issued against the object. It is reset to EICC_OK when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EIMessage Class

Data Type:
Long

Legal Values:

- EICC_OK
- EICC_WARNING
- EICC_FAILED

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

FieldCount Property

Read-only. Returns the number of fields in the EIMessage object. For a named EIMessage this value is set from the MQEI Definition database when the EIMessage object is created.

Defined in:
EIMessage Class

Data Type:
Long

Syntax:
To get: *count& = EIMessage*.**FieldCount**

Format Property

Read-write. This is used to indicate the format of the data within the message to be sent. It is of use to enterprise services that can process messages of different formats and need to understand the format that has been sent to them.
The initial value of this property is EIFMT_NONE.

An example of a Format is an MQSeries message format, or an IMS MID or MOD name.

Defined in:
EIMessage Class

Data Type:
String of 8 characters

Legal Values:
- EIFMT_NONE meaning no format.
- EIFMT_STRING meaning the message data is entirely characters.
- User-defined, allowable characters A-Z, a-z, 0-9, _ (underscore)

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

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

Reason codes:
EIRC_INVALID_MESSAGE_FORMAT

Name Property

Read-only. Returns the name of the message on the MQEI Definition database that the object represents.

Defined in:
EIMessage Class

Data Type:
String of 16 characters

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

ReasonCode Property

Read-only. Returns the MQEI LSX reason code set by the most recent method or property access issued against the object. It is reset to EIRC_NONE when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EIMessage Class

Data Type:
Long

Legal Values:
See the list of possible reason codes listed under Reason codes in Chapter 11.

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

ClearErrorCodes Method

Resets CompletionCode property to EICC_OK and ReasonCode property to EIRC_NONE for this object.

Defined in:
EIMessage Class

Syntax:
Call *EIMessage*.**ClearErrorCodes**

GetColor Method

For an EIMessage field that has screen attributes, returns the color attribute of the specified field as a long. If the field does not have screen attributes, EICOL_NOT_APPLICABLE is returned. If the color attribute is unknown at the time of the method call, EICOL_UNKNOWN is returned.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Legal Values:

- EICOL_NOT_APPLICABLE
- EICOL_UNKNOWN
- EICOL_BLACK
- EICOL_BLUE
- EICOL_DARK_BLUE
- EICOL_GREEN
- EICOL_GRAY
- EICOL_NEUTRAL
- EICOL_ORANGE
- EICOL_PALE_GREEN
- EICOL_PALE_TURQUOISE
- EICOL_PINK
- EICOL_PURPLE
- EICOL_RED
- EICOL_TURQUOISE
- EICOL_WHITE
- EICOL_YELLOW

Syntax:
color& = EIMessage.**GetColor**(*fieldname$*)

Parameter:
fieldname$ String. The name of a field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

GetDataType Method

Returns the data type attribute of the specified EIMessage field as a long.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Legal Values:
- EIDT_AUTHENTICATOR
- EIDT_BYTE
- EIDT_LONG
- EIDT_SHORT
- EIDT_STRING
- EIDT_SYSTEM_AUTHENTICATOR
- EIDT_VAR_STRING

Syntax:
datatype& = *EIMessage*.**GetDataType**(*fieldname$*)

Parameter:
fieldname$ String. The name of a field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

GetFieldName Method

Returns the name of the EIMessage field in the position passed to the method.

If there is no field present at the index specified, an error event is raised, and the ReasonCode and CompletionCode are set.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Syntax:
fieldname$ = EIMessage.**GetFieldName**(*index&*)

Parameters:
index& Long. The index of the field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_INVALID_INDEX

GetFieldValue Method

Returns the value of the EIMessage field. The value can be of data type String, Long or Byte.

If the field is unknown, an error event is raised, and the ReasonCode and CompletionCode are set.

For more information, see "Getting the value of a field in a message" in Chapter 4.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Syntax:
value = *EIMessage*.**GetFieldValue**(*fieldname$*)

Parameters:
fieldname$ String. The name of the field in the message.

value LotusScript string, variant, integer, long. The value of the field.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

EIRC_FIELD_TRUNCATED

GetHighLight Method

For an EIMessage field that has screen attributes, returns the HighLight attribute of the specified field as a long. If the field does not have screen attributes, EIHIL_NOT_APPLICABLE is returned. If the HighLight attribute is unknown at the time of the method call, EIHIL_UNKNOWN is returned.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Legal Values:

- EIHL_NOT_APPLICABLE
- EIHL_UNKNOWN
- EIHL_OFF
- EIHL_BLINK
- EIHL_REVERSE
- EIHL_UNDERLINE

Syntax:
highlight& = EIMessage.**GetHighLight**(*fieldname$*)

Parameter:
fieldname$ String. The name of a field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

GetIntensity Method

For an EIMessage field that has screen attributes, returns the intensity attribute of the specified field as a long. If the field does not have screen attributes, EIINT_NOT_APPLICABLE is returned. If the intensity attribute is unknown at the time of the method call, EIINT_UNKNOWN is returned.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Legal Values:

- EIINT_NOT_APPLICABLE
- EIINT_UNKNOWN
- EIINT_BRIGHT
- EIINT_NORMAL
- EIINT_DARK

Syntax:
intensity& = EIMessage.**GetIntensity**(*fieldname$*)

Parameter:
fieldname$ String. The name of a field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

GetLength Method

If the data type is EIDT_VAR_STRING, returns the length of the current contents of the field in bytes as a long.
If the data type is other than EIDT_VAR_STRING, returns the length attribute of the specified EIMessage field in bytes as a long.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Syntax:
length& = EIMessage.**GetLength**(*fieldname$*)

Parameter:
fieldname$ String. The name of a field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

GetProtection Method

For an EIMessage field that has screen attributes, returns the protection attribute of the specified field as a long. If the field does not have screen attributes, EIPRO_NOT_APPLICABLE is returned. If the protection attribute is unknown at the time of the method call, EIPRO_UNKNOWN is returned.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set for this object.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Legal Values:

- EIPRO_NOT_APPLICABLE
- EIPRO_UNKNOWN
- EIPRO_PROTECTED
- EIPRO_UNPROTECTED

Syntax:
protection& = EIMessage.**GetProtection**(*fieldname$*)

Parameter:
fieldname$ String. The name of a field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

GetSegment Method

Returns the segment attribute of the specified EIMessage field as a long.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set for this object .

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Syntax:
segment& = EIMessage.**GetSegment**(*fieldname$*)

Parameter:
fieldname$ String. The name of a field in the message.

Reason codes:
EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

SetFieldValue Method

Sets the value of the EIMessage field.

If the field is unknown, an error event is raised, and ReasonCode and CompletionCode are set.

For more information, see "Setting the value of a field in a message" in Chapter 4.

Note Events are only raised if you have an event handler registered.

Defined in:
EIMessage Class

Syntax:
Call *EIMessage*.**SetFieldValue**(*fieldname$*, *value*)

Parameters:
fieldname$ String. The name of the field in the message.

value LotusScript string, variant, integer, or long. The new value of the field.

Reason codes:
EIRC_DATA_OVERFLOW

EIRC_EMPTY_MESSAGE

EIRC_FIELD_NOT_FOUND

EIRC_FIELD_TRUNCATED

EIRC_INVALID_DATATYPE

EIRC_NULL_POINTER

EIRC_PROTECTED_FIELD

EISendOptions Class

An EISendOptions object is used to control how an EIMessage object is sent to an enterprise service, via the EIService. It gives you the flexibility to specify options in your LotusScript program, such as message priority, enterprise unit of work control, and so on.

An EISendOptions object can only be created using the CreateSendOptions method of the EISession object.

Properties:

- AttentionId property
- CompletionCode property
- Identifier property
- MessageType property
- Priority property
- ReasonCode property
- SelectedField property
- UnitOfWork property

Methods:

- ClearErrorCodes method

LotusScript Events:

- EIError
- EIWarning

Creation:

Use the CreateSendOptions method from the EISession class.

Property Access:

When assigning values to properties, errors can occur. These cause an error event to be raised, the CompletionCode property to be set to EICC_FAILED and an appropriate value to be set in the ReasonCode property for the EISendOptions object.

Note Events are only raised if you have an event handler registered.

AttentionId Property

Read-write. This allows your LotusScript program to send an attention identifier with the message. It is for use by enterprise services that support transmission of attention identifiers. As such they can be used to govern the actions of the enterprise service.
Examples of attention identifiers are the ENTER key, the CLEAR key and function keys.
Its initial value is EIAI_ENTER.

An example of an AttentionId is a CICS 3270 AID.

Defined in:
EISendOptions Class

Data Type:
Long

Legal Values:
EIAI_F1 to EIAI_F24 (Function keys 1 through to 24) inclusive

EIAI_PA1, EIAI_PA2, EIAI_PA3

EIAI_ENTER

EIAI_CLEAR

Syntax:
To get: *attentionid& = EISendOptions*.**AttentionId**

To set: *EISendOptions*.**AttentionId** = *attentionid&*

Reason codes:
EIRC_INVALID_ATTENTION_ID

CompletionCode Property

Read-only. Returns the MQEI LSX completion code set by the last method or property access issued against the object. It is reset to EICC_OK when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EISendOptions Class

Data Type:
Long

Legal Values:

- EICC_OK
- EICC_WARNING
- EICC_FAILED

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

Delivery Property

Read-write. This may be used to specify a class of delivery for the message to be sent. If the enterprise service in question supports different classes of delivery, this property can be used for this purpose.

The initial value of this property is EIDEL_DEFAULT.

Defined in:
EISendOptions Class

Data Type:
Long

Legal Values:
EIDEL_EXPRESS
Delivery is optimized for speed but the message may be lost if the network fails (non-persistent).

EIDEL_ASSURED
Delivery is slower but the message will survive network failures (persistent).

EIDEL_DEFAULT
A default delivery is assigned that is enterprise service dependent.

Syntax:
To get: *delivery&* **=** *EISendOptions.***Delivery**

To set: *EISendOptions.***Delivery =** *delivery&*

Identifier Property

Read-write. This is used to give an identity to the message to be sent. If the enterprise service supports receiving messages selectively by identifier, and assigning such an identifier to any associated reply, this property can be used to provide such an identifier.
If the special value EIID_NONE is used, then a unique identifier is generated automatically when the message is sent, and returned in this property.
The initial value of this property is EIID_NONE.

The Identifier property is also used to indicate that a group of messages belong to the same unit of work. See the UnitOfWork property.

An example of an Identifier is an MQSeries message identifier or a CICS ECI message qualifier.

Defined in:
EISendOptions Class

Data Type:
String of hexadecimal digits representing ASCII characters, valid digits are 0-9 and A-F, maximum length given by the IdentifierLength property of the associated EIService object.

Syntax:
To get: *identifier$ = EISendOptions*.**Identifier**

To set: *EISendOptions*.**Identifier** = *identifier$*

Reason codes:
EIRC_INVALID_IDENTIFIER

MessageType Property

Read-write. This property indicates the type of message to be sent. This is of use to certain enterprise services that support processing several different message types.
The initial value of this property is EIMT_REQUEST.

An example of a MessageType is an MQSeries message type.

Defined in:
EISendOptions Class

Data Type:
Long

Legal Values:

- EIMT_DATAGRAM meaning no reply is required.
- EIMT_REQUEST meaning a reply is required.
- EIMT_REPLY meaning this is the reply to a request.
- User-defined in the range EIMT_MIN to EIMT_MAX (65536 to 999999999).

Syntax:
To get: *messagetype&* = *EISendOptions*.**MessageType**

To set: *EISendOptions*.**MessageType** = *messagetype&*

Reason codes:
EIRC_INVALID_MESSAGE_TYPE

Priority Property

Read-write. This is used to give a priority to the message to be sent. If the enterprise service supports ordering of messages by priority, then this property can be used for this purpose. If the special value EIPRI_DEFAULT is used, then a default priority is assigned which is enterprise service dependent. The initial value of this property is EIPRI_DEFAULT. The range for this property is zero or greater, zero being the lowest priority.

An example of a Priority is an MQSeries message priority.

Defined in:
EISendOptions Class

Data Type:
Long

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

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

Reason codes:
EIRC_INVALID_PRIORITY

ReasonCode Property

Read-only. Returns the MQEI LSX reason code set by the last method or property access issued against the object. It is reset to EIRC_NONE when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EISendOptions Class

Data Type:
Long

Legal Values:
See the list of possible reason codes listed in Chapter 11.

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

SelectedField Property

Read-write. This allows your LotusScript program to indicate to the enterprise service that a particular named field in the message has been selected. It is for use by enterprise services that support selection of fields in this manner. An example of this is selecting one of several items in a list. Its initial value is EISF_NONE indicating no field is selected.

An example of a selected field is positioning a CICS 3270 cursor.

Defined in:
EISendOptions Class

Data Type:
String of maximum 32 characters

Legal Values:
Must be the name of a field in the associated EIMessage object.

Syntax:
To get: *selectedfield$* = *EISendOptions*.**SelectedField**

To set: *EISendOptions*.**SelectedField** = *selectedfield$*

Reason codes:
EIRC_INVALID_SELECTED_FIELD

UnitOfWork Property

Read-write. This is used to indicate the unit of work processing to be performed by the enterprise service. It is of use to enterprise services where units of work can be explicitly controlled.
The initial value of this property is EIUOW_ONLY.

If a value of EIUOW_FIRST is specified, the message identifier returned in the Identifier property by the SendMessage method must be used on all subsequent SendMessage calls for the same unit of work, that is, where one of EIUOW_MIDDLE, EIUOW_LAST, EIUOW_COMMIT or EIUOW_BACKOUT is specified.

An example of a unit of work is a series of invocations of CICS DPL programs all running under the same CICS task, as can be achieved using the CICS ECI or MQSeries CICS/ESA DPL bridge logical unit of work facilities.

> **Note** A Notes client or Domino server does not support transactional concepts, so can not be used to coordinate units of work on distributed systems. Each enterprise unit of work that your MQEI application initiates runs independently of others.
> This property always refers to a unit of work on the enterprise system. It has no effect on local commit or backout of messages, such as that provided by MQSeries. Control over local commit or backout is not supported by the MQEI LSX.

Defined in:
EISendOptions Class

Data Type:
Long

Legal Values:

- EIUOW_ONLY meaning this message is an entire UOW and a syncpoint will be taken.
- EIUOW_FIRST meaning this message is the first in a new UOW.
- EIUOW_MIDDLE meaning this message is in the middle of a UOW.
- EIUOW_LAST meaning this message is the last in a UOW and a syncpoint will be taken.
- EIUOW_COMMIT meaning a syncpoint will be taken. Any EIMessage is ignored.
- EIUOW_BACKOUT meaning the UOW is aborted and backed out. Any EIMessage is ignored.

Syntax:
To get: *unitofwork&* = *EISendOptions*.**UnitOfWork**

To set: *EISendOptions*.**UnitOfWork** = *unitofwork&*

Reason codes:
EIRC_INVALID_UNIT_OF_WORK

ClearErrorCodes Method

Resets CompletionCode property to EICC_OK and ReasonCode property to EIRC_NONE for this object.

Defined in:
EISendOptions Class

Syntax:
Call *EISendOptions*.**ClearErrorCodes**

EIReceiveOptions Class

An EIReceiveOptions object is used to control how an EIMessage is received from an enterprise service via the EIService and returned to the LotusScript program. It gives you the flexibility to specify options in your LotusScript program, such as whether to block or poll, whether to receive a new message or just return the existing message in the MQEI LSX inbound buffer, and so on.

An EIReceiveOptions object can only be created via the CreateReceiveOptions Method of the EISession object.

Properties:

- CompletionCode property
- Format property
- Identifier property
- MessageType property
- ReasonCode property
- ReceiveType property
- WaitInterval property
- WaitType property

Methods:

- ClearErrorCodes method

LotusScript Events:

- EIError
- EIWarning

Creation:

Use the CreateReceiveOptions method from the EISession class.

Property Access:

When assigning values to properties, errors can occur. These cause an error event to be raised, the CompletionCode property to be set to EICC_FAILED and an appropriate value to be set in the ReasonCode property for the EIReceiveOptions object.

Note Events are only raised if you have an event handler registered.

CompletionCode Property

Read-only. Returns the MQEI LSX completion code set by the last method or property access issued against the object. It is reset to EICC_OK when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EIReceiveOptions Class

Data Type:
Long

Legal Values:
- EICC_OK
- EICC_WARNING
- EICC_FAILED

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

Format Property

Read-only. This field is completed by the MQEI LSX after completion of a ReceiveMessage method call. It is used to indicate the format of the data within the message received. It can also be used to help identify the message just received, if the exact nature of the message is unknown.

An example of a Format is an MQSeries message format or an IMS MOD name.

Defined in:
EIReceiveOptions Class

Data Type:
String of 8 characters

Legal Values:
- EIFMT_NONE meaning no format
- EIFMT_STRING meaning the data is entirely characters
- User defined

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

Identifier Property

Read-write. This is used to specify that the receive call is to target a specific message. If the enterprise service supports receiving inbound messages selectively by identifier, this property can be used to provide such an identifier. If the special value EIID_NONE is used, then the first message that arrives is received and its identifier returned in this property.
The initial value of this property is EIID_NONE.

An example of an Identifier is an MQSeries correlation identifier.

Defined in:
EIReceiveOptions Class

Data Type:
String of hexadecimal digits representing ASCII characters, valid digits are 0-9 and A-F, maximum length given by the IdentifierLength property of the associated EIService object.

Syntax:
To get: *identifier$* = *EIReceiveOptions*.**Identifier**

To set: *EIReceiveOptions*.**Identifier** = *identifier$*

Reason codes:
EIRC_INVALID_IDENTIFIER

MessageType Property

Read-only. This field is completed by the MQEI LSX after completion of an EIService ReceiveMessage method call. It indicates the type of message just received. It can be used to help identify the message just received and govern the future action of the LotusScript program.

An example of a MessageType is an MQSeries message type.

Defined in:
EIReceiveOptions Class

Data Type:
Long

Legal Values:

- EIMT_DATAGRAM meaning no reply is required
- EIMT_REQUEST meaning a reply is required
- EIMT_REPLY meaning this is the reply to a request
- EIMT_REPORT meaning a message reporting on an expected or unexpected occurrence
- User-defined or enterprise system defined

Syntax:
To get: *messagetype& = EIReceiveOptions*.**MessageType**

ReasonCode Property

Read-only. Returns the MQEI LSX reason code set by the last method or property access issued against the object. It is reset to EIRC_NONE when a call, other than a property Get, is made successfully against the object, or when the ClearErrorCodes method is called.

Defined in:
EIReceiveOptions Class

Data Type:
Long

Legal Values:
See the list of possible reason codes listed under Reason Codes in Chapter 11.

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

ReceiveType Property

Read-write. This indicates exactly what to receive and return to the LotusScript program.
The options are:

- EIRT_RECEIVE. Discard the last message received, and receive a new message into the MQEI LSX inbound buffer. Return this new message to the LotusScript program as an EIMessage object.
- EIRT_RETURN. Return the message currently in the MQEI LSX inbound buffer to the LotusScript program as an EIMessage object. This is useful when processing variant messages.

The initial value of this property is EIRT_RECEIVE.

Defined in:
EIReceiveOptions Class

Data Type:
Long

Legal Values:
- EIRT_RECEIVE
- EIRT_RETURN

Syntax:
To get: *receivetype&* = *EIReceiveOptions*.**ReceiveType**

To set: *EIReceiveOptions*.**ReceiveType** = *receivetype&*

Reason codes:
EIRC_INVALID_RECEIVE_TYPE

WaitInterval Property

Read-write. The maximum time in milliseconds that the receive call waits for a suitable message to arrive. If no suitable message has arrived after this time has elapsed, a warning with reason code EIRC_NO_MESSAGE_AVAILABLE is raised. It is of use only when the communication mechanism supports timeout, and the WaitType property has the value EIWT_WAIT.
A special value EIWI_UNLIMITED may be specified which means an unlimited wait is required.
The initial value of this property is 0 (EIWT_NO_WAIT).

An example of a WaitInterval is an MQSeries get message options wait interval or a CICS ECI get reply call timeout.

Defined in:
EIReceiveOptions Class

Data Type:
Long

Syntax:
To get: *waitinterval&* = *EIReceiveOptions*.**WaitInterval**

To set: *EIReceiveOptions*.**WaitInterval** = *waitinterval&*

Reason codes:
EIRC_INVALID_WAIT_INTERVAL

WaitType Property

Read-write. This indicates whether to wait for the time specified by the WaitInterval property, or to return immediately without waiting, when receiving a message. If no message is returned, either because no wait was specified or the wait interval expired, an error is raised with reason code EIRC_NO_MESSAGE_AVAILABLE.
The initial value of this property is EIWT_NO_WAIT.

An example of a WaitType is an MQSeries get message options wait type, or the type of an CICS ECI get reply call, or the CICS EPI get event call wait type.

Defined in:
EIReceiveOptions Class

Data Type:
Long

Legal Values:

- EIWT_WAIT meaning return after the specified wait interval with or without the message.
- EIWT_NO_WAIT meaning return immediately with or without the message.

Syntax:
To get: *waittype&* = *EIReceiveOptions*.**WaitType**

To set: *EIReceiveOptions*.**WaitType** = *waittype&*

Reason codes:
EIRC_INVALID_WAIT_TYPE

ClearErrorCodes Method

Resets CompletionCode property to EICC_OK and ReasonCode property to EIRC_NONE for this object.

Defined in:
EIReceiveOptions Class

Syntax:
Call *EIReceiveOptions*.**ClearErrorCodes**

Appendix A Sample using a Native MQSeries service

This sample demonstrates how you can use the MQEI LSX in conjunction with a Notes database to interact with MQSeries.

For general information relating to all the MQEI samples, see "MQEI Samples" in Chapter 4.

This appendix describes the sample that uses the Native MQSeries service:

- Its design
- Preparation required
- How to run
- How it works

For more information on programming for a Native MQSeries service, see Chapter 6.

Design of the Native MQSeries sample

This sample allows you to send data from a Notes application and get a reply (containing the same data) from an MQSeries environment.

In summary:

1. The Notes application creates a message by concatenating the text you input with some system defined fields.
2. By clicking the OK button (Send Message), this message is passed to the MQSeries environment.
3. By clicking the OK button again (Receive Message), the sample can receive the message from the MQSeries environment.
4. A binary check is carried out to ensure that the message sent is the same as the one received. The result is displayed in a Notes document.

To keep the sample as simple as possible, there is no enterprise program processing the message. It is first put onto an MQSeries queue and then retrieved from it. This demonstrates the use of MQEI with MQSeries without needing to run another application to process the message.

Notes MQEI Samples database (mqeisamp.nsf)

This sample is implemented by the "MQ sample" form.

Notes MQEI Definition database (mqeidata.nsf)

The definitions used by this sample are:

- **MQServ (MQEI Service definition)**
  This is a Native MQSeries service.
  - Connection Manager - Not specified, assuming connection is to the default MQSeries queue manager
  - Outbound Connection - An MQSeries queue named MQEI.MQ.INOUTQ
  - Inbound Connection - An MQSeries queue named MQEI.MQ.INOUTQ
  - The defaults are used for all the other fields of this service.
- **MQMsg (MQEI Message definition)**
  This message defines the structure of the MQSeries message sent from and received by the sample.
  It contains three STRING fields:
  - UserName (max 30 characters)
  - Date (max 8 characters)

- Message (max 30 characters)

MQEI MQSC command file (mqeisamp.tst)

The Native MQSeries section of this MQSC command file sets up the queue that you need to run this sample.

Before you run the Native MQSeries sample

This sample will not run successfully until you have completed the following:

Domino Server system

Lotus Notes

- Add the MQEI Samples database (mqeisamp.nsf) to your workspace.
- Ensure that the MQEI LSX is installed correctly.

  Note The MQEI Security database is not used in this sample.

For more information on Getting Started, see Chapter 2.

MQSeries

- Ensure a local default MQSeries queue manager is running.
- Create the necessary MQSeries queue (MQEI.MQ.INOUTQ). You can do this by running the MQEI MQSC command file (mqeisamp.tst) that is supplied in the MQEI package.

  Tip The MQEI MQSC command file (mqeisamp.tst) provided with the MQEI package creates the channels and queues needed by all the samples. Consider making a copy of this file and removing (or commenting out) anything not needed by this sample. There are instructions within the MQSC command file (mqeisamp.tst).

  Note If you want to use a different MQSeries queue manager, add the MQEI Definition database to your workspace and change the Connection Manager in the MQServ MQEI Service definition to your queue manager name.

For more information, see "MQEI Service definition" in Chapter 3.

Running the Native MQSeries sample

1. Check that all the appropriate setup work has been completed. See "Before you run the Native MQSeries sample" for details.
2. From your Notes workspace, open the MQEI Samples database.
3. From the navigator, select the "Native MQSeries" icon. An "MQSeries Enterprise Integrator for Native MQSeries" sample document is displayed on the screen.
   The form has the following fields and button:
   - Name (field)
   - Date (field)
   - Message Data (field)
   - Message To Be Sent
   - Message Received
   - Message Comparison Result (field)
   - OK (button)

To send a message:

1. In the field "Message Data", enter a string of data that you want the message to contain.
2. Tab to the next field.
   As soon as you move the cursor from the text input field, "Message Data" the message is created.
3. Click OK (Send Message) to send the message.

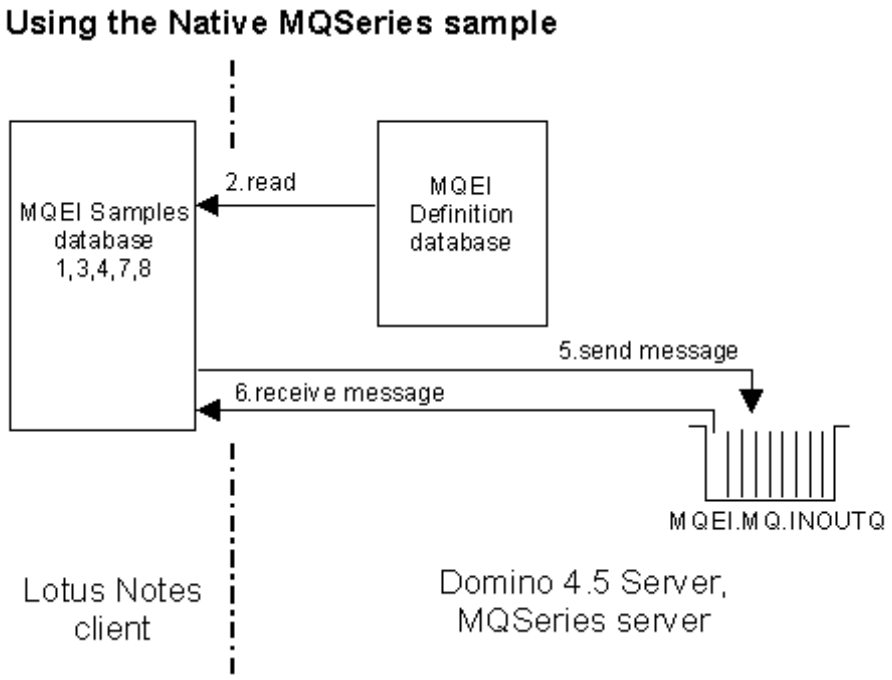
   Note The first time you send a message there may be a delay while the MQEI LSX connects to the queue manager.

To receive a message:

1. Click OK (Receive Message) to receive the message.
2. The message is retrieved from the queue and the contents are displayed in the Message Received field.
   If there are no messages on the queue, you get the message, "Your LotusScript program has tried to get an EIMessage, but no message was received. The target enterprise system may not have sent the message yet, or the target system may have encountered an error."
   To get the message back, you must keep the Native MQSeries sample form open otherwise the sample does not allow you to retrieve the sent message from the queue.
3. The message received is compared with the one sent and the result of the binary comparison is displayed in the Message Comparison Result field.

How the Native MQSeries sample works

The sample expects you to send a message before you can receive it.



Steps:

1. When you click on the "Native MQSeries" icon, Notes creates a document using the "MQ Sample" form.
2. Before the document is opened, the Notes application creates the MQEI objects (EISession, EIMessage and EIService). It does this by invoking the appropriate create method which causes the MQEI LSX to read the MQEI Definition database for the details of the MQEI Service and MQEI Message definitions used by this sample.
3. If the MQEI object creation is successful, the document is displayed.
4. You enter a string of data up to 30 characters.
   When the cursor leaves the field on the Notes document, the Notes application builds the message in the correct EIMessage format.
5. When you click the "Send Message" button, the Notes application issues an MQEISendMessage that puts a message on the MQSeries queue (the name of which is specified in the outbound connection name of the MQEI Service definition).

6. When you click the "Receive Message" button, the MQEI gets the message from the MQSeries queue (the name of which is specified in the inbound connection name of the MQEI Service definition).
7. The Notes application compares the sent message with the received message to see that any data conversion was successful.
8. The Notes application displays the received message and the results in the document.

Note You should ensure that you receive all messages before closing the document.
This is to prevent the existence of obsolete messages on the queue.

Error handling in the Native MQSeries sample

Error checking takes place throughout this sample, using both ON ERROR routines and EVENT handlers.

For more information, see "Error handling" in Chapter 4.

Appendix B Sample using an IMS via MQSeries service

This sample demonstrates how you can use the MQEI LSX in conjunction with a Notes database to interact with IMS via the MQSeries IMS bridge, and more specifically, how to issue an IMS command and handle its multi-segment reply.

For general information relating to all the MQEI samples, see "MQEI Samples" in Chapter 4.

This appendix describes the sample that uses the IMS via MQSeries service:

- Its design
- Preparation required
- How to run
- How it works

For more information on programming for an IMS via MQSeries service, see Chapter 7.

Design of the IMS via MQSeries sample

This sample allows you to display information about an IMS transaction by sending a message containing the /DISPLAY TRANSACTION command to the IMS system.

In summary:

1. The Notes application creates a message containing an IMS /DISPLAY TRANSACTION command by concatenating the text you input, with some system defined fields.
2. By clicking the "Submit Command" action button, this message is passed to the IMS environment via the MQSeries IMS bridge.
3. IMS processes the command.
4. IMS sends a reply back to the Notes application via MQSeries. The sample application then receives the message and displays the results in a document.

Notes MQEI Samples database (mqeisamp.nsf)

This sample is implemented by the "IMS Sample" form.

Notes MQEI Definition database (mqeidata.nsf)

The definitions used by this sample are:

- **IMSMQServ (MQEI Service definition)**
  This is an IMS via MQSeries service:
  - Connection Manager - Not specified assuming connection is to the default MQSeries queue manager.
  - Outbound Connection - An MQSeries remote queue definition named MQEI.IMS.OUTPUTQ for the IMS bridge queue
  - Inbound Connection - An MQSeries queue named MQEI.IMS.INPUTQ
- **IMSMQMsg (MQEI Message definition)**
  This message defines the structure of the outbound message being sent from the sample. It contains three STRING fields.
- **IMSMQRCDMsg1 (MQEI Message definition)**
  This message defines the structure of a valid return message from the IMS sample. This message has 5 segments.
- **IMSMQRcdMsg2 (MQEI Message definition)**
  This message defines the structure of an invalid return message from the IMS sample. This message has 3 segments.

MQEI MQSC command file (mqeisamp.tst)
The IMS via MQSeries section of this file defines the queues you need to run this sample.

MQEI MVS/ESA command file (mqeisamp.jcs)
This file contains the MQSeries for MVS/ESA resource definitions required for this sample.

Before you run the IMS via MQSeries sample

This sample will not run successfully until you have completed the following:

Domino Server system

Lotus Notes

- Add the MQEI Samples database (mqeisamp.nsf) to your workspace.
- Add the MQEI Security database (mqeisecu.nsf) to your workspace and create an MQEI Security definition for the IMS system you want to use.
- Add the MQEI Definition database (mqeidata.nsf) to your workspace and modify the IMSMQServ MQEI Service definition and set the System Name field to match the name of the MQEI Security definition for your IMS system.
- Ensure that the MQEI LSX is installed correctly.

For more information on Getting Started, see Chapter 2.

MQSeries

- Ensure a local default MQSeries queue manager is running.
- Create the necessary MQSeries queues (MQEI.IMS.INPUTQ (local) and MQEI.IMS.OUTPUTQ (remote)). You can do this by running the MQEI MQSC command file (mqeisamp.tst) that is supplied in the MQEI package.

  Tip The MQEI MQSC command file (mqeisamp.tst) provided with the MQEI package creates the channels and queues needed by all the samples. Consider making a copy of this file and removing (or commenting out) anything not needed by this sample. There are instructions within the MQSC command file (mqeisamp.tst).

  Note If you want to use a different MQSeries queue manager, add the MQEI Definition database to your workspace and change the Connection Manager in the IMSMQServ MQEI Service definition to your queue manager name.

For more information, see "MQEI Service definition" in Chapter 3.

MVS system

MQSeries for MVS/ESA

- Create the necessary MQSeries queue (MQEI.IMS.BRIDGE.QUEUE). You can do this by running the MQEI MQSC command file (mqeisamp.jcs) that is supplied in the MQEI package.

  Tip The MQEI MQSC command file (mqeisamp.jcs) provided with the MQEI package creates the channels and queues needed by all the samples. Consider making a copy of this file and removing (or commenting out) anything not needed by this sample. There are instructions within the MQSC command file (mqeisamp.jcs).

For more information, see "MQEI Service definition" in Chapter 3.

IMS/ESA

- Ensure that OTMA is started and that the IMS bridge is operational and configured correctly.
- Ensure you have a valid User ID and Authenticator (password) for IMS.

If you are directly connected to MQSeries for MVS/ESA

It is assumed that you are running the sample where your MQSeries client is **not** connected directly to MQSeries for MVS/ESA. For this reason the command files supplied include definitions for remote queues and channels.

If you are running with your Notes client or Domino Server with an MQSeries client directly connected to an MQSeries for MVS/ESA server, you only need to define an input and output queue to MQSeries:

- Change the Outbound Connection of the IMSMQServ MQEI Service definition to MQEI.IMS.BRIDGE.QUEUE
- Delete the definition of the transmission queue and the channels from the mqeisamp.jcs file.
- Use the mqeisamp.jcs file only.

For information on system configuration, see "Possible system configurations" in Chapter 2.

Running the IMS via MQSeries sample

Sending a message

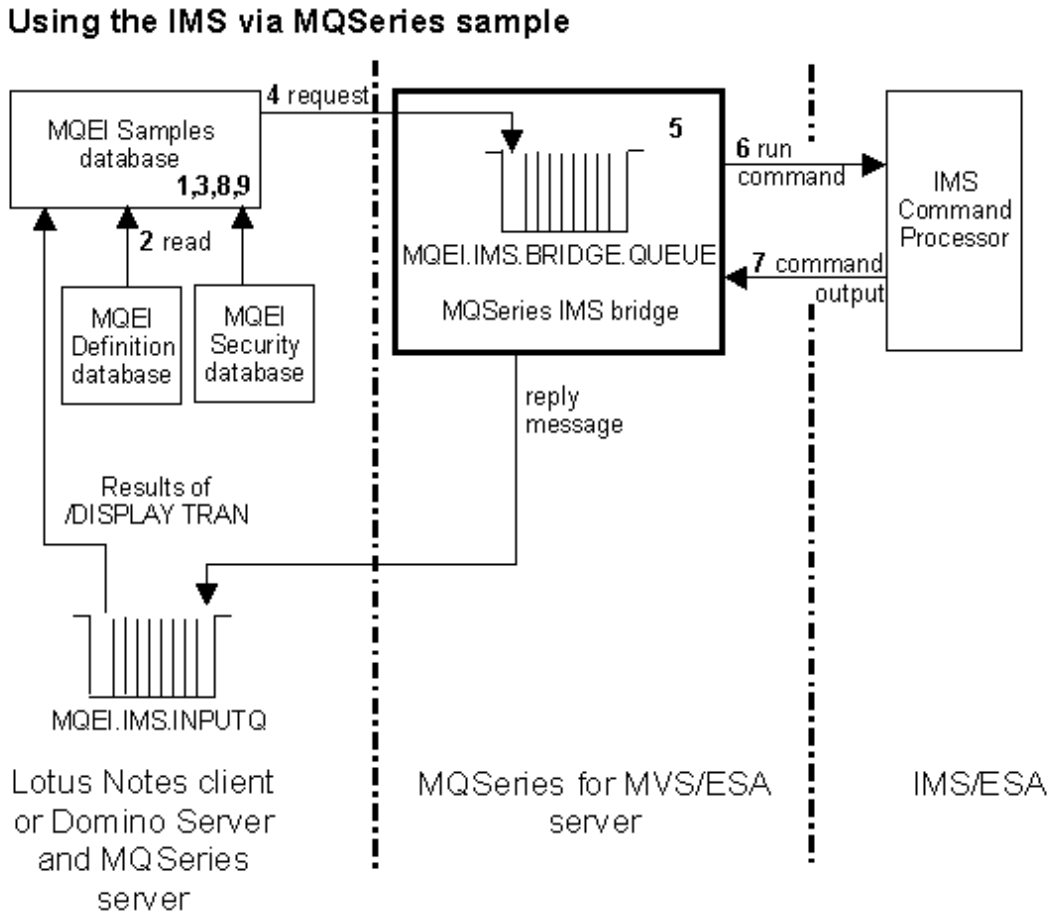
1. Check that all the appropriate setup work has been completed. See "Before you run the IMS via MQSeries sample" for details.
2. From your Notes workspace, open the MQEI Samples database.
3. Select the "IMS via MQSeries" button from the Notes Navigator. If you have set everything up correctly, an IMS sample document should appear.
4. Enter a valid transaction ID in the field labeled, "Please enter the transaction to be displayed."
5. Press the "Submit Command" button.
6. The status bar gives you information about what is happening when you run the sample.

Receiving a message

You don't have to do anything to receive a message back from IMS. This is done automatically.

1. If everything was successful, IMS returns a message back to the Notes application.
2. The Notes application displays the results of the IMS /DISPLAY TRANSACTION command.

How the IMS via MQSeries sample works



1. When you click on the "IMS via MQSeries" icon, Notes creates a document using the "IMS Sample" form.
2. Before the document is opened, the Notes application creates the MQEI objects (EISession, EIMessage and EIService). It does this by invoking the appropriate create method which causes the MQEI LSX to read the MQEI Definition database for the details of the MQEI Service and MQEI Message definitions used by this sample.
3. If the MQEI object creation is successful, the document is displayed.

4. When you click on the button labeled, "Submit Command", the LotusScript code behind the button generates an EIMessage containing the /DISPLAY TRANSACTION command and issues an MQEI SendMessage to put a request on the IMS bridge input queue.

   Note In IMS, a command is always preceded by a forward slash (/).

5. The message, that contains the /DISPLAY TRANSACTION command, is stored on the IMS bridge input queue until the IMS bridge (which continually monitors all bridge queues) takes the message off the bridge queue.
6. The IMS bridge receives the request message and passes it to the IMS command processor which issues the command to IMS.
7. The IMS command processor generates a reply which is converted to a message on the MQSeries reply-to queue by the MQ-IMS bridge.
8. The Notes application receives the reply message in an EIMessage by issuing an MQEI Receive Message.
9. The Notes application displays information about the transaction in a document.

Error handling in the IMS via MQSeries sample

The IMS via MQSeries sample application uses variant message handling. If the expected (valid) message is not received (EIRC_WRONG_MESSAGE), then the invalid reply message is checked. Only if this is not accepted does the sample flag an error.

For more information, see "Variant Messages" in Chapter 4.

Error checking takes place throughout this sample, using both ON ERROR routines and EVENT handlers. For more information, see "Error handling" in Chapter 4.

Appendix C Sample using a CICS DPL via MQSeries service

This sample demonstrates how you can use the MQEI LSX in conjunction with a Notes database to interact with CICS via the MQSeries-CICS/ESA bridge.

For general information relating to all the MQEI samples, see "MQEI Samples" in Chapter 4.

This appendix describes the sample that uses the CICS DPL via MQSeries service:

- Its design
- Preparation required
- How to run
- How it works

For more information on programming for a CICS DPL via MQSeries service, see Chapter 8.

Design of the CICS DPL via MQSeries sample

This sample is a Notes application that browses file FILEA by running the CICS sample program DFH$AXCS.

In summary:

1. The Notes application creates a message containing the name of the CICS program to be run, DFH$AXCS.
2. By clicking the GO action button, this message is passed to CICS via the MQSeries CICS/ESA bridge.
3. DFH$AXCS runs (reads FILEA), passes the data back to the MQSeries CICS/ESA bridge, where a message is created and passed to the Notes application.
4. The information in the message is displayed in a document.
5. This process repeats until every record in FILEA has been read and displayed.

Note This sample only runs on CICS/ESA Version 4.1 and above (the DFH$AXCS program is not available on CICS/ESA Version 3.3).

Notes MQEI Samples database (mqeisamp.nsf)

This sample is implemented by the "CICSDPLMQ Sample" form.

Notes MQEI Definition database (mqeidata.nsf)

The definitions used by this sample are:

- **CICSDPLMQServ (MQEI Service definition)**
  This is a CICS DPL via MQSeries service
  - Service Step - DFH$AXCS (this is the CICS program that the sample requests to be run)
  - Connection Manager - Not specified, assuming connection is to the default MQSeries queue manager
  - Outbound Connection - An MQSeries remote queue definition named MQEI.CICS.DPL.OUTPUTQ for the MQSeries - CICS/ESA bridge queue.
  - Inbound Connection - An MQSeries queue named MQEI.CICS.DPL.INPUTQ
- **CICSDPLMsg (MQEI Message definition)**
  This defines a message that describes the commarea layout as expected by the CICS program DFH$AXCS.
  It contains four fields:
  - CallTypeCode

- TargetFileName
- RidFieldID
- RecordArea

MQEI MQSC command file (mqeisamp.tst)

The CICS DPL via MQSeries section of this file sets up the queues and channels that you need to run this sample.

MQEI MVS/ESA command file (mqeisamp.jcs)

This file contains the MQSeries for MVS/ESA resource definitions required for this sample.

Before you run the CICS DPL via MQSeries sample

This sample will not run successfully until you have completed the following:

Domino Server system

Lotus Notes

- Add the MQEI Samples database (mqeisamp.nsf) to your workspace.
- Ensure the MQEI LSX is installed correctly.

  Note The security database is not used as it is assumed there is no security associated with the DFH$AXCS program.
  If there is:
  - Add the MQEI Security database to your workspace and create an MQEI Security definition for the CICS system you want to use.
  - Add the MQEI Definition database (mqeidata.nsf) to your workspace and modify the CICSDPLMQServ MQEI Service definition and set the System Name field to match the name of the MQEI Security definition for your CICS system.

MQSeries

- Ensure a local default MQSeries queue manager is running.
- Create the necessary MQSeries queues (MQEI.CICS.DPL.INPUTQ (local) and MQEI.CICS.DPL.OUTPUTQ (remote definition)). You can do this by running the MQEI MQSC command file (mqeisamp.tst) that is supplied in the MQEI package.

  Tip The MQEI MQSC command file (mqeisamp.tst) provided with the MQEI package creates the channels and queues needed by all the samples. Consider making a copy of this file and removing (or commenting out) anything not needed by this sample. There are instructions within the MQSC command file.

  Note If you want to use a different MQSeries queue manager, add the MQEI Definition database to your workspace and change the Connection Manager in the CICSDPLMQServ MQEI Service definition to your queue manager name.

MVS system

MQSeries for MVS/ESA

- Create the necessary MQSeries queue (MQEI.CICS.BRIDGE.QUEUE). You can do this by running the MVS/ESA command file (mqeisamp.jcs) that is supplied in the MQEI package.

  Tip The MVS/ESA command file (mqeisamp.jcs) provided with the MQEI package creates the channels and queues needed by all the samples. Consider making a copy of this file and removing (or commenting out) anything not needed by this sample. There are instructions within the command file.

For more information, see "MQEI Service Definition" in Chapter 3.

If you are directly connected to MQSeries for MVS/ESA

It is assumed that you are running the sample where your MQSeries client is **not** connected directly to MQSeries for MVS/ESA. For this reason the command files supplied include definitions for remote queues and channels.

If you are running with your Notes client or Domino Server with an MQSeries client directly connected to an MQSeries for MVS/ESA server, you only need to define an input and output queue to MQSeries:

- Change the Outbound Connection of the CICSDPLMQServ MQEI Service definition to MQEI.CICS.BRIDGE.INPUTQ
- Delete the definition of the transmission queue and the channels from the mqeisamp.jcs file.
- Use the mqeisamp.jcs file only.

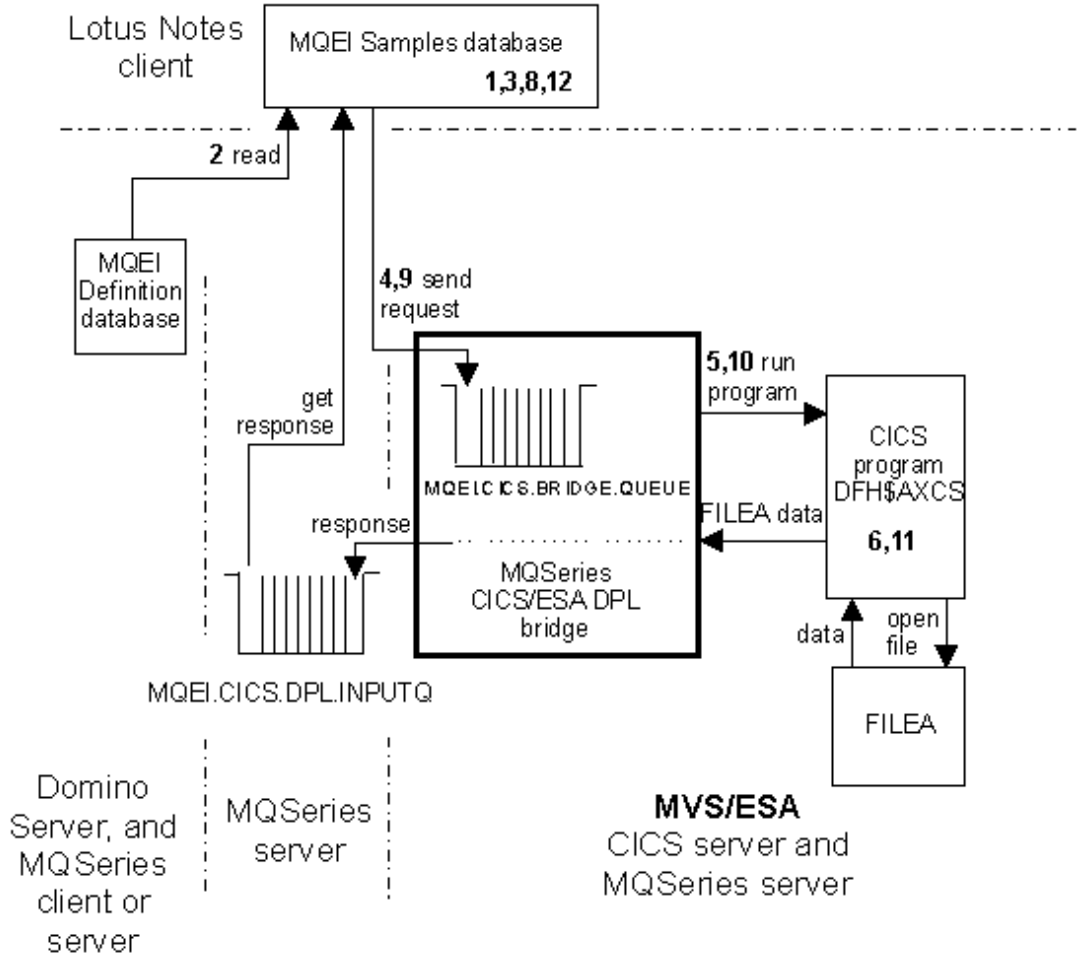
For information on system configuration, see "Possible system configurations" in Chapter 2.

Running the CICS DPL via MQSeries sample

1. Check all the appropriate setup work has been completed. See "Before you run the CICS DPL via MQSeries sample" for details.
2. From your Notes workspace, open the MQEI Samples database.
3. From the navigator, select the "CICS DPL via MQSeries" icon. If you have everything set up correctly, an "MQSeries Enterprise Integrator for CICS DPL via MQSeries" sample document is displayed on the screen.
4. Click GO.
5. The status bar gives you information about what is happening when you run the sample.
6. The Notes application displays the data returned by the DFH$AXCS program.

How the CICS DPL via MQSeries sample works

Using the CICS DPL via MQSeries sample



Steps:

1. When you click the "CICS DPL via MQSeries" icon, Notes creates a document using the "CICSDPLMQ Sample" form.
2. Before the document is opened, the Notes application creates the MQEI objects (EISession, EIMessage and EIService). It does this by invoking the appropriate create method which causes the MQEI LSX to read the

MQEI Definition database for details of the MQEI Service and MQEI Message definitions used by this sample.

3. If the MQEI object creation is successful, the document is displayed.
4. The Notes application (via the MQEI LSX) sends a request message, to run the CICS program DFH$AXCS, to the MQSeries queue MQEI.CICS.DPL.OUTPUTQ (specified in the outbound connection name of the service definition) and waits for a reply.

   Note MQEI.CICS.DPL.OUTPUTQ is a remote queue definition for the MQSeries - CICS/ESA bridge queue, MQEI.CICS.BRIDGE.QUEUE
5. The MQSeries - CICS/ESA bridge issues an EXEC CICS link call to run the DFH$AXCS program to open the file called FILEA (named in the MQSeries message).
6. The DFH$AXCS program attempts to open the FILEA file if it isn't already open and returns the results to the MQSeries queue MQEI.CICS.DPL.INPUTQ via the MQSeries CICS/ESA bridge.
7. The Notes application (via the MQEI LSX) receives the message and displays an error message if the FILEA file is not open.

If the file is open:

8. The Notes application (via the MQEI LSX) sends a request message to run DFH$AXCS to get the data from the next record in the file.
9. The bridge issues an EXEC CICS link call to run the DFH$AXCS program.
10. The DFH$AXCS program runs, reads the next record in FILEA and returns the data to the MQSeries - CICS/ESA bridge which puts a message on the MQSeries queue MQEI.DPL.INPUTQ (specified in the InboundConnection property).
11. The Notes application (via the MQEI LSX) gets the message from the queue MQEI.CICS.DPL.INPUTQ. If the read was successful, the Notes application displays the text read on the form. If the read was not successful, the Notes application displays an error message.
12. The Notes application (via the MQEI LSX) continues sending further request messages to run DFH$AXCS to get the data from the records in the file, until the end-of-file or an error condition occurs.

Error handling in the CICS DPL via MQSeries sample

Error checking takes place throughout this sample, using both ON ERROR routines and EVENT handlers.

For more information, see "Error handling" in Chapter 4.

Appendix D Sample using a CICS DPL direct service

This sample demonstrates how you can use the MQEI LSX in conjunction with a Notes database to interact with CICS via the CICS ECI interface.

For general information relating to all the MQEI samples, see "MQEI Samples" in Chapter 4.

This appendix describes the sample that uses the CICS DPL direct service:

- Its design
- Preparation required
- How to run
- How it works

For more information on programming for a CICS DPL direct service, see Chapter 9.

Restrictions

The following restrictions apply to this sample:

- CICS DPL direct services are not supported when running the MQEI LSX under AIX, HP-UX or Sun Solaris. The CICS DPL direct sample must be run from OS/2, Windows 3.1, Windows 95 or Windows NT.
- CICS 3270 direct services and CICS DPL direct services may not be used from the same Notes client or Domino Server agent manager at the same time. If you wish to run both CICS 3270 direct and CICS DPL direct samples, you must restart your Notes client.

Design of the CICS DPL direct sample

This sample is a Notes application that browses file FILEA by running the CICS sample program DFH$AXCS.

In summary:

1. The Notes application creates a message containing the name of the CICS program to be run, DFH$AXCS.
2. By clicking the GO action button, this message is passed to CICS via the CICS ECI interface.
3. DFH$AXCS runs (reads FILEA), passes the data back to the CICS ECI interface, where a message is created and passed to the Notes application.
4. The information in the message is displayed in a document.
5. This process repeats until every record in FILEA has been read and displayed.

Note This sample only runs on CICS/ESA Version 4.1 and above (the DFH$AXCS program is not available on CICS/ESA Version 3.3).

Notes MQEI Samples database (mqeisamp.nsf)

This sample is implemented by the "CICSDPLD Sample" form.

Notes MQEI Definition database (mqeidata.nsf)

The definitions used by this sample are:

- **CICSDPLDServ (MQEI Service definition)**
  This is a CICS DPL direct service:
  - Service Step - DFH$AXCS (this is the CICS program that the sample requests to be run)
  - Connection Manager - CICSSNA, the default name for a SNA connected CICS server
- **CICSDPLMsg (MQEI Message definition)**
  This defines a message that describes the commarea layout as expected by the CICS program DFH$AXCS.
  It contains four fields:
  - CallTypeCode
  - TargetFileName
  - RidFieldID
  - RecordArea

Before you run the CICS DPL direct sample

This sample will not run successfully until you have completed the following:

Domino Server system

Lotus Notes

- Add the MQEI Samples database (mqeisamp.nsf) to your workspace.
- Ensure the MQEI LSX is installed correctly.

  Note The security database is not used as it is assumed there is no security associated with the DFH$AXCS program.
  If there is:
  - Add the MQEI Security database to your workspace and create an MQEI Security definition for the CICS system you want to use.
  - Add the MQEI Definition database (mqeidata.nsf) to your workspace and modify the CICSDPLDServ MQEI Service definition and set the System Name field to match the name of the MQEI Security definition for your CICS system.
- If the name of your CICS server defined in your CICS Client initialization file is not CICSSNA:
  - Add the MQEI Definition database (mqeidata.nsf) to your workspace and change the Connection Manager in the CICSDPLDServ MQEI Service definition to your server name.

CICS Client

- A CICS Client must be available on your Domino Server system.

MVS system

CICS/ESA server

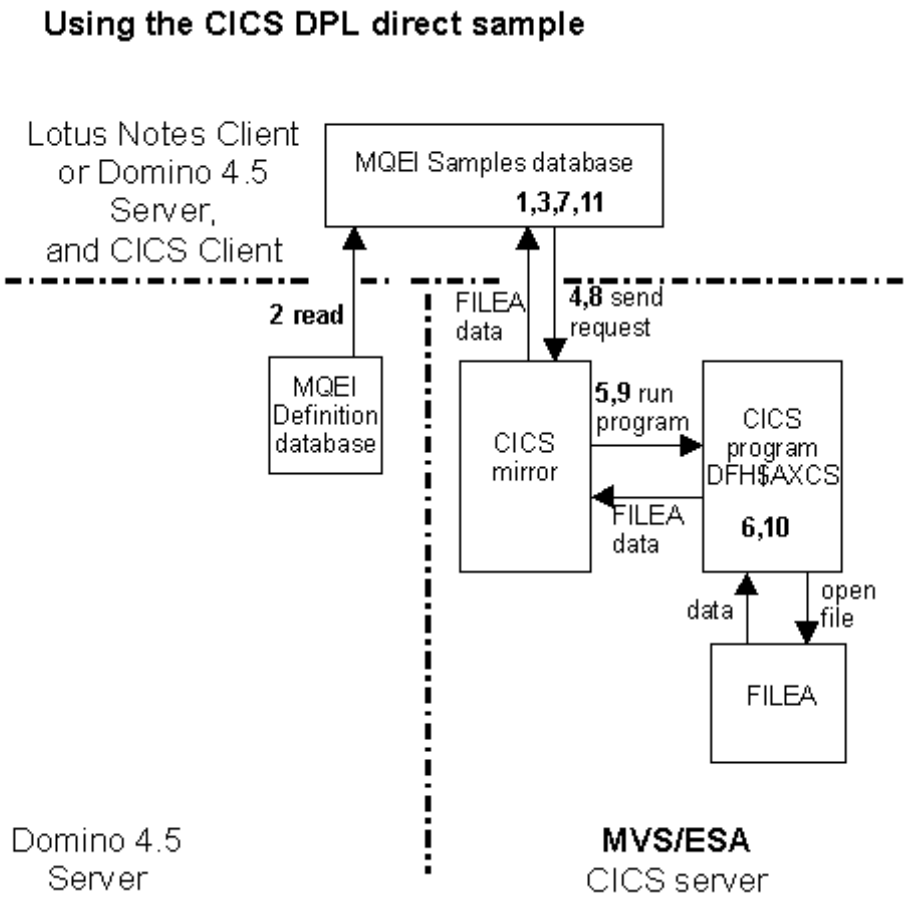
- Make sure that the CICS/ESA FILEA sample is installed and the server is up and running.
- Turn off data conversion on CICS/ESA.

For information on system configuration, see "Possible system configurations" in Chapter 2.

Running the CICS DPL direct sample

1. Check all the appropriate setup work has been completed. See "Before you run the CICS DPL direct sample " for details.
2. From your Notes workspace, select the MQEI Samples database icon.
3. From the navigator, select the "CICS DPL direct" icon. If you have everything set up correctly, an "MQSeries Enterprise Integrator for CICS DPL direct" sample document is displayed on the screen.
4. Click GO.
5. The status bar gives you information about what is happening when you run the sample.
6. The Notes application displays the data returned by the DFH$AXCS program.

How the CICS DPL direct sample works



Steps.

1. When you click the "CICS DPL direct" icon, Notes creates a document using the "CICSDPLD Sample" form.
2. Before the document is opened, the Notes application creates the MQEI objects (EISession, EIMessage and EIService). It does this by invoking the appropriate create method which causes the MQEI LSX to read the MQEI Definition database for details of the MQEI Service and MQEI Message definitions used by this sample.
3. If the MQEI object creation is successful, the document is displayed.
4. The Notes application (via the MQEI LSX) sends a request message, to run the CICS program DFH$AXCS.

5. A CICS mirror task issues an EXEC CICS link call to run the DFH$AXCS program to open the file called FILEA.
6. The DFH$AXCS program attempts to open the FILEA file if it isn't already open and returns the results to the Notes application.
7. The Notes application (via the MQEI LSX) receives the message and displays an error message if the FILEA file is not open.

If the file is open:

8. The Notes application (via the MQEI LSX) sends a request message to run DFH$AXCS to get the data from the next record in the file.
9. A mirror task issues an EXEC CICS link call to run the DFH$AXCS program.
10. The DFH$AXCS program runs, reads the next record in FILEA and returns the data in the record to the MQEI.
11. The Notes application displays the data on the document. If the read was not successful, the Notes application displays an error message.
12. The Notes application (via the MQEI LSX) continues sending further messages to run DFH$AXCS to get the data from the records in the file until an end-of-file or error condition occurs.

Error handling in the CICS DPL direct sample

Error checking takes place throughout this sample, using both ON ERROR routines and EVENT handlers.

For more information, see "Error handling" in Chapter 4.

Appendix E Sample using a CICS 3270 direct service (signon)

This sample demonstrates how you can use the MQEI LSX in conjunction with a Notes database to interact with a CICS 3270 application via the CICS EPI interface.

For general information relating to all the MQEI samples, see "MQEI Samples" in Chapter 4.

This appendix describes the sample that uses the CICS 3270 direct service:

- Its design
- Preparation required
- How to run
- How it works

For more information on programming for the CICS 3270 direct service, see Chapter 10.

Restrictions

The following restrictions apply to this sample:

- CICS 3270 direct services are not supported when running the MQEI LSX under AIX, HP-UX, Sun Solaris, Windows 3.1, or WIN-OS/2. The CICS 3270 samples must be run from OS/2, Windows 95 or Windows NT.
- CICS 3270 direct services and CICS DPL direct services may not be used from the same Notes client or Domino Server agent manager at the same time. If you wish to run both CICS 3270 direct and CICS DPL direct samples, you must restart your Notes client.

Design of the CICS 3270 signon sample

This sample allows you to signon to CICS or change your password, from a Notes document.

In summary:

This sample application uses the CICS CESN transaction.

1. The Notes application uses the appropriate MQEI Message and Service definitions depending upon the CICS system you select.
2. The Notes application creates a message from the text you input, and some system defined fields.
3. By clicking an action button, data from this message is sent to CICS to initiate the CESN transaction, either for "Signon" or to "Change Password".
4. CICS sends a reply back to the Lotus Notes sample application which displays the results in the document.

Notes MQEI Samples database (mqeisamp.nsf)

This sample is implemented by the "CICS 3270 Direct Sample: signon" form.

Notes MQEI Definition database (mqeidata.nsf)

The MQEI Service definitions used by this sample are:

- **CICS3270DServOS2 (MQEI Service definition)**
  This is a CICS 3270 direct service that is used to run CESN on CICS for OS/2.
  - Service Step - CESN, the initial transaction to be run.
  - Connection Manager - CICSMRO, the name required to use a CICS server via a built-in Client.
- **CICS3270DServOPN (MQEI Service definition)**
  This is a CICS 3270 direct service that will be used to run CESN on CICS for Open Systems.
  - Service Step - CESN, the initial transaction to be run
  - Connection Manager - CICSTCP, the default name for a TCP/IP connected CICS server using a separate CICS Client.

The MQEI Message definitions used by this sample are:

- **CICS3270DMsgOS2 -** CESN map on CICS for OS/2
- **CICS3270DMsgOPN -** CESN map on CICS for Open Systems
- **CICS3270DMsgCLR -** Single byte for screen clear
- **CICS3270DMsgL01 -** General line 1 response message
- **CICS3270DMsgL23 -** General line 23 response message
- **CICS3270DMsgL24 -** General line 24 response message

Appendix E: Sample using the CICS 3270 via MQSeries service 371

Before you run the CICS 3270 signon sample

This sample will not run successfully until you have completed the following:

Domino Server system

Lotus Notes

- Add the MQEI Samples database (mqeisamp.nsf) to your workspace.

  Note If you are accessing a CICS for OS/2 server via a separate Client you need to add the MQEI Definition database to your workspace and change the Connection Manager in the CICS3270DServOS2 MQEI Service definition to the name of the CICS server defined in your CICS Client initialization file.

  Note If you are accessing a CICS for Open Systems server and are not using server name CICSTCP, add the MQEI Definition database to your workspace and change the Connection Manager in the CICS3270DServOPN MQEI Service definition to the name of the CICS server defined in your CICS Client initialization file.

- Add the MQEI Definition database (mqeidata.nsf) to your workspace.

  Note The use of the MQEI Security database is optional in this sample. For more information, see "Running the CICS 3270 Signon sample" later in this appendix.

- Ensure the MQEI LSX is installed correctly.

CICS Client

- A CICS Client is required.
- Unless you are using a local server and its built-in client, you will need a separate CICS Client which you should start using the /n option. The /c option (providing your CICS userid and password) is allowed but unnecessary since the sample Notes application allows you enter your userid and password.

CICS Server system

- Make sure that you have a valid User ID and Authenticator (password) on your CICS Server and that the CICS Client and CICS Server are running.

For more information on system configurations, see "Possible system configurations" in Chapter 2.

Running the CICS 3270 signon sample

Signon to CICS

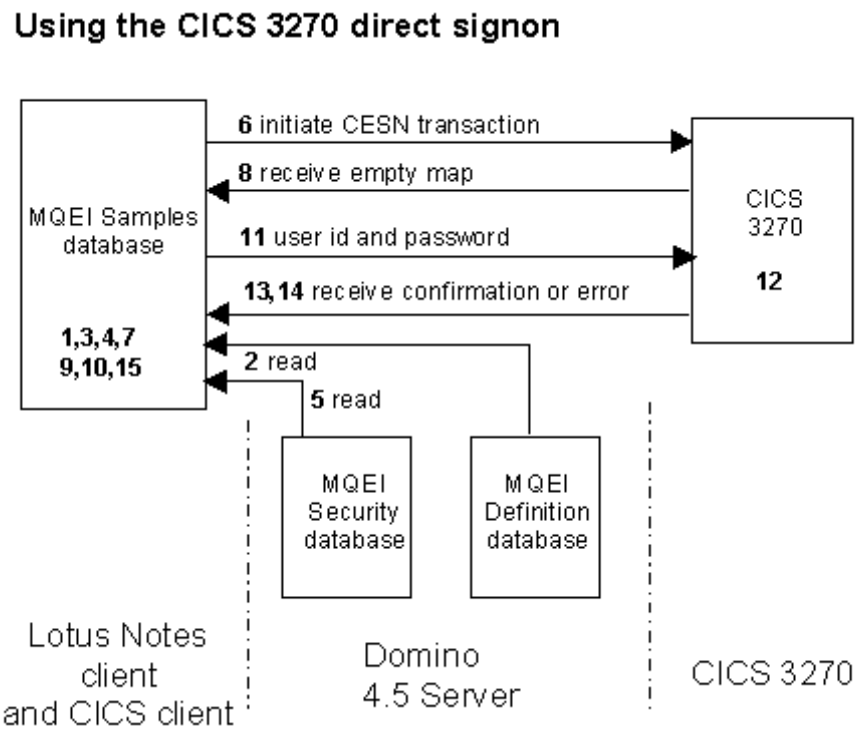
1. Check that all the appropriate setup work has been completed. See "Before you run the CICS 3270 signon sample" for details.
2. From your Notes workspace, open the MQEI Samples database (mqeisamp.nsf).
3. In the navigator pane, click "CICS 3270 direct".
   If you have set everything up correctly, a CICS 3270 Direct document should appear.
4. Select the type of CICS system you want to signon to from the dialogue list.
   For example, CICS for OS/2.
5. Enter your User Id and password into the fields on the form.

   Note Alternatively, you can create an entry in the MQEI Security database, update the appropriate MQEI Service definition to reference it and leave the User Id and password fields blank.
6. Click the button on the document labeled, "Signon".
7. The status bar gives you information about what is happening when you run the sample.
8. The Notes application displays the results returned by the CICS CESN transaction.

Change your password on CICS

1. Follow steps 1-5 as you would to signon to CICS.
2. Enter and verify your new password in the new password fields.
3. Click the button, labeled "Change password".
4. The status bar gives you information about what is happening when you run the sample.
5. The Notes application displays the results returned by the CICS CESN transaction.

How the CICS 3270 signon sample works



Steps (for signon):

1. When you click on the "CICS 3270 signon" icon, Notes creates a document using the "CICS 3270 Direct Sample: signon" form.
2. Before the document is opened, the Notes application creates non-system specific MQEI objects (EISession and EIMessage). It does this by invoking the appropriate create method which causes the MQEI LSX to read the MQEI Definition database for the details of the MQEI Message definitions used by this sample.
3. If the MQEI object creation is successful, the document is displayed.
4. When you enter the type of CICS system you want to signon to from the dialog list, the Notes application creates the appropriate EIService and connects to it.
5. If you didn't setup security data for the service, you need to enter your User Id and password. If you did, you can let the Notes application copy the User Id and password from the EIService.

Note The MQEI does this automatically for the password field because it is defined in the message as a SYSTEM_AUTHENTICATOR field.

6. When you click on either of the two buttons, "Signon" or "Change Password" the Notes application builds the message and sends a CESN transaction to CICS via the EPI.
7. To allow the CESN transaction time to run, the Notes application waits on a timer before attempting to receive a reply.
8. CESN returns an empty map to your Notes application.
9. When the timer pops, the Notes application receives the returned (empty) map from CESN.
10. The Notes application fills in the blank fields in the empty map with your User Id and password.
11. Your Notes application sends the populated map back to the CICS system and again waits on a timer.
12. The CESN transaction attempts to signon.

If the signon was successful:

13. CESN terminates and returns a one-line confirmation message.
14. When the timer pops, the Notes application receives the reply and displays the results in a message box.

If the signon was unsuccessful:

15. CESN returns a map containing an error message.
16. When the timer pops, the Notes application receives the reply and displays the error message in a message box.
17. The Notes application sends 'F3' to terminate the transaction and again waits on a timer.
18. CESN terminates and returns a one line termination message.
19. When the timer pops, the Notes application receives the reply and displays the termination message in a message box.

Error handling in the CICS 3270 signon sample

The CICS 3270 signon sample application uses variant message handling. If the expected (valid) message is not received (EIRC_WRONG_MESSAGE), then the invalid reply message is checked. Only if this is not accepted does the sample flag an error.

For more information, see "Variant Messages" in Chapter 4.

Error checking takes place throughout this sample, using both ON ERROR routines and EVENT handlers.

For more information, see "Error handling" in Chapter 4.

Appendix F Sample using a CICS 3270 direct service (FILEA)

This sample demonstrates how you can use the MQEI LSX in conjunction with a Notes database to interact with a CICS/ESA 3270 application via the CICS EPI interface.

For general information relating to all the MQEI samples, see "MQEI Samples" in Chapter 4.

This appendix describes the sample that uses the CICS 3270 direct service:

- Its design
- Preparation required
- How to run
- How it works

For more information on programming for the CICS 3270 direct service, see Chapter 10.

Restrictions

The following restrictions apply to this sample:

- CICS 3270 direct services are not supported when running the MQEI LSX under AIX, HP-UX, Sun Solaris, Windows 3.1, or WIN-OS/2. The CICS 3270 samples must be run from OS/2, Windows 95 or Windows NT.
- CICS 3270 direct services and CICS DPL direct services may not be used from the same Notes client or Domino Server agent at the same time. If you wish to run both CICS 3270 direct and CICS DPL direct samples, you must restart your Notes client.

Design of the CICS 3270 FILEA sample

This sample runs the CICS sample transaction ABRW on a CICS/ESA server. This CICS transaction browses FILEA (a file created by running the CICS/ESA sample). The transaction outputs four records at a time to a 3270 screen. This sample uses the MQEI to intercept the 3270 output and display it in a Notes document.

In summary:

This sample application uses the CICS AMNU and ABRW transactions.

1. When you click the "Start browse" button, the sample Notes application sends a transaction start request (AMNU) to CICS.
2. CICS runs the AMNU transaction which sends an empty AMNU map back to the Notes application.
3. The Notes application sends a transaction start request (ABRW) to CICS.
4. ABRW runs and sends back a map containing the first four records from FILEA.
5. The Notes application receives the data and displays it in the document.
6. Clicking the "More..." button, 'F1' is sent to your CICS system requesting the next four entries in the file.
7. Clicking the "End browse" button, 'Clear' is sent to your CICS system which ends the ABRW transaction.

Notes MQEI Samples database (mqeisamp.nsf)

This sample is implemented by the "CICS 3270 direct sample: FILEA" form.

Notes MQEI Definition database (mqeidata.nsf)

The MQEI Service definitions used by this sample are:

- **CICS3270DServMVS**
  This is a CICS 3270 direct service.
  - Service Step - AMNU, the initial transaction to be run.
  - Connection Manager - CICSSNA, the default name for a SNA connected CICS server.

The MQEI Message definitions used by this sample are:

- **CICS3270DMsgMVSA -** CICS BMS map DFH$AGA for AMNU
- **CICS3270DMsgMVSC -** CICS BMS map DFH$AGC for ABRW
- **CICS3270DMsgL23 -** General line 23 response message

Before you run the CICS 3270 FILEA sample

This sample will not run successfully until you have completed the following:

Domino Server system

Lotus Notes

- Add the MQEI Samples database (mqeisamp.nsf) to your workspace.
- If you are not using Server name CICSSNA, add the MQEI Definition database (mqeidata.nsf) to your workspace and change the CICS3270DServMVS MQEI Service definition to the name of the CICS server defined in your CICS client initialization file.

  Note The MQEI Security database is not used.
- Ensure the MQEI LSX is installed correctly.

CICS client

CICS Client must be available on your Domino Server system

If you have defined your client connection to CICS/ESA with ATTACHSEC=LOCAL you should start your CICS client with the /n option.

Otherwise, you must provide your userid and password for your CICS client, using the /c option.
For example:

```
CICSCLI /c=servername /u=userid /p=password
```

For more information, see your CICS Clients Administration documentation.

Note You must ensure that the Client has sufficient authority to run the AMNU and ABRW transactions.

MVS system

CICS/ESA server

Make sure that the CICS/ESA FILEA sample is installed and the CICS/ESA server is up and running.

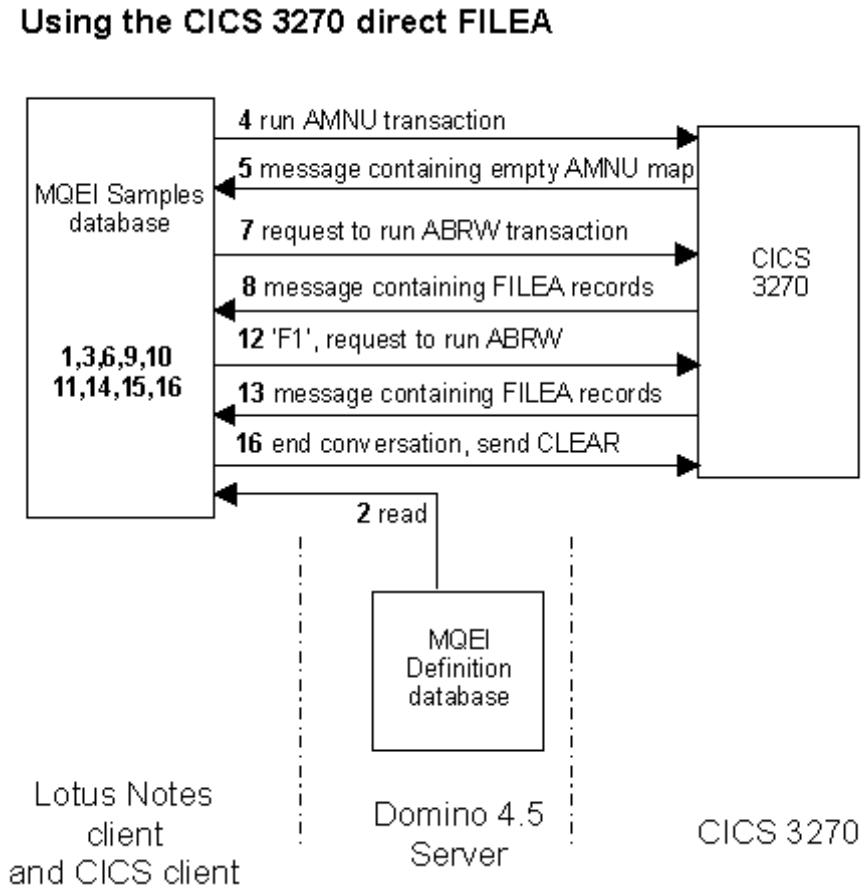
For more information on system configurations, see "Possible system configurations" in Chapter 2.

Running the CICS 3270 FILEA sample

1. Check all the appropriate setup work has been completed. See "Before you run the CICS 3270 FILEA sample" for details.
2. From your Notes workspace, open the MQEI Samples database.
3. From the navigator, select the "CICS 3270 FILEA" icon.
   An MQSeries Enterprise Integrator "CICS 3270 Direct Sample: FILEA" document is displayed on the screen. This document has three buttons:
   - Start browse - starts the ABRW transaction on your CICS system and gets the first four records from FILEA.
   - More... - gets the next four records from FILEA.
   - End browse - Stops the ABRW transaction on your CICS system.
4. Click "Start browse" to start browsing the CICS file and get the first four records.
   The Notes application will display the first four entries in the Notes document.
5. Click "More..." to display the next four records from the CICS file.

   Note You can continue to browse the CICS file by pressing the "More..." button until you get to the end of the file.
6. Click "Stop browse" to stop the ABRW transaction.

How the CICS 3270 FILEA sample works



Steps:

1. When you click on the "CICS 3270 FILEA" icon, Notes creates a document using the "CICS 3270 Direct Sample: FILEA" form.
2. Before the document is opened, the Notes application creates the MQEI objects (EISession, EIMessage and EIService). It does this by invoking the appropriate create method which causes the MQEI LSX to read the MQEI Definition database for the details of the MQEI Service and MQEI Message definitions used by this sample.
3. If the MQEI object creation is successful, the document is displayed.
4. When you click the "Start browse" button, the Notes application initiates the AMNU transaction on your CICS/ESA server. A Notes

timer is enabled here to allow the AMNU transaction to run before the Notes application attempts to get a reply from it.

5. When the Notes timer pops, the Notes application attempts to receive a response from CICS.
6. If it is successful, the sample Notes application receives the message from CICS containing an empty AMNU map.
7. The sample Notes application populates the empty AMNU map with the ABRW transaction ID and sends it to CICS to run the ABRW transaction.
8. The Notes timer is again enabled to allow ABRW time to run before the Notes application attempts to receive a reply.
9. When the Notes timer pops, the Notes application attempts to receive a response from CICS.
10. If it is successful, the sample Notes application receives the message from CICS containing the first four records in FILEA.

    Note If no message arrived back from CICS, the Notes application will retry once and will then ask you if you want to retry for another three seconds.
11. The sample Notes application displays the first four records of FILEA in the Notes document.
12. When you click the "More..." button, the Notes application continues the conversation with your CICS/ESA server, sends 'F1' to CICS, and waits on the timer.
13. On receipt of 'F1', ABRW retrieves the next four records from FILEA and sends a map containing them back to the sample Notes application.
14. When the Notes timer pops, the Notes application attempts to receive the response from CICS.
15. If it is successful, the sample Notes application receives the message from CICS containing the next four records in FILEA.

    Note If no message arrived back from CICS, your LotusScript application will retry once and will then ask you if you want to retry for another three seconds.
16. The Notes application displays the next four records of FILEA in the Notes document.
17. When you click the "End browse" button, the Notes application ends the ABRW transaction by sending 'CLEAR' and waiting on a timer.
18. ABRW sends back an empty AMNU map.
19. When the timer pops, the Notes application receives the AMNU map and sends 'CLEAR' again to end the conversation.

Error handling in the CICS 3270 FILEA sample

The CICS 3270 FILEA sample application uses variant message handling. If the expected (valid) message is not received (EIRC_WRONG_MESSAGE), then the invalid reply message is checked. Only if this is not accepted does the sample flag an error.

For more information, see "Variant Messages" in Chapter 4.

Error checking takes place throughout this sample, using both ON ERROR routines and EVENT handlers.

For more information, see "Error handling" in Chapter 4.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Sending your comments to IBM

MQSeries Enterprise Integrator for Lotus Notes

User's Guide - Release 1.0

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.