MQSeries®

Release Guide

Version 5.2

GC34-5761-01

MQSeries®

Release Guide

Version 5.2

GC34-5761-01

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix. Notices" on page 85.

Second edition (December 2000)

This edition applies to the following products:

- MQSeries for AIX®, V5.2
- MQSeries for AS/400®, V5.2
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Windows NT® and Windows® 2000, V5.2

and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this book

This book introduces all new function in MQSeries Version 5 Release 2.

Who this book is for

This book is for all users of any of the following products:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Windows NT and Windows 2000, V5.2

How to use this book

The descriptions of the new function in this book are in addition to, and must be used in conjunction with, the latest editions of the MQSeries cross-product books. These books are provided in softcopy form (Adobe Acrobat PDF and HTML) in the product package.

Terms used in this book

The terms “click” and “right-click” are used to describe item selection with the mouse. For keyboard alternatives refer to the Windows NT or Windows 2000 help.

Chapter 1. New function in all V5.2 products

This chapter introduces the new function that applies to all MQSeries V5.2 products. It contains these sections:

Performance enhancements

Version 5.2 introduces performance enhancements to all the MQSeries products. Although no general statement can be made about how these enhancements might affect your applications, their primary aim is to improve queue manager throughput. The enhancements comprise improvements to:

- The function accessed through the Message Queue Interface (MQI)
- Channel performance
- Writing persistent messages to the log
- Application initialization and termination

These improvements are described briefly below. You will recognize some of them in the function described later in this book, although others will be apparent only when you run your MQSeries applications. For a more detailed description of the performance measurements IBM® has made with MQSeries V5.2, see the *V5.2 Performance Report* on the MQSeries Web site at `http://www.ibm.com/software/mqseries/`.

MQI function

The function accessed through the MQI has been optimized for better performance and makes better use of system services. Optimized MQI functions also contribute to the other improvements described in this section.

Performance enhancements

Channel performance
Channels can now transfer messages more efficiently with a process called *pipelining*, which is described in "Multiple thread support — pipelining" on page 10.

Persistent messages
Persistent messages are written to logs and queue data files. Writing to the log is more efficient with MQSeries V5.2, thereby enhancing the throughput of persistent messages.

Application initialization and termination
Application initialization involves obtaining resources from MQSeries; those resources are returned when the application is terminated. These operations are now less time-consuming, which can mean a significant improvement if your application processes a few messages only.

The performance of your applications depends on many factors including, for example, whether messages are persistent or nonpersistent. That is why no general statement can be made about how your applications might be affected by the enhancements described briefly above.

MQSeries for Linux

MQSeries is now available for the Linux platform. See the *MQSeries for Linux Quick Beginnings* book for further information.

Application programming enhancements

The MQSeries V5.2 products provide several minor enhancements to the application programming interface.

MQCMD_LEVEL_520 CommandLevel value

MQCMD_LEVEL_520 is returned by:
- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Windows NT and Windows 2000, V5.2

Refer to the *MQSeries Application Programming Reference* book for more information about the CommandLevel attribute for the queue manager.

MQRFH2 — Version-2 rules and formatting header

MQRFH2, the version–2 rules and formatting header, is supported by the MQSeries V5.2 products.

MQRFH2 is the standard header:

- Used by the MQSeries Integrator broker.
- Used for message passing between MQSeries implementations of the Java™ Messaging Service (JMS) API.

Refer to the *MQSeries Application Programming Reference* book for a description of the MQRFH2 header.

Processing of CodedCharSetId fields in MQSeries headers

The *CodedCharSetId* field in the MQMD and other MQSeries headers can have a new value, MQCCSI_INHERIT. As a result, the behavior of the existing value MQCCSI_Q_MGR, and of the MQPUT, MQPUT1, and MQGET calls, is slightly altered. For more information about these changes, read the description of the *CodedCharSetId* field of the MQMD and other MQSeries headers in the *MQSeries Application Programming Reference* book.

The description of the MQCCSI_DEFAULT value of the *CodedCharSetId* field in the MQCFST and MQCFSL structures is now as follows:

MQCCSI_DEFAULT
Default coded character set identifier.

The *CodedCharSetId* of the data in the *String* field is defined by the *CodedCharSetId* field in the header structure that precedes the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH is at the start of the message.

For a description of the MQCFST and MQCFSL structures, see the *MQSeries Programmable System Management* book.

C++ support for MQCNO Version 2 and Version 3

Clients making connections with the ImqQueueManager class in the C++ interface can associate a channel definition with the connection. For details of the additions to the C++ interface, read the descriptions of the ImqQueueManager and ImqChannel classes in the *MQSeries Using C++* book. This function is supported only when the *Version* field in MQCNO structure is set to MQCNO_VERSION_2 or higher.

When the *Version* field in MQCNO structure is set to MQCNO_VERSION_3, the ImqQueueManager class in the C++ interface also supports connection tags. Refer to the *MQSeries Using C++* book for more information.

Applications using the IMS bridge

Applications using the IMS bridge can obtain the transaction state in IMS architected form if the *TranState* field in the MQIIH (IMS information header) is set to MQITS_ARCHITECTED.

Application programming

Refer to the description of the MQIIH in the *MQSeries Application Programming Reference* book for more information, and to the *MQSeries Application Programming Guide* for guidance on how to use the MQITS_ARCHITECTED constant.

Object Authority Manager (OAM) enhancements

This section describes some enhancements to the Object Authority Manager (OAM). See the *MQSeries System Administration* book for a complete description of the OAM.

Refreshing the OAM after changing a user's authorization

In versions of MQSeries prior to V5.2, most changes to a user's authorization group membership made at the operating system level were not implemented by the OAM immediately, but took effect only after the queue manager was stopped and restarted.

In MQSeries V5.2, you can request that the OAM's authorization group information be updated immediately, reflecting changes made at the operating system level, without needing to stop and restart the queue manager.

Note: When you change authorizations with the **setmqaut** command, the OAM supplied with MQSeries implements such changes immediately.

This change is implemented as follows:

- The OAM has a new function, MQZ_REFRESH_CACHE, with a corresponding entry point, MQZID_REFRESH_CACHE.
- The REFRESH SECURITY command is supported by the V5.2 products.
- MQZ_INIT_AUTHORITY returns MQZAS_VERSION_3 for the V5.2 products.

These enhancements are described fully in the remainder of this section.

MQZ_REFRESH_CACHE function

The authorization service provides an additional entry point for use by the queue manager:

MQZ_REFRESH_CACHE
Refresh all authorizations.

This function is supplied as an installable service

Refer to the *MQSeries Programmable System Management* book for a complete description of the authorization service interface and for information about installable services.

The reference information for the MQZ_REFRESH_CACHE installable service is:

This function is provided by an MQZAS_VERSION_3 authorization service component, and is invoked by the queue manager to refresh the list of authorizations held internally by the component.

The function identifier for this function (for MQZEP) is MQZID_REFRESH_CACHE (8L).

Syntax:

MQZ_REFRESH_CACHE
(QMgrName, ComponentData, Continuation, CompCode, Reason)

Parameters:

QMgrName (MQCHAR48) — input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×*ComponentDataLength*) — input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) — output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_REFRESH_CACHE this has the same effect as MQZCI_CONTINUE.
MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) — output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) — output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information on this reason code, see the *MQSeries Application Programming Reference* book.

C invocation:

```
MQZ_REFRESH_CACHE (QMgrName, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

RFRMQMAUT command for MQSeries for AS/400

The RFRMQMAUT, Refresh Security command is added to the CL Security Commands for MQSeries for AS/400. The syntax of this command is:

RFRMQMAUT MQMNAME (*queue-manager-name*)

Refer to the MQSeries for AS/400 online help for more information about CL commands and their syntax.

REFRESH SECURITY — Programmable Command Format (PCF)

The reference material for the PCF of the REFRESH SECURITY command is provided below. Refer to the *MQSeries Programmable System Management* book for more information about PCFs. The list of PCF commands and responses is extended with:

Security command
"Refresh Security"

The Refresh Security (MQCMD_REFRESH_SECURITY) command refreshes the list of authorizations held internally by the authorization service component.

This PCF is supported if you are using the V5.2 products only.

Required parameters:
None

Optional parameters:
None

Error codes: In addition to the values for any command shown in the *MQSeries Programmable System Management* book, for this command the following may be returned in the response format header:

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

REFRESH SECURITY MQSC command

The syntax for the V5.2 products is:

The optional * parameter specifies that the security refresh is to be performed for all resource classes.

Refer to the *MQSeries MQSC Command Reference* book for a complete description of the REFRESH SECURITY command. The Table of "Commands for security administration" in the same book is amended to show that:

- This command is supported on AIX, OS/400®, HP-UX, Linux, OS/390®, Sun Solaris, and Windows NT only.
- There is a PCF equivalent for REFRESH SECURITY.

Authorization data held on local queue

Queue managers running the OAM provided with MQSeries V5.2 store authorization data on a local queue, called SYSTEM.AUTH.DATA.QUEUE. This name is added to the list of reserved queue names in the *MQSeries MQSC Command Reference* book.

Authorization data is managed by the amqzfuma process (amqzfuma.exe for Windows NT and Windows 2000). When you stop and remove queue managers manually, as described in Appendix E of the *MQSeries System Administration* book, `amqzfuma` (`AMQZFUMA.EXE`) should be ended after `amqzlaa0` (`AMQZLAA0.EXE`) and before `amqzxma0` (`AMQZXMA0.EXE`). AMQZFUMA is added to the section in the *MQSeries for AS/400 System Administration* book, listing the MQSeries tasks that run when a queue manager is running. The function of AMQZFUMA is to "Manage authorization data".

The function provided by the OAM is unaffected by this change and queue managers are automatically created to use the Version 5.2 OAM as the default authorization service component. This version creates no new authorization files, and existing files are no longer updated or deleted. The section entitled "Authorization files" in Chapter 10 of the *MQSeries System Administration* book no longer applies.

Migration

All authorization data is migrated from the authorization files to the authorization queue the first time you restart the queue manager after installing Version 5.2. If the OAM detects a missing file and:

- The authorization applies to a single object, the OAM gives the mqm group (Administrator for Windows NT and Windows 2000) access to the object and continues with the migration. Message AMQ5528 is written to the queue manager's error log. Refer to the *MQSeries Messages* book for more information about message AMQ5528.
- The authorization applies to a class of objects, the OAM stops the migration. The queue manager does not start until the file has been replaced.

When you still want to store authorization data in files

This section tells you how you can continue to store authorization data in files. However, if you do so, the performance of the OAM can be affected. Storing authorization data on a local queue reduces the time required to check an authorization.

The default OAM service module is **amqzfu** (amqzfu.dll for Windows NT and Windows 2000). Version 5.2 also provides the previous service module as **amqzfu0** (amqzfu0.dll). There are two ways in which you can use the previous module to continue to store authorization data in files:

- Modify the **Module** attribute in the `ServiceComponent` stanza of the qm.ini file (registry entry for Windows NT and Windows 2000) to use amqzfu0 (amqzfu0.dll). Note that this option is possible only for queue managers created with a version of MQSeries prior to V5.2.
- Replace the amqzfu module (amqzfu.dll) by the previous version. For example, you can do this by:
  1. Removing the *new* amqzfu module
  2. Renaming amqzfu0 as amqzfu

  Note: You can restore the *new* amqzfu module from the copy provided as amqzfu1 (amqzfu1.dll).

Note

Once you have created or restarted a queue manager with the *new* amqzfu module, you can no longer replace the amqzfu module (amqzfu.dll) with the previous version. The migration process, described above, is not reversible.

You can view authorization data with the **dspmqaut** command. Refer to the *MQSeries System Administration* book for a complete description of this command.

Installation

This section tells you about installing support for Java on MQSeries and MQSeries clients for the V5.2 products.

Support for Java on MQSeries

Support for Java on MQSeries is separately installable from the CD-ROM included in this product package. Alternatively, you can download support for Java on MQSeries from the MQSeries Web site at `http://www.ibm.com/software/mqseries/`, where the latest version of this support is always available.

MQSeries clients

The following MQSeries clients are available on the Clients CD-ROM included in the V5.2 product package:

- MQSeries client for AIX
- MQSeries client for HP-UX
- MQSeries client for Linux
- MQSeries client for Sun Solaris
- MQSeries client for Windows NT and Windows 2000
- MQSeries client for Windows 95 and Windows 98

Other clients can be obtained from the MQSeries Web site.

Enhanced channel support

The V5.2 products provide enhancements to MQSeries channel support that:

- Allow a message channel agent (MCA) to use multiple threads.
- Allow channel send exit programs to add their own data prior to transmission.

Advice is added for "User IDs with encrypted passwords" on page 14.

Multiple thread support — pipelining

With MQSeries V5.2, you can optionally allow a message channel agent (MCA) to transfer messages using multiple threads. This process, called *pipelining*, enables the MCA to transfer messages more efficiently, with fewer wait states, which improves channel performance. For Version 5.2, each MCA is limited to a maximum of two threads.

Refer to the *MQSeries Intercommunication* book for a complete description of channels and how they operate.

You control pipelining with the *PipeLineLength* parameter in the qm.ini file. This parameter is added to the `CHANNELS` stanza:

PipeLineLength=1 | *number*
: This attribute specifies the maximum number of concurrent threads a channel will use. The default is 1. Any value greater than 1 will be treated as 2.

With MQSeries for Windows NT and Windows 2000, V5.2, use the MQSeries Services snap-in to set the *PipeLineLength* parameter in the registry. Refer to the *MQSeries System Administration* book for a complete description of the `CHANNELS` stanza.

Notes:

1. *PipeLineLength* applies only to the V5.2 products.
2. Pipelining is effective only for TCP/IP channels.

When you use pipelining, the queue managers at both ends of the channel must be configured to have a *PipeLineLength* greater than 1.

Channel exit considerations

Note that pipelining can cause some exit programs to fail, because:

- Exits might not be called serially.
- Exits might be called alternately from different threads.

Check the design of your exit programs before you use pipelining:

- Exits must be reentrant at all stages of their execution.
- When you use MQI calls, remember that MQI handles are thread-specific.

Consider a message exit that opens a queue and uses its handle for MQPUT calls on all subsequent invocations of the exit. This fails in pipelining mode because the exit is called from different threads. To avoid this failure, keep a queue handle for each thread and check the thread identifier each time the exit is invoked.

Channel send exit programs — reserving space

You can use send and receive exits to transform the data before transmission. With MQSeries V5.2, channel send exit programs can add their own data about the transformation by reserving space in the transmission buffer. This data is processed by the receive exit program and then removed from the buffer. For example, you might want to encrypt the data and add a security key for decryption. Refer to the *MQSeries Intercommunication* book for a complete description of channel exit programs.

How you reserve space and use it

Channel exit programs are passed an MQCXP parameter block. With MQSeries V5.2 a new field, *ExitSpace*, is added to the MQCXP structure, as described in "MQCXP — Channel exit parameter structure" on page 12.

When the send exit program is called for initialization, set the *ExitSpace* field to the number of bytes to be reserved. *ExitSpace* can be set only during initialization, that is when *ExitReason* has the value MQXR_INIT. When the send exit is invoked immediately before transmission, with *ExitReason* set to MQXR_XMIT, *ExitSpace* bytes are reserved in the transmission buffer.

The send exit need not use all of the reserved space. It can use less than *ExitSpace* bytes or, if the transmission buffer is not full, the exit can use more than the amount reserved. Refer to the *MQSeries Intercommunication* book for information about the maximum transmission size. When setting the value of *ExitSpace*, you must leave at least 1 KB for message data in the transmission buffer. Note that channel performance can be affected if reserved space is used for large amounts of data.

What happens at the receiving end of the channel

Channel receive exit programs must be set up to be compatible with the corresponding send exits. Receive exits must know the number of bytes in the reserved space and must remove the data in that space.

Multiple send exits

You can specify a list of send and receive exit programs to be run in succession. MQSeries maintains a total for the space reserved by all of the send exits. This total space must leave at least 1 KB for message data in the transmission buffer.

The following example shows how space is allocated for three send exits, called in succession:

1. When called for initialization:
   - Send exit A reserves 1 KB.
   - Send exit B reserves 2 KB.
   - Send exit C reserves 3 KB.
2. The maximum transmission size is 32 KB and the user data is 5 KB long.
3. Exit A is called with 5 KB of data; up to 27 KB are available, because 5KB is reserved for exits B and C. Exit A adds 1KB, the amount it reserved.
4. Exit B is called with 6 KB of data; up to 29 KB are available, because 3KB is reserved for exit C. Exit B adds 1KB, less than the 2KB it reserved.
5. Exit C is called with 7 KB of data; up to 32 KB are available. Exit C adds 10K, more than the 3KB it reserved. This is valid, because the total amount of data, 17 KB, is less than the 32KB maximum.

MQCXP — Channel exit parameter structure

Refer to the *MQSeries Intercommunication* book for a complete description of the MQCXP structure.

The following value is added to the description of the *Version* field:

MQXCP_VERSION_5
Version-5 channel exit parameter structure.

The field has this value in the following environments: AIX, HP-UX, Linux, AS/400, Sun Solaris, Windows NT. Send and receive exits are also supported for MQSeries clients.

A new field, *ExitSpace*, is added to the end of the MQCXP structure.

ExitSpace (MQLONG)
Number of bytes in transmission buffer reserved for exit to use.

This field is relevant only for a send exit. It specifies the amount of space in bytes that the MCA will reserve in the transmission buffer for the exit to use. This allows the exit to add data to the transmission buffer, for use by a complementary receive exit at the other end. The data added by the send exit must be removed by the receive exit.

Note: This facility should not be used to send large amounts of data, as this may degrade the performance of the channel.

By setting *ExitSpace*, the exit is guaranteed that there will always be at least that number of bytes available in the transmission buffer for the exit to use. However, the exit can use less than the amount reserved, or more than the amount reserved if there is space available in the transmission buffer. The exit space in the buffer is provided after the existing data.

ExitSpace can be set by the exit only when *ExitReason* has the value MQXR_INIT; in all other cases the value returned by the exit is ignored. On input to the exit, *ExitSpace* is zero for the MQXR_INIT call, and is the value returned by the MQXR_INIT call in other cases.

If any of the following applies, the MCA outputs an error message and closes the channel:

- The *ExitSpace* value returned by the MQXR_INIT call is negative.
- When MQXR_INIT is called, there is less than 1 KB available in the transmission buffer for message data after reserving the requested exit space for all of the send exits in the chain.
- During data transfer the exits in the send exit chain allocate more user space than they reserved such that there is less than 1 KB left in the transmission buffer for message data.

The limit of 1 KB allows the control and administrative flows that occur on a channel to be processed by any chain of send exits without a need for them ever to be segmented.

This is an input/output field to the exit if *ExitReason* is MQXR_INIT, and an input field in all other cases. The field is not present if *Version* is less than MQCXP_VERSION_5.

C declaration: For illustration purposes, only the end of the C declaration is shown.

```
  MQLONG    ExitNumber;        /* Exit number */
  MQLONG    ExitSpace;         /* Number of bytes in transmission
                                  buffer for exit to use */
} MQCXP;
```

Channel support

User IDs with encrypted passwords

The following advice is added to the description of the **UserID (USERID)** channel attribute in Chapter 6 of the *MQSeries Intercommunication* book, and to the description of the **USERID** parameter to the **DEFINE CHANNEL** command in Chapter 2 of the *MQSeries MQSC Command Reference* book.

On the receiving end, if passwords are kept in encrypted format and the LU 6.2 software is using a different encryption method, an attempt to start the channel fails with invalid security details. You can avoid this by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

Support for DHCP in queue manager clusters

MQSeries V5.2 includes two enhancements pertaining to the use of clusters.

- The first enables you to define a cluster-receiver channel without specifying your queue manager's network address.
- The second enables you to define a cluster-sender channel without specifying the name of the repository queue manager.

The purpose of these two enhancements is further to simplify the task of the system administrator. It is no longer necessary for you to know the network address of your queue manager, nor the names of the other queue managers in the cluster.

This section describes these two enhancements. Read it in conjunction with the *MQSeries Queue Manager Clusters* book, which describes the existing function in full. Changes to the *MQSeries MQSC Command Reference* and *MQSeries Programmable System Management* books, resulting from these enhancements, are described at the end of this section.

When you don't know your queue manager's network address

When the network protocol that you are using is TCP/IP it is no longer necessary to specify the network address of your queue manager when you define a cluster-receiver channel. You can issue the DEFINE CHANNEL command without supplying a value for CONNAME. MQSeries generates a CONNAME for you, assuming the default port and using the current IP address of the system. The generated CONNAME is always in the dotted-decimal form, rather than in the form of an alphanumeric DNS host name.

This facility is useful when you have machines using Dynamic Host Configuration Protocol (DHCP). In earlier releases of MQSeries, you needed to

update definitions manually each time DHCP issued a new IP address. With MQSeries V5.2 you do not need to make any changes to your definitions when your IP address changes.

Example of how to use this

Consider the second task described in the *MQSeries Queue Manager Clusters* book, 'Adding a new queue manager to a cluster'. Let us suppose that the queue manager to be added, PARIS, is on a system that runs DHCP. The steps you must take to perform this task are as follows.

1. Prepare the PARIS queue manager: This step is unaltered.

2. Determine which full repository PARIS should refer to first: This step is unaltered.

Note: The remaining steps may be performed in any order. Before proceeding with these steps make sure that queue manager PARIS is started. Also, start a listener program on queue manager PARIS.

3. Define a CLUSRCVR channel on queue manager PARIS: On PARIS, define:

```
DEFINE CHANNEL(TO.PARIS) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CLUSTER(INVENTORY)
```

This advertises the queue manager's availability to receive messages from other queue managers in the cluster INVENTORY. There is no need to specify the CONNAME, although you could, if you wish, specify `CONNAME(' ')`. MQSeries generates the CONNAME value using the current IP address of the system.

4. Define a CLUSSDR channel on queue manager PARIS: This step is unaltered. On PARIS, make the following definition:

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
```

The cluster achieved by task 2

The cluster set up by this task looks like this:

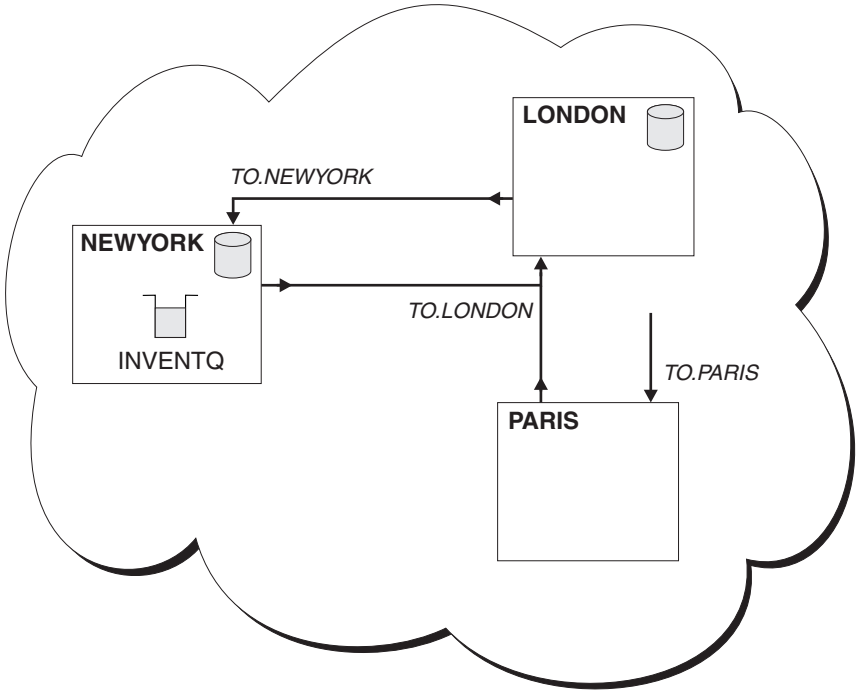


Figure 1. The INVENTORY cluster with three queue managers

By making only two definitions, a CLUSRCVR definition and a CLUSSDR definition, we have added the queue manager PARIS to the cluster.

What are the effects?
Because you specified a blank for the CONNAME on the CLUSRCVR definition, MQSeries generates a CONNAME from the IP address of the system. Only the generated CONNAME is stored in the repositories. Other queue managers in the cluster do not know that the CONNAME was originally blank.

If you issue the DISPLAY CLUSQMGR command you will see the generated CONNAME. However, if you issue the DISPLAY CHANNEL command from the local queue manager, you will see that the CONNAME is blank.

If the queue manager is stopped and restarted with a different IP address, because of DHCP, MQSeries regenerates the CONNAME and updates the repositories accordingly.

Note

Auto-defined cluster-sender channels take their attributes from those specified in the corresponding cluster-receiver channel definition on the receiving queue manager. Even if there is a manually-defined cluster-sender channel, its attributes are automatically modified to ensure that they match those on the corresponding cluster-receiver definition. Beware, therefore, that you could, for example, define a CLUSRCVR without specifying a port number in the CONNAME parameter, whilst manually defining a CLUSSDR that does specify a port number. When the auto-defined CLUSSDR replaces the manually defined one, the port number (taken from the CLUSRCVR) becomes blank. The default port number would be used and the channel would fail.

Migration

This enhancement to MQSeries raises three migration questions:

- How can I upgrade an existing queue manager to a system that uses DHCP?
- How will queue managers on previous versions of MQSeries handle it?
- Can I choose a queue manager to be a repository if it is on a system that uses DHCP?

Upgrading an existing queue manager to a system that uses DHCP: These are the steps:

1. Suspend the queue manager
2. Stop the cluster-receiver channel on the queue manager
3. Alter the CLUSRCVR definition to remove the CONNAME or set it to blank
4. Restart the queue manager

How queue managers on previous versions of MQSeries handle it: Other queue managers in a cluster do not know that the CLUSRCVR was defined with a blank CONNAME. The repositories notify them of the generated CONNAME only.

Cluster repositories and DHCP: If a queue manager is to host a cluster's repository, you need to know its host name or its IP address. You have to specify this information in the CONNAME parameter when you make the CLUSSDR definition on other queue managers joining the cluster.

If you use DHCP, the IP address is subject to change because DHCP allocates a new IP address each time you restart a system. Therefore, it would not be possible to specify the IP address in the CLUSSDR definitions. Even if all your CLUSSDR definitions specified the hostname rather than the IP address, the

definitions would still not be reliable. This is because DHCP does not necessarily update the DNS directory entry for the host, with the new address.

> **Note**
> Unless you have installed software that guarantees to keep your DNS directory up-to-date, you should not nominate queue managers as repositories if they are on systems that use DHCP.

When you don't know the repository queue manager's name

It is no longer necessary to specify the repository queue manager's name when you define a cluster-sender channel. So long as you know the naming convention used for channels in your cluster you can make a CLUSSDR definition using the +QMNAME+ construction. MQSeries substitutes the correct repository queue manager name in place of +QMNAME+. The resulting channel name is truncated to 20 characters.

Clearly this works only if your convention for naming channels includes the name of the queue manager. For example, if you have a repository queue manager called QM1 with a cluster-receiver channel called TO.QM1.ALPHA, another queue manager can define a cluster-sender channel to this queue manager, but specify the channel name as TO.+QMNAME+.ALPHA.

If you use the same naming convention for all your channels, be aware that only one +QMNAME+ definition can exist at one time.

Example of how to use this

Again, consider the second task described in the *MQSeries Queue Manager Clusters* book. Let us suppose that you do not know the name of the repository queue manager on the system in London, but you do know the channel-naming convention in use. The steps you must take to perform this task are as follows.

1. Prepare the PARIS queue manager: This step is unaltered.

2. Determine which full repository PARIS should refer to first: This step is unaltered.

Note: The remaining steps may be performed in any order. Before proceeding with these steps make sure that queue manager PARIS is started. Also, start a listener program on queue manager PARIS.

3. Define a CLUSRCVR channel on queue manager PARIS: On PARIS, define:

```
DEFINE CHANNEL(TO.PARIS) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(PARIS.CHSTORE.COM) CLUSTER(INVENTORY)
```

4. Define a CLUSSDR channel on queue manager PARIS: Although you know the naming convention for channels in the cluster, you do not know the name of the repository queue manager. On PARIS, make the following definition:

```
DEFINE CHANNEL(TO.+QMNAME+) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
```

The cluster achieved by task 2

Again, we have added the queue manager PARIS to the cluster.

What are the effects?

On the PARIS queue manager, the CLUSSDR definition containing the string +QMNAME+ is stored in the partial repository. On the system whose CONNAME is `LONDON.CHSTORE.COM` (which you specified), MQSeries resolves the +QMNAME+ to the queue manager name (LONDON). MQSeries then matches the definition for a channel called TO.LONDON to the corresponding CLUSRCVR definition.

MQSeries sends back the resolved channel name to the PARIS queue manager. At PARIS, the CLUSSDR channel definition for the channel called TO.+QMNAME+ is replaced by an internally-generated definition for TO.LONDON. This definition contains the resolved channel name, but otherwise is the same as the +QMNAME+ definition that you made. The cluster repositories are also brought up-to-date with the channel definition with the newly-resolved channel name.

Notes:

1. The channel created with the +QMNAME+ name becomes inactive immediately. It is never used to transmit data.
2. Channel exits may see the channel name change between one invocation and the next.

Migration

It is not possible to define a CLUSSDR channel using the +QMNAME+ construction to a queue manager on an earlier version of MQSeries. The queue manager would not recognize the construction and would not be able to find the channel.

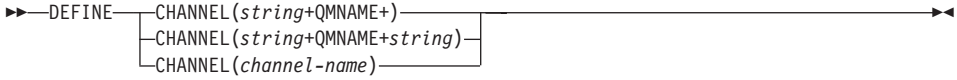
MQSC command syntax changes

These enhancements to the use of clusters alter the syntax of the DEFINE CHANNEL command. Read this description in conjunction with the description of the DEFINE CHANNEL command in the *MQSeries MQSC Command Reference* book.

Note: These enhanced options are valid only if TRTYPE is TCP.

Cluster-sender channel

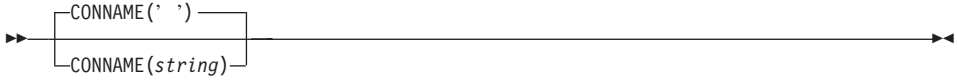
The parameter options for CHANNEL are changed. All other keywords and parameters are unaltered.



Note: The string +QMNAME+ must be complete and in upper case. Another string can follow the +QMNAME+, but the resolved *channel–name* will be truncated to 20 characters.

Cluster-receiver channel

The parameter options for CONNAME are changed. All other keywords and parameters are unaltered.



Programmable Command Formats (PCFs)

These enhancements to the queue manager clusters alter the definitions for two of the parameters for the Create Channel (MQCMD_CREATE_CHANNEL) command. Refer to the *MQSeries Programmable System Management* book for a complete description of this command.

ChannelName (MQCFST)
: Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

  For a *ChannelType* of MQCHT_CLUSRCVR and a *TransportType* of MQXPT_TCP, this parameter can contain the string +QMNAME+.

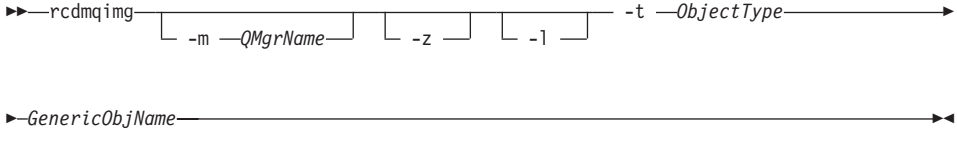
ConnectionName (MQCFST)
: Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

For a *ChannelType* of MQCHT_CLUSSDR and a *TransportType* of MQXPT_TCP, this parameter can be blank.

Command rcdmqimg issues media recovery messages synchronously

MQSeries V5.2 simplifies the process of managing your log files for recovery and restart. The **rcdmqimg** command writes an image of one or more MQSeries objects to the log for use in media recovery. In the MQSeries V5.2 products, **rcdmqimg** has an additional flag, the –l flag, that causes messages AMQ7467 and AMQ7468 to be issued when **rcdmqimg** completes. Although the queue manager will still issue these two messages when it periodically takes a checkpoint, you do not need to wait for it to do so when you use **rcdmqimg** with the –l flag.

Message AMQ7467 contains the name of the oldest log file needed to restart the queue manager. Message AMQ7468 contains the name of the oldest log file needed to do media recovery. You can use this information to archive and delete files older than those identified by these messages. Read the *MQSeries System Administration* book for more information about managing log files and for a complete description of the **rcdmqimg** command. The revised syntax of this command is:



Optional parameters

Add the following optional parameter:

–l Requests that the log file information for the queue manager is issued when the command completes.

Two messages, AMQ7467 and AMQ7468, are issued to the error log and the standard error destination. These messages contain the names of the log files that must be available to restart the queue manager and to perform media recovery for all objects owned by the queue manager.

Notes:

1. When issuing a sequence of **rcdmqimg** commands, include the –l parameter only on the last command in the sequence. The log file information is then gathered once only.

2. When both the –z and –l parameters are included, the AMQ7467 and AMQ7468 messages are sent to the error logs but not to the standard error destination. If no objects are processed, no messages are issued to either location.

RCDMQMIMG command for MQSeries for AS/400

The RCDMQMIMG, Record MQM Object Image command has an additional parameter, DSPJRNDATA, described below. Refer to the *MQSeries for AS/400 System Administration* book for a discussion of media images and to the MQSeries for AS/400 online help for more information about CL commands and their syntax.

The help text for the DSPJRNDATA parameter is:

Display Journal Receiver Data (DSPJRNDATA) — Help

Specifies whether additional messages should be written to the job log when the command completes to inform the user about the journal receivers that are still required by MQSeries.

The possible values are:

***NO** No messages are written to the job log. This is the default option.

***YES** Messages are sent to the job log when the command completes. The messages contain details about the journal receivers that are required by MQSeries.

Queue manager cluster workload exits

MQSeries V5.2 enhances the operation of cluster workload exit programs. See the *MQSeries Queue Manager Clusters* book for guidance on cluster workload management and workload exit programs.

Dynamic space allocation for workload data records

MQSeries makes an initial space allocation for the repository that holds workload management records. With V5.2, the allocation is increased dynamically. In previous versions, the space allocation remained fixed.

When the size of the repository is dynamic, the offset fields in the workload records contain handles. When the total size is fixed, these fields contain true physical offsets. This affects how your workload exit program navigates the records in the repository.

Dynamic space allocation is the default option, but if you have written your own cluster workload exit program for versions prior to V5.2, and you do not want to change it, you must set the repository size to be STATIC.

Note
If you want to change your cluster workload exit program to navigate records in a dynamically allocated repository, you must use the MQXCLWLN call, described in "Navigating cluster workload records".

You set the repository size with the *ClusterCacheType* parameter in the qm.ini file. This parameter is added to the `TuningParameters` stanza:

ClusterCacheType=DYNAMIC|STATIC
Specify `DYNAMIC` if you want the size of the workload record repository to be increased automatically. This is the default option.

Specify `STATIC` if you want the total size of the workload record repository to remain fixed at its initial value.

With MQSeries for Windows NT and Windows 2000, V5.2, use the MQSeries Services snap-in to set the *ClusterCacheType* parameter in the registry.

Refer to the *MQSeries System Administration* book for more information about the qm.ini file.

Navigating cluster workload records

This section describes the use of the MQXCLWLN call for navigating cluster workload records and also documents:

- The current version of "MQWXP — Cluster workload exit parameter structure" on page 26
- Changes to "MQWDR - Cluster workload destination-record structure" on page 37
- Changes to "MQWQR - Cluster workload queue-record structure" on page 37
- Changes to "MQWCR - Cluster workload cluster-record structure" on page 38

You can use the MQXCLWLN call for both types of repository, that is, whether the *ClusterCacheType* is set to `DYNAMIC` or `STATIC`. Your workload exit program passes information about a record in the chain and the call returns the address of the next record in the chain. The MQXCLWLN call is defined in "MQXCLWLN - Cluster workload navigate records" on page 24.

If your cluster workload exit program expects fields to contain true physical offsets, it will not work when the *ClusterCacheType* is set to `DYNAMIC`. When your exit program is called with the *ExitReason* field in the version-2 MQWXP

exit parameter structure set to MQXR_INIT, return the value MQXR2_STATIC_CACHE in the *ExitResponse2* field. This response causes the queue manager to write to the error log or to display a message on the operator console. The queue manager does not call the exit again until *ExitReason* is set to MQXR_TERM.

See "MQWXP — Cluster workload exit parameter structure" on page 26 for a full description of the version-2 MQWXP exit parameter structure, including the use of the *ExitResponse2* field.

Refer to the *MQSeries Queue Manager Clusters* book for further information about the cluster workload exit and the data structures it uses.

MQXCLWLN - Cluster workload navigate records

The MQXCLWLN call is used to navigate through the chains of MQWDR, MQWQR, and MQWCR records stored in the cluster cache. The cluster cache is an area of main storage used to store information relating to the cluster.

This call is supported in the following environments: AIX, HP-UX, Linux, AS/400, Sun Solaris, and Windows NT.

Syntax:

```
MQXCLWLN (ExitParms, CurrentRecord, NextOffset, NextRecord,
          CompCode, Reason)
```

Parameters: The MQXCLWLN call has the following parameters.

ExitParms (MQWXP) – input/output
Exit parameter structure.

This is the parameter structure that was passed to the exit when the exit was invoked.

CurrentRecord (MQPTR) – input
Address of current record.

This is the address of the record currently being examined by the exit. The record must be one of the following types:

- Cluster workload destination record (MQWDR)
- Cluster workload queue record (MQWQR)
- Cluster workload cluster record (MQWCR)

NextOffset (MQLONG) – input
Offset of next record.

This is the offset of the next record or structure. *NextOffset* is the value of the appropriate field in the current record, and must be the value of one of the following fields:

- *ChannelDefOffset* field in MQWDR
- *ClusterRecOffset* field in MQWDR
- *ClusterRecOffset* field in MQWQR
- *ClusterRecOffset* field in MQWCR

NextRecord (MQPTR) – output
Address of next record or structure.

This is the address of the next record or structure. If *CurrentRecord* is the address of an MQWDR, and *NextOffset* is the value of the *ChannelDefOffset* field, *NextRecord* is the address of the channel definition structure MQCD.

If there is no next record or structure, the queue manager sets *NextRecord* to the null pointer, and the call returns completion code MQCC_WARNING and reason code MQRC_NO_RECORD_AVAILABLE.

CompCode (MQLONG) – output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_WARNING
Warning (partial completion).
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:
MQRC_NO_RECORD_AVAILABLE
(2359, X'937') No record available.

If *CompCode* is MQCC_FAILED:
MQRC_CURRENT_RECORD_ERROR
(2357, X'935') Current-record parameter not valid.
MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.
MQRC_NEXT_OFFSET_ERROR
(2358, X'936') Next-offset parameter not valid.
MQRC_WXP_ERROR
(2356, X'934') Workload exit parameter structure not valid.

Usage notes:

1. If the cluster cache is dynamic, the MQXCLWLN call *must* be used to navigate through the records; the exit will terminate abnormally if simple pointer-and-offset arithmetic is used to navigate through the records.

   If the cluster cache is static, the MQXCLWLN call need not be used to navigate through the records. However, it is recommended that MQXCLWLN be used even when the cache is static, as this allows migration to a dynamic cache without needing to change the cluster workload exit.

C invocation:

```
MQXCLWLN (&ExitParms, CurrentRecord, NextOffset, &NextRecord,
         &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQWXP   ExitParms;      /* Exit parameter structure */
MQPTR   CurrentRecord;  /* Address of current record */
MQLONG  NextOffset;     /* Offset of next record */
MQPTR   NextRecord;     /* Address of next record or structure */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

MQWXP — Cluster workload exit parameter structure

The following table summarizes the fields in the structure.

Table 1. Fields in MQWXP

Field	Description	Page
StrucId	Structure identifier	27
Version	Structure version number	27
ExitId	Type of exit	28
ExitReason	Reason for invoking exit	28
ExitResponse	Response from exit	29
ExitResponse2	Secondary response from exit	30
Feedback	Feedback code	30
ExitUserArea	Exit user area	31
ExitData	Exit data	31
MsgDescPtr	Address of message descriptor (MQMD)	31
MsgBufferPtr	Address of buffer containing some or all of the message data	32
MsgBufferLength	Length of buffer containing message data	32
MsgLength	Length of complete message	32

Table 1. Fields in MQWXP (continued)

Field	Description	Page
QName	Name of queue	32
QMgrName	Name of local queue manager	32
DestinationCount	Number of possible destinations	32
DestinationChosen	Destination chosen	33
DestinationArrayPtr	Address of an array of pointers to destination records (MQWDR)	33
QArrayPtr	Address of an array of pointers to queue records (MQWQR)	33
Context	Context information	33
CacheType	Type of cluster cache	34

The MQWXP structure describes the information that is passed to the cluster workload exit.

This structure is supported in the following environments: AIX, Compaq Tru64 UNIX, HP-UX, OS/390, AS/400, OS/2, Sun Solaris, and Windows NT.

Fields:

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQWXP_STRUC_ID
Identifier for cluster workload exit parameter structure.

For the C programming language, the constant MQWXP_STRUC_ID_ARRAY is also defined; this has the same value as MQWXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)
Structure version number.

The value is one of the following:

MQWXP_VERSION_1
Version-1 cluster workload exit parameter structure.

This version is supported in all environments.

MQWXP_VERSION_2
Version-2 cluster workload exit parameter structure.

This version is supported in the following environments: AIX, HP-UX, Linux, AS/400, Sun Solaris, Windows NT.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions that follow. The following constant specifies the version number of the current version:

MQWXP_CURRENT_VERSION
Current version of cluster workload exit parameter structure.

Note: When a new version of the MQWXP structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

ExitId (MQLONG)
Type of exit.

This indicates the type of exit being called. The value is:

MQXT_CLUSTER_WORKLOAD_EXIT
Cluster workload exit.

This type of exit is supported in the following environments: AIX, Compaq Tru64 UNIX, HP-UX, OS/2, OS/390, AS/400, Sun Solaris, and Windows NT.

This is an input field to the exit.

ExitReason (MQLONG)
Reason for invoking exit.

This indicates the reason why the exit is being called. Possible values are:

MQXR_INIT
Exit initialization.

This indicates that the exit is being invoked for the first time. It allows the exit to acquire and initialize any resources that it may need (for example: main storage).

MQXR_TERM
Exit termination.

This indicates that the exit is about to be terminated. The exit should free any resources that it may have acquired since it was initialized (for example: main storage).

MQXR_CLWL_OPEN
Called from MQOPEN processing.

MQXR_CLWL_PUT
Called from MQPUT or MQPUT1 processing.

MQXR_CLWL_MOVE
Called from MCA when the channel state has changed.

MQXR_CLWL_REPOS
Called from MQPUT or MQPUT1 processing for a repository-manager PCF message.

MQXR_CLWL_REPOS_MOVE
Called from MCA for a repository-manager PCF message when the channel state has changed.

This is an input field to the exit.

ExitResponse (MQLONG)
Response from exit.

This is set by the exit to indicate whether processing of the message should continue. It must be one of the following:

MQXCC_OK
Continue normally.

This indicates that processing of the message should continue normally. *DestinationChosen* identifies the destination to which the message should be sent.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

This indicates that processing of the message should be discontinued:

- For MQXR_CLWL_OPEN, MQXR_CLWL_PUT, and MQXR_CLWL_REPOS invocations, the MQOPEN, MQPUT, or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_STOPPED_BY_CLUSTER_EXIT.
- For MQXR_CLWL_MOVE and MQXR_CLWL_REPOS_MOVE invocations, the message is placed on the dead-letter queue.

MQXCC_SUPPRESS_EXIT
Suppress exit.

This indicates that processing of the current message should continue normally, but that the exit should not be invoked again until termination of the queue manager. The queue manager processes subsequent messages as if the *ClusterWorkloadExit*

queue-manager attribute were blank. *DestinationChosen* identifies the destination to which the current message should be sent.

If any other value is returned by the exit, the queue manager processes the message as if MQXCC_SUPPRESS_FUNCTION had been specified.

This is an output field from the exit.

ExitResponse2 (MQLONG)
Secondary response from exit.

This is set to zero on entry to the exit. It can be set by the exit to provide further information to the queue manager.

When *ExitReason* has the value MQXR_INIT, the exit can set one of the following values in *ExitResponse2*:

MQXR2_STATIC_CACHE
Exit requires a static cluster cache.

If the exit returns this value, the exit need not use the MQXCLWLN call to navigate the chains of records in the cluster cache, but the cache must be static.

If the exit returns this value and the cluster cache is dynamic, the exit cannot navigate correctly through the records in the cache. In this situation, the queue manager processes the return from the MQXR_INIT call as though the exit had returned MQXCC_SUPPRESS_EXIT in the *ExitResponse* field.

MQXR2_DYNAMIC_CACHE
Exit can operate with either a static cache or a dynamic cache.

If the exit returns this value, the exit *must* use the MQXCLWLN call to navigate the chains of records in the cluster cache. Failure to do this is likely to result in the exit terminating abnormally.

If the exit does not set this field, or sets a value which is neither of the above, MQXR2_STATIC_CACHE is assumed.

This is an input/output field to the exit.

Feedback (MQLONG)
Reserved.

This is a reserved field. The value is zero.

Reserved (MQLONG)
Reserved.

This is a reserved field. The value is zero.

ExitUserArea (MQBYTE16)
Exit user area.

This is a field that is available for the exit to use. It is initialized to MQXUA_NONE (binary zero) before the first invocation of the exit, and thereafter any changes made to this field by the exit are preserved across the invocations of the exit that occur between the MQCONN call and the matching MQDISC call. The field is reset to MQXUA_NONE when the MQDISC call occurs. The first invocation of the exit is indicated by the *ExitReason* field having the value MQXR_INIT.

The following value is defined:

MQXUA_NONE
No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

ExitData (MQCHAR32)
Exit data.

This is set on input to the exit routine to the value of the *ClusterWorkloadData* queue-manager attribute. If no value has been defined for that attribute, this field is all blanks.

The length of this field is given by MQ_EXIT_DATA_LENGTH. This is an input field to the exit.

MsgDescPtr (PMQMD)
Address of message descriptor.

This is the address of a copy of the message descriptor (MQMD) for the message being processed. Any changes made to the message descriptor by the exit are ignored by the queue manager.

No message descriptor is passed to the exit if *ExitReason* has one of the following values:
MQXR_INIT
MQXR_TERM
MQXR_CLWL_OPEN

In these cases, *MsgDescPtr* is the null pointer.

This is an input field to the exit.

MsgBufferPtr (PMQVOID)
Address of buffer containing some or all of the message data.

This is the address of a buffer containing a copy of the first *MsgBufferLength* bytes of the message data. Any changes made to the message data by the exit are ignored by the queue manager.

No message data is passed to the exit in the following cases:
- When *MsgDescPtr* is the null pointer.
- When the message has no data.
- When the *ClusterWorkloadLength* queue-manager attribute is zero.

In these cases, *MsgBufferPtr* is the null pointer.

This is an input field to the exit.

MsgBufferLength (MQLONG)
Length of buffer containing message data.

This is the length of the message data passed to the exit. This length is controlled by the *ClusterWorkloadLength* queue-manager attribute, and may be less than the length of the complete message (see *MsgLength*).

This is an input field to the exit.

MsgLength (MQLONG)
Length of complete message.

Be aware that the length of the message data passed to the exit (*MsgBufferLength*) may be less than the length of the complete message. *MsgLength* is zero if *ExitReason* is MQXR_INIT, MQXR_TERM, or MQXR_CLWL_OPEN.

This is an input field to the exit.

QName (MQCHAR48)
Queue name.

This is the name of the destination queue; this queue is a cluster queue.

The length of this field is given by MQ_Q_NAME_LENGTH. This is an input field to the exit.

QMgrName (MQCHAR48)
Name of local queue manager.

This is the name of the queue manager that has invoked the cluster workload exit.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. This is an input field to the exit.

DestinationCount (MQLONG)
Number of possible destinations.

This specifies the number of destination records (MQWDR) that describe instances of the destination queue. There is one MQWDR structure for each possible route to each instance of the queue. The MQWDR structures are addressed by an array of pointers (see *DestinationArrayPtr*).

This is an input field to the exit.

DestinationChosen (MQLONG)
Destination chosen.

This is the number of the MQWDR structure that identifies the route and queue instance to which the message should be sent. The value is in the range 1 through *DestinationCount*.

On input to the exit, *DestinationChosen* indicates the route and queue instance that the queue manager has selected. The exit can accept this choice, or choose a different route and queue instance. However, the value returned by the exit must be in the range 1 through *DestinationCount*. If any other value is returned, the queue manager uses the value that *DestinationChosen* had on input to the exit.

This is an input/output field to the exit.

DestinationArrayPtr (PPMQWDR)
Address of an array of pointers to destination records.

This is the address of an array of pointers to destination records (MQWDR). There are *DestinationCount* destination records.

This is an input field to the exit.

QArrayPtr (PPMQWQR)
Address of an array of pointers to queue records.

This is the address of an array of pointers to queue records (MQWQR). If queue records are available, there are *DestinationCount* of them. If no queue records are available, *QArrayPtr* is the null pointer.

Note: *QArrayPtr* can be the null pointer even when *DestinationCount* is greater than zero.

This is an input field to the exit.

The following fields in this structure are not present if *Version* is less than MQWXP_VERSION_2.

Context (MQPTR)
Context information.

This field is reserved for use by the queue manager. The exit must not alter the value of this field.

This is an input field to the exit. This field is not present if *Version* is less than MQWXP_VERSION_2.

CacheType (MQLONG)
Type of cluster cache.

This is the type of the cluster cache. It is one of the following:

MQCLCT_STATIC
Static cluster cache.

If the cluster cache has this type, the size of the cache is fixed, and cannot grow as the queue manager operates. The MQXCLWLN call need not be used to navigate the records in this type of cache.

MQCLCT_DYNAMIC
Dynamic cluster cache.

If the cluster cache has this type, the size of the cache is fixed, and cannot grow as the queue manager operates. The MQXCLWLN call need not be used to navigate the records in this type of cache.

This is an input field to the exit. This field is not present if *Version* is less than MQWXP_VERSION_2.

C declaration:

```
typedef struct tagMQWXP {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    ExitId;               /* Type of exit */
  MQLONG    ExitReason;           /* Reason for invoking exit */
  MQLONG    ExitResponse;         /* Response from exit */
  MQLONG    ExitResponse2;        /* Secondary response from exit */
  MQLONG    Feedback;             /* Reserved */
  MQLONG    Reserved;             /* Reserved */
  MQBYTE16  ExitUserArea;         /* Exit user area */
  MQCHAR32  ExitData;             /* Exit data */
  PMQMD     MsgDescPtr;           /* Address of message descriptor */
  PMQVOID   MsgBufferPtr;         /* Address of buffer containing some
                                     or all of the message data */
  MQLONG    MsgBufferLength;      /* Length of buffer containing message
                                     data */
  MQLONG    MsgLength;            /* Length of complete message */
  MQCHAR48  QName;                /* Queue name */
  MQCHAR48  QMgrName;             /* Name of local queue manager */
  MQLONG    DestinationCount;     /* Number of possible destinations */
  MQLONG    DestinationChosen;    /* Destination chosen */
  PPMQWDR   DestinationArrayPtr;  /* Address of an array of pointers to
                                     destination records */
  PPMQWQR   QArrayPtr;            /* Address of an array of pointers to
                                     queue records */
  MQPTR     Context;              /* Context information */
  MQLONG    CacheType;            /* Type of cluster cache */
} MQWXP;
```

System/390® assembler declaration:

```
MQWXP                           DSECT
MQWXP_STRUCID                   DS   CL4      Structure identifier
MQWXP_VERSION                   DS   F        Structure version number
MQWXP_EXITID                    DS   F        Type of exit
MQWXP_EXITREASON                DS   F        Reason for invoking exit
MQWXP_EXITRESPONSE              DS   F        Response from exit
MQWXP_EXITRESPONSE2             DS   F        Secondary response from exit
MQWXP_FEEDBACK                  DS   F        Reserved
MQWXP_RESERVED                  DS   F        Reserved
MQWXP_EXITUSERAREA              DS   XL16     Exit user area
MQWXP_EXITDATA                  DS   CL32     Exit data
MQWXP_MSGDESCPTR                DS   F        Address of message
*                                             descriptor
MQWXP_MSGBUFFERPTR              DS   F        Address of buffer containing
*                                             some or all of the message
*                                             data
MQWXP_MSGBUFFERLENGTH           DS   F        Length of buffer containing
*                                             message data
MQWXP_MSGLENGTH                 DS   F        Length of complete message
MQWXP_QNAME                     DS   CL48     Queue name
MQWXP_QMGRNAME                  DS   CL48     Name of local queue manager
MQWXP_DESTINATIONCOUNT          DS   F        Number of possible
*                                             destinations
MQWXP_DESTINATIONCHOSEN         DS   F        Destination chosen
MQWXP_DESTINATIONARRAYPTR       DS   F        Address of an array of
*                                             pointers to destination
*                                             records
MQWXP_QARRAYPTR                 DS   F        Address of an array of
*                                             pointers to queue records
MQWXP_LENGTH                    EQU  *-MQWXP  Length of structure
                                ORG  MQWXP
MQWXP_AREA                      DS   CL(MQWXP_LENGTH)
```

MQWDR - Cluster workload destination-record structure

The definition of the following fields in the MQWDR structure is changed:

ClusterRecOffset (MQLONG)
Offset of first cluster record.

This is the offset of the first MQWCR structure that belongs to this MQWDR structure.

If the cluster cache is static, *ClusterRecOffset* contains the physical offset of the next record; the offset is measured in bytes from the start of the MQWDR structure.

If the cluster cache is dynamic, *ClusterRecOffset* contains the *logical* offset of the next record; the logical offset *cannot* be used in pointer arithmetic. To obtain the address of the next record, the MQXCLWLN call must be used.

It is recommended that MQXCLWLN be used to navigate the records in the cluster cache, as the call returns the address of the next record regardless of type of cache in use.

This is an input field to the exit.

ChannelDefOffset (MQLONG)
Offset of channel definition structure.

This is the offset of the channel definition (MQCD) for the channel that links the local queue manager to the queue manager identified by this MQWDR structure.

If the cluster cache is static, *ChannelDefOffset* contains the physical offset of the channel definition structure; the offset is measured in bytes from the start of the MQWDR structure.

If the cluster cache is dynamic, *ChannelDefOffset* contains the *logical* offset of the channel definition structure; the logical offset *cannot* be used in pointer arithmetic. To obtain the address of the channel definition structure, the MQXCLWLN call must be used.

It is recommended that MQXCLWLN be used to navigate the records and structures in the cluster cache, as the call returns the address of the next record or structure regardless of type of cache in use.

This is an input field to the exit.

MQWQR - Cluster workload queue-record structure

The definition of the following field in the MQWQR structure is changed:

ClusterRecOffset (MQLONG)
Offset of first cluster record.

This is the offset of the first MQWCR structure that belongs to this MQWQR structure.

If the cluster cache is static, *ClusterRecOffset* contains the physical offset of the next record; the offset is measured in bytes from the start of the MQWQR structure.

If the cluster cache is dynamic, *ClusterRecOffset* contains the *logical* offset of the next record; the logical offset *cannot* be used in pointer arithmetic. To obtain the address of the next record, the MQXCLWLN call must be used.

It is recommended that MQXCLWLN be used to navigate the records in the cluster cache, as the call returns the address of the next record regardless of type of cache in use.

This is an input field to the exit.

MQWCR - Cluster workload cluster-record structure

The definition of the following field in the MQWCR structure is changed:

ClusterRecOffset (MQLONG)
Offset of next cluster record.

This is the offset of the next MQWCR structure. If there are no more MQWCR structures, *ClusterRecOffset* is zero.

If the cluster cache is static, *ClusterRecOffset* contains the physical offset of the next record; the offset is measured in bytes from the start of the MQWCR structure.

If the cluster cache is dynamic, *ClusterRecOffset* contains the *logical* offset of the next record; the logical offset *cannot* be used in pointer arithmetic. To obtain the address of the next record, the MQXCLWLN call must be used.

It is recommended that MQXCLWLN be used to navigate the records in the cluster cache, as the call returns the address of the next record regardless of type of cache in use.

This is an input field to the exit.

The MQSeries library

The title of the *MQSeries Command Reference* book, SC33–1369, has been changed to *MQSeries MQSC Command Reference* to reflect more accurately the book's contents. The order number remains the same.

The *Application Programming Reference Summary*, SX33–6095, has become the *MQSeries Programming Interfaces Reference Summary*, and its scope has been expanded to incorporate all MQSeries programming interfaces, including the Message Queuing Interface (MQI), the Application Messaging Interface (AMI), the administration interface (MQAI), event messages, PCF messages, and installable services. The order number remains the same.

Chapter 2. New function in MQSeries for Windows NT and Windows 2000 V5.2 only

This chapter introduces the new function that applies only to MQSeries for Windows NT and Windows 2000 V5.2. It contains these sections:

- "Microsoft® Transaction Server (MTS) support"
- "Installing MQSeries for Windows NT and Windows 2000"
- "Custom services" on page 42
- "Browsing the dead-letter header" on page 45
- "Guidelines for Windows 2000" on page 45

Note that, in the latest editions of the MQSeries cross-product books, all references to MQSeries for Windows NT apply also to MQSeries running in the Windows 2000 environment.

Note also that the application type MQAT_WINDOWS_NT applies equally to the Windows 2000 environment, and that the APPLTYPE value WINDOWSNT, used with the MQSC DEFINE PROCESS and ALTER PROCESS commands, applies to MQSeries in the Windows 2000 environment.

Microsoft® Transaction Server (MTS) support

MQSeries for Windows NT and Windows 2000 V5.2 includes support for Microsoft Transaction Server (MTS). Refer to the online information provided with the MQSeries product for a complete description of this support.

Installing MQSeries for Windows NT and Windows 2000

This section tells you about enhancements to the installation process for MQSeries for Windows NT and Windows 2000.

Launching the Default Configuration

You can launch the Default Configuration Wizard at the end of the installation process. Read the *MQSeries for Windows NT and Windows 2000 Quick Beginnings* book for a complete description of the installation process.

Default Configuration for DHCP machines

You can set up the default configuration when your machine is using the Dynamic Host Configuration Protocol (DHCP). With previous versions of MQSeries, only machines with static IP addresses could set up the default configuration. Refer to the *MQSeries for Windows NT and Windows 2000 Quick Beginnings* book for more information.

Postcard application enhancements

The Postcard application can be used to verify communication between your machine and:

- A queue manager in the default configuration cluster
- A queue manager in a cluster other than the default configuration cluster
- Queue managers that you have defined and added to the default configuration cluster

With Version 5.1 of the Postcard application, you could communicate only with machines that had installed a default queue manager in the default configuration cluster.

Refer to the *MQSeries for Windows NT and Windows 2000 Quick Beginnings* book for more information about using the Postcard application.

Custom services

The MQSeries Services snap-in for MQSeries for Windows NT and Windows 2000 V5.2 has an additional folder, *Custom Services*. You can find instances of this folder in the IBM MQSeries Services folder and in the folder for each queue manager. Custom services are services that you want to be started by the IBM MQSeries Service to coordinate with the starting of queue managers. See "MQSeries Services - Custom Services" in the online *MQSeries Information Center* for information about working with custom services.

Refer to the *MQSeries System Administration* book for a description of administration using the MQSeries Services snap-in.

The **amqmdain** command is added to the MQSeries control commands described in the *MQSeries System Administration* book. **Note that the amqmdain command can be issued only in a Windows NT or Windows 2000 environment.**

Use **amqmdain** to configure or control MQSeries Services, as an alternative to using the MQSeries Services snap-in. Note that starting a queue manager service with **amqmdain** is not the same as using **strmqm** from the command line, because MQSeries Services execute in a non-interactive session, running under a different user account. You can also configure a queue manager service to start associated processes, such as listeners and trigger monitors.

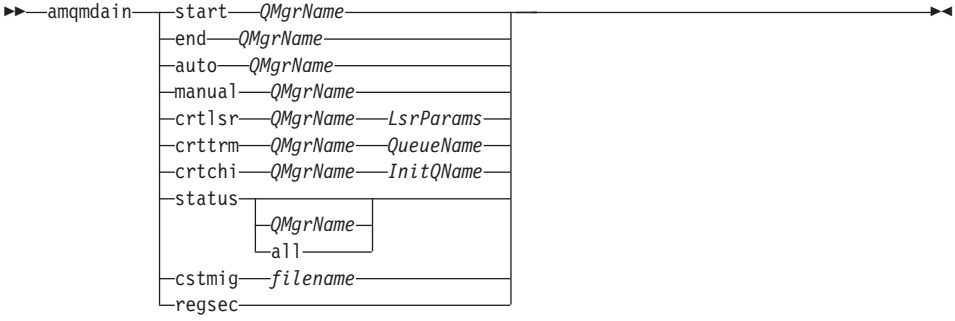
New command: amqmdain (MQSeries services control)

The **amqmdain** command is defined as follows:

Purpose

Use the **amqmdain** command to query or configure MQSeries queue manager services from the command line. You can also use **amqmdain** to ensure that any registry entries you have edited manually are assigned the correct security permissions.

Syntax



Keyword and parameter descriptions

Parameters are required unless the description states they are optional.

In every case, *QMgrName* is the name of the MQSeries queue manager to which the command applies.

start *QMgrName*
Starts a queue manager service.

end *QMgrName*
Ends a queue manager service.

auto *QMgrName*
Sets a queue manager service to automatic startup.

manual *QMgrName*
Sets a queue manager service to manual startup.

crtlsr *QMgrName LsrParams*
Creates a listener service for a queue manager.

LsrParams
Parameters applicable to the **runmqlsr** command, for example `-t trptype -p Port`. The parameters must be in pairs, but the `-m` *QMgrName* parameter is not required, because it is specified by the preceding parameter to the **crtlsr** keyword.

Refer to the *MQSeries System Administration* book for a complete description of the **runmqlsr** command.

crttrm *QMgrName QueueName*
Creates a trigger monitor service for a queue manager.

QueueName
The name of the MQSeries queue to be used by the trigger monitor service. See the **runmqtrm** command in the *MQSeries System Administration* book for more information.

crtchi *QMgrName InitQName*
Creates a channel initiator service for a queue manager.

InitQName
Name of the MQSeries initiation queue to be used by the channel initiator. See the **runmqchi** command in the *MQSeries System Administration* book for more information.

status *QMgrName* | all
The parameter for **status** is optional.

If no parameter is supplied:
Displays the status of the MQSeries service.

If a *QMgrName* is supplied:
Displays the status of the named queue manager service.

If the parameter all is supplied:
Displays the status of all services.

cstmig *filename*
Imports definitions of custom services.

amqmdain loads custom services from a comma–separated value (CSV) configuration file. Note that **amqmdain** *must* be executed to store the custom service parameters in the registry, add the key and values in the correct place and to assign the appropriate security permissions to the registry.

The format of an entry in the configuration file is:

Command Name, Start Command, End Command, Flags, Reserved

For example:

```
PubSub Broker, strmqbrk -p blue.queue.manager, endmqbrk -i
  -m blue.queue.manager, SUFFIX|ROOT|STARTUP|SHUTDOWN|COMMAND, 1
```

regsec Ensures that the security permissions assigned to the MQSeries registry keys are correct.

Return codes

0	Command completed normally
–3	Failed to initialize COM library
–4	Failed to initialize MQSeries COM components
–5	Failed to create channel initiator
–6	Failed to create service
–7	Failed to configure service
–8	Invalid Port Type specified for crtlsr

Browsing the dead-letter header

You can browse the header information for messages on the dead-letter queue with the MQSeries Explorer interface. The information is presented on the MQDLH page of the *Message* Property Sheet. For a description of the header information, and of the conditions under which the MQDLH page is displayed, see the online help for the *Message* Property Sheet.

Guidelines for Windows 2000

This information applies only when you are running MQSeries V5.2 in a Windows 2000 environment.

MQSeries for Windows NT and Windows 2000 V5.2 runs on either platform but the operation of MQSeries security can be affected by differences between Windows 2000 and Windows NT.

MQSeries security relies on calls to the operating system API for information about user authorizations and group memberships. Some functions do not behave identically on Windows 2000 and on Windows NT. This section includes descriptions of how those differences might affect MQSeries security when you are running MQSeries V5.2 in a Windows 2000 environment.

For further information about MQSeries security, refer to the *MQSeries System Administration* book.

When you get a “group not found” error

This problem can arise because MQSeries loses access to the local mqm group when Windows 2000 servers are promoted to, or demoted from, domain controllers. The symptom is an error indicating the lack of a local mqm group, for example:

```
> crtmqm qm0
AMQ8066: Local mqm group not found.
```

You should be aware that altering the state of a machine between server and domain controller can affect the operation of MQSeries. This is because MQSeries makes use of a locally defined mqm group. When a server is

promoted to be a domain controller, the scope changes from local to domain local. When the machine is demoted to server, all domain local groups are removed. This means that changing a machine from server to domain controller, and back to server causes access to a local mqm group to be lost.

To remedy this problem, recreate the local mqm group using the standard Windows 2000 management tools. Because all group membership information is lost, you must reinstate privileged MQSeries users in the newly created local mqm group. If the machine is a domain member you must also add the domain mqm group to the local mqm group to grant privileged domain MQSeries user IDs the required level of authority.

When you have problems with MQSeries and domain controllers

Note: This section describes the problems that can arise with security settings when Windows 2000 servers are promoted to domain controllers. This section applies also when creating the default configuration fails under Windows 2000.

During the promotion of Windows 2000 servers to domain controllers, you are presented with the option to select a default or nondefault security setting relating to user and group permissions. This option controls whether arbitrary users are able to retrieve group memberships from the active directory. Because MQSeries relies on group membership information to implement its security policy, it is important that the user ID that is performing MQSeries operations is able to determine the group memberships of other users.

When a domain is created using the default security option, it is possible for the default user ID created by MQSeries during the installation process (MUSR_MQADMIN) to obtain group memberships for other users as required. The product then installs normally, including the creation of default objects, and the queue manager is able to determine the access authority of local and domain users if required.

When a domain is created using the nondefault security option, the user ID created by MQSeries during the installation (MUSR_MQADMIN) is not always able to determine the required group memberships. In this case, the following information applies:

Windows 2000 domain with nondefault security permissions

When the MQSeries installation is performed by a local user, the local user (MUSR_MQADMIN) created is able to retrieve the group membership information of the installing user, and so installation can complete normally, including the creation of default objects. However, note that the queue manager will be unable to determine access authority of domain users (if required).

When the MQSeries installation is performed by a domain user, the local user (MUSR_MQADMIN) created is unable to retrieve the group membership information of the installing user. This prevents the creation of default objects, and the resulting installation is incomplete. Additional steps are then required to complete the installation.

You can use the Active Directory *Delegation of Control* Wizard to grant permission to a nominated group to the property **Read group membership**. Specifically, it can be used to allow Domain mqm group members permission to read group membership information of an arbitrary user.

To allow MQSeries to retrieve group membership information for an arbitrary domain user, MQSeries must be configured to run under a suitably authorized domain user. This is because local users on servers in a domain cannot be given rights to access objects in Active Directory on the domain controller. The domain user configured for MQSeries should belong to the Domain mqm group.

In summary, for MQSeries on a Windows 2000 domain with the domain controller created using the nondefault user and group permission setting:

- Active Directory at the domain controller must be configured to allow group memberships to be read.
- MQSeries Services (COM server) must be configured to run under a domain user.

Allowing Domain mqm group members to read group membership

This section tells you how to use the Active Directory Wizard to allow Domain mqm group members to read group membership information for an arbitrary user.

In *Active Directory Users and Computers*, right–click the domain name, for example `mqdev.hursley.ibm.com`:

1. Click **Delegate Control**, then click **Next**.
2. Click **Groups and Users**:
   a. Click **Add**.
   b. Highlight Domain mqm and click **Add**.
3. Click **OK**.
4. Highlight the Domain mqm selection and click **Next**.
5. Select the **Create a custom task to delegate** check box and click **Next**.
6. Check **Only the following objects in the folder** and then search under object types for **User objects** (the list is alphabetical, so go to the last one).
7. Check **User Objects** and click **Next**.
8. Check **Property-specific** and then search down to:

```
Read Group Membership
Read groupMembershipSAM
```

(the list is sorted alphabetically on the second word). Select both of these check boxes, then click **Next**.

9. Click **Finish**.

Configuring MQSeries Services to run under a domain user

You can change the user account under which MQSeries Services runs to be other than the default MUSR_MQADMIN. To do this, first create the new domain user account that you wish to use. Then, on each MQSeries server running MQSeries V5.2, use the following command to configure the MQSeries Services to run under the user ID you require, and also to allocate the correct security rights and group memberships to this user account:

```
AMQMSRVN -regserver -user [domain]\[userid] - password [password]
```

Note that checking **Password never expires** when you create a user ID for use by MQSeries Services will prevent the need to reconfigure the services owing to password expiry.

When MQSeries appears to halt when reporting an error

Windows 2000 can inherit permissions differently from Windows NT. This might cause MQSeries to lose write access to queue manager error logs, depending on the access rights that have been allocated to the parent directory into which MQSeries is installed. To remedy this problem, ensure that all appropriate users of the product have write access to the queue manager error directories.

When Default Configuration gives errors

For default configuration to work correctly, set up the computer name and DNS domain in the following manner:

1. Right–click the **My Computer** icon.
2. Click **Properties**.
3. Click the **Network Identification** tab on the System Properties panel.
4. Click **Properties** and type the computer name.
5. Click **More** and type the DNS domain name.
6. Click **OK** to retain the updates.

When you have problems with a Multilanguage system

This section tells you what to do if you get:

- Some English messages when using MQSeries on a Windows 2000 Multilanguage system
- ?????? on the installation panels when installing MQSeries on a Windows 2000 DBCS Multilanguage system

To install and use MQSeries on a Windows 2000 Multilanguage system, set up the local and system locale values in the following manner:

1. Select the Control Panel.
2. Click **Regional Options**.
3. On the Regional Options panel, update both **Your locale** and the setting for the required language.
4. Click **Set default** and set the **System locale** to the required language.
5. Click **Apply**.
6. Reboot the system.

MQSeries should now install and run correctly.

Applying security template files

Windows 2000 supports text-based security template files which you can use to apply uniform security settings to one or more computers with the Security Configuration and Analysis MMC snap-in. In particular, Windows 2000 supplies several templates that include a range of security settings with the aim of providing specific levels of security; these include compatible, basic, secure, and highly-secure.

Be aware that applying one of these templates might affect the security settings applied to MQSeries files and directories. If you want to use the highly-secure template, configure your machine *before* you install MQSeries. If you apply the highly-secure template to a machine on which MQSeries is already installed, all the permissions you have specifically set on the MQSeries files and directories are removed. This means that you lose Administrator and mqm group access and, when applicable, Everyone group access from the error directories.

Note

Refer to the MQSeries Web site for the latest Frequently Asked Questions (FAQs).

Chapter 3. New function in V5.2 UNIX® systems only

This chapter introduces the new function that applies to the following products:

- MQSeries for AIX, V5.2
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for Sun Solaris, V5.2

The general discussion that introduces the section "UNIX signal handling on MQSeries V5.2 products" on page 52 also applies to the MQSeries for AS/400, V5.2 product.

For more information on the functions that apply to your product, refer to the sections:

- "New function for UNIX systems"
- "UNIX signal handling on MQSeries V5.2 products" on page 52
- "New function for Sun Solaris only" on page 57
- "New function for AIX only" on page 78

New function for UNIX systems

This section describes the changes in MQSeries V5.2 that apply to more than one of the UNIX products. "Threaded applications" applies to all the UNIX products. "Support for Websphere as an XA coordinator" on page 52 applies to the AIX, HP-UX, and Sun Solaris products only.

The section "UNIX signal handling on MQSeries V5.2 products" on page 52 applies to all the UNIX products.

Threaded applications

If you are writing multithreaded MQSeries applications that create new processes, note that calls to create a new process using the fork system call must be followed by an exec system call to execute a program. MQSeries V5.2 does not support the use of the fork system call unless it is followed by an exec system call.

You can refer to the *MQSeries Application Programming Guide* for more information about writing multithreaded MQSeries applications. Note, however, that this book does not discuss using the fork system call to create a new process.

All UNIX systems

Support for Websphere as an XA coordinator

MQSeries V5.2 supports Websphere as a transaction coordinator in the following environments: AIX, HP-UX, and Sun Solaris. Refer to the Websphere documentation for more information on connecting Websphere to MQSeries.

For more information about the MQSeries application adaptor, and about how to write Component Broker applications please see the *WebSphere™ Application Server Enterprise Edition Component Broker MQSeries Application Adaptor Development Guide* SC09–4444.

UNIX signal handling on MQSeries V5.2 products

Note

This section is a complete replacement for the section "UNIX Signal handling on MQSeries Version 5 products" in the *MQSeries Application Programming Guide*.

Changes have been made to the following:

- The description of the "libmqm_r library" on page 53 in this book, in particular the behavior for SIGALRM. Furthermore, MQSeries V5.2 does not need to run clean–up code in the event of abnormal termination.
- The handling of a synchronous signal in "Unthreaded applications" on page 54 .
- The section "Threaded applications" on page 54.
- The section "Fastpath (trusted) applications" on page 55.

In general, UNIX and AS/400 systems have moved from a nonthreaded (process) environment to a multithreaded environment. In the nonthreaded environment, some functions could be implemented only by using signals, though most applications did not need to be aware of signals and signal handling. In the multithreaded environment, thread-based primitives support some of the functions that used to be implemented in the nonthreaded environments using signals. In many instances, signals and signal handling, although supported, do not fit well into the multithreaded environment and various restrictions exist. This can be particularly problematic when you are integrating application code with different middleware libraries (running as part of the application) in a multithreaded environment where each is trying to handle signals. The traditional approach of saving and restoring signal handlers (defined per process), which worked when there was only one thread of execution within a process, does not work in a multithreaded

environment: many threads of execution could be trying to save and restore a process-wide resource, with unpredictable results.

For a standard application, MQSeries supports both nonthreaded and threaded application environments on AIX, AS/400, HP-UX, and Linux.

All MQSeries applications in the Sun Solaris environment are threaded. MQSeries for Sun Solaris V2.2 supported only single-threaded applications (though there was no way to enforce this) and, because there was only one thread of execution, was able to make use of the traditional signal handling functions. In MQSeries for Sun Solaris, V5.0, and subsequent releases, true multithreaded applications are supported and so the signal behavior has changed.

The library libmqm is provided for migration of nonthreaded applications from Version 2 of MQSeries for AIX or MQSeries for HP-UX to Version 5. The goal of this library is to maintain the Version 2 behavior (including signals) for nonthreaded applications. Within an application in this environment there is only one thread of execution, which means that signal handlers can be saved and restored safely across MQSeries API calls (as can any middleware library that is part of the application). Therefore, if you have an application suite on V2 of MQSeries for AIX or MQSeries for HP-UX that uses signals, and you do not want to move to the threaded environment, the suite should run unchanged on V5 using the nonthreaded library, libmqm.

The library libmqm_r is provided for threaded applications on MQSeries for AIX, MQSeries for HP-UX, and MQSeries for Linux. On AS/400 libmqm_r is provided as a service program. However, the behavior, particularly for signals, is different:

- As in the nonthreaded environment, MQSeries still establishes signal handlers for synchronous terminating signals (SIGBUS, SIGFPE, SIGSEGV, SIGILL).
- In the threaded environment MQSeries does not need to use the SIGALRM signal as it does in the nonthreaded environment.

  Note: Some system functions may use signals internally (for example, SIGALRM in a nonthreaded environment). For a particular operating system, some of these functions may have thread-safe equivalents or it may be stated that they are not multithread safe. Any non-thread-safe operating system call should be replaced if moving to a multithreaded environment.

UNIX signal handling

Unthreaded applications

Each MQI function sets up its own signal handler for the signals:

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

Users' handlers for these are replaced for the duration of the MQI function call. Other signals can be caught in the normal way by user-written handlers. If you do not install a handler, the default actions (for example, ignore, core dump, or exit) are left in place.

Following the handling of a synchronous signal (SIGSEGV, SIGBUS, SIGFPE, SIGILL) by MQSeries it will attempt to pass the signal on to any signal handler registered prior to making the MQI function call.

Note: On Sun Solaris all applications are threaded even if they use a single thread.

Threaded applications

A thread is considered to be connected to MQSeries from MQCONN (or MQCONNX) until MQDISC.

Synchronous signals

Synchronous signals arise in a specific thread. UNIX safely allows the setting up of a signal handler for such signals for the whole process. However, MQSeries sets up its own handler for the following signals, in the application process, while any thread is connected to MQSeries:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

If you are writing multithreaded applications, you should note that there is only one process-wide signal handler for each signal. When MQSeries sets up its own synchronous signal handlers it saves any previously registered handlers for each signal. Following the handling by MQSeries of one of the signals listed above, MQSeries attempts to call the signal handler that was in effect at the time of the first MQSeries connection within the process. The previously registered handlers are restored when all application threads have disconnected from MQSeries.

Because signal handlers are saved and restored by MQSeries, application threads ideally should not establish signal handlers for these signals while there is any possibility that another thread of the same process is also connected to MQSeries.

Note: When an application, or a middleware library (running as part of an application), does establish a signal handler while a thread is connected to MQSeries, the application's signal handler *must* call the corresponding MQSeries handler during the processing of that signal.

When establishing and restoring signal handlers, the general principle is that the last signal handler to be saved must be the first to be restored:

- When an application establishes a signal handler *after* connecting to MQSeries, the previous signal handler must be restored *before* the application disconnects from MQSeries.
- When an application establishes a signal handler *before* connecting to MQSeries, the application must disconnect from MQSeries *before* restoring its signal handler.

Note: Failure to observe the general principle that the last signal handler to be saved must be the first to be restored can result in unexpected signal handling in the application and, potentially, the loss of signals by the application.

Asynchronous signals

MQSeries does not make use of any asynchronous signals in threaded applications unless they are client applications.

Threaded client applications - additional considerations

MQSeries handles the following signals during I/O to a server. These signals are defined by the communications stack. The application should not establish a signal handler for these signals while a thread is connected to a queue manager:

SIGPIPE
(for TCP/IP)

Fastpath (trusted) applications

Fastpath applications run in the same process as MQSeries and so are running in the multithreaded environment. In this environment MQSeries handles the synchronous signals SIGSEGV, SIGBUS, SIGFPE, and SIGILL. All other signals must not be delivered to the Fastpath application whilst it is connected to MQSeries. Instead they must be blocked or handled by the application. If a Fastpath application intercepts such an event the Queue Manager must be stopped and restarted, or it may be left in an undefined state. For a full list of the restrictions for Fastpath applications under MQCONNX see "Connecting to a queue manager using the MQCONNX call"

UNIX signal handling

MQI function calls within signal handlers

While you are in a signal handler, you cannot call an MQI function. If you call an MQI function, while another MQI function is active, MQRC_CALL_IN_PROGRESS is returned. If you call an MQI function, while no other MQI function is active, it is likely to fail because of the operating system restrictions on which calls can be issued from within a handler.

In the case of C++ destructor methods, which may be called automatically during program exit, you may not be able to stop the MQI functions from being called. Therefore, ignore any errors about MQRC_CALL_IN_PROGRESS. If a signal handler calls exit(), MQSeries backs out uncommitted messages in syncpoint as normal and closes any open queues.

Signals during MQI calls

MQI functions do not return the code EINTR or any equivalent to application programs. If a signal occurs during an MQI call, and the handler calls 'return', the call continues to run as if the signal had not happened. In particular, MQGET cannot be interrupted by a signal to return control immediately to the application. If you want to break out of an MQGET, set the queue to GET_DISABLED; alternatively, use a loop around a call to MQGET with a finite time expiry (MQGMO_WAIT with gmo.WaitInterval set), and use your signal handler (in a nonthreaded environment) or equivalent function in a threaded environment to set a flag which breaks the loop.

In the AIX environment, MQSeries requires that system calls interrupted by signals are restarted. You must establish the signal handler with sigaction(2) and set the SA_RESTART flag in the sa_flags field of the new action structure. The default behavior is that calls are not restarted (the SA_RESTART flag is not set).

User exits and installable services

User exits and installable services that run as part of an MQSeries process in a multithreaded environment have the same restrictions as for Fastpath applications. They should be considered as permanently connected to MQSeries and so not use signals or non-threadsafe operating system calls.

New function for Sun Solaris only

This section introduces the new function that applies only to MQSeries for Sun Solaris, V5.2.

Sun Workshop C++ Compiler 5.0 and 6.0

MQSeries V5.2 supports Version 5.0 and Version 4.2 of the Sun Workshop C++ Compiler and the Forte C++ 6 (Sun WorkShop 6 C++) compiler. Refer to the *MQSeries Using C++* book for information about compiling and linking programs with the compiler you are using. The instructions for the Forte C++ 6 (Sun WorkShop 6 C++) compiler are the same as for Version 5.0 of the Sun Workshop C++ Compiler.

Communications support extended to include SNAP–IX

This section gives an example of how to set up communication links from MQSeries for Sun Solaris using SNAP–IX V6.2 or later. See Chapter 18 in the *MQSeries Intercommunication* book for a description of the existing support for SunLink Version 9.1. Note the following with regard to that Chapter:

- The section “Configuration parameters for an LU 6.2 connection” applies only to SunLink Version 9.1.
- The section “Establishing a connection using SunLink Version 9.1” is unchanged.
- The sections “Configuration parameters for an LU 6.2 connection using SNAP–IX” on page 58 and “Establishing a session using SNAP–IX” on page 63 are added.
- The section “Establishing a TCP connection” is unchanged.
- In the section “MQSeries for Sun Solaris configuration”, the existing “MQSeries for Sun Solaris sender-channel definitions using SNA” applies only to SunLink Version 9.1.
- “MQSeries for Sun Solaris sender-channel definitions using SNAP-IX SNA” on page 77 is added to the section “MQSeries for Sun Solaris configuration”.

You determine the MQSeries library loaded to support SNA with the *MQCommLibrary* parameter in the qm.ini file. With MQSeries for Sun Solaris, V5.2, the `LU62` stanza applies. *MQCommLibrary* is the only attribute that can be specified:

MQCommLibrary=amqcc62a|amqcc62s

This attribute specifies the MQSeries library loaded to support SNA for MQSeries for Sun Solaris, V5.2.

SNAP–IX support is provided if amqcc62a is loaded. This is the default if this attribute is not specified.

SunLink Version 9.1 support is provided if amqcc62s is loaded.

Refer to the *MQSeries System Administration* book for a full description of the LU62 stanza in the qm.ini file.

Configuration parameters for an LU 6.2 connection using SNAP–IX

Table 2 presents a worksheet listing all the parameters needed to set up communication from Sun Solaris using SNAP–IX to one of the other MQSeries platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet in this chapter in conjunction with the worksheet in the chapter of the *MQSeries Intercommunication* book for the platform to which you are connecting.

Configuration worksheet: Use this worksheet to record the values you use for your configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in the *MQSeries Intercommunication* book. The examples that follow in this chapter refer back to the values in the ID column. The entries in the Parameter Name column are explained in "Explanation of terms" on page 61.

Table 2. Configuration worksheet for SNAP–IX

ID	Parameter Name	Ref.	Example	User Value
Parameters for local node				
1	Configuration file name		**sna_node.cfg**	
2	Control point name		**SOLARXPU**	
3	Node ID to send		**05D 23456**	
4	Network name		**NETID**	
5	Local APPC LU		**SOLARXLU**	
6	APPC mode		**#INTER**	
7	Invokable TP		**MQSERIES**	
8	Local MAC address		**08002071CC8A**	
9	Port name		**MQPORT**	
10	Command path		**/opt/mqm/bin/amqcrs6a**	
11	Local queue manager		**SOLARIS**	
Connection to an OS/2® system The values in this section of the table must match those used in the Table for OS/2 and LU6.2, as indicated.				
12	Link station name		**OS2CONN**	
13	Network name	2	**NETID**	
14	CP name	3	**OS2PU**	
15	Remote LU	6	**OS2LU**	
16	Application TP	8	**MQSERIES**	
17	Mode name	17	**#INTER**	

Table 2. Configuration worksheet for SNAP–IX (continued)

ID	Parameter Name	Ref.	Example	User Value
18	CPI-C symbolic destination name		**OS2CPIC**	
19	Remote network address	10	**10005AFC5D83**	
20	Node ID to receive	4	**05D 12345**	
Connection to a Windows NT or Windows 2000 system The values in this section of the table must match those used in the Table for Windows NT and LU6.2, as indicated.				
12	Link station name		**NTCONN**	
13	Network name	2	**NETID**	
14	CP name	3	**WINNTCP**	
15	Remote LU	5	**WINNTLU**	
16	Application TP	7	**MQSERIES**	
17	Mode name	17	**#INTER**	
18	CPI-C symbolic destination name		**NTCPIC**	
19	Remote network address	9	**08005AA5FAB9**	
20	Node ID to receive	4	**05D 30F65**	
Connection to an AIX system The values in this section of the table must match those used in the Table for AIX and LU6.2, as indicated.				
12	Link station name		**AIXCONN**	
13	Network name	1	**NETID**	
14	CP name	2	**AIXPU**	
15	Remote LU	4	**AIXLU**	
16	Application TP	6	**MQSERIES**	
17	Mode name	14	**#INTER**	
18	CPI-C symbolic destination name		**AIXCPIC**	
19	Remote network address	8	**123456789012**	
20	Node ID to receive	3	**071 23456**	
Connection to an AT&T GIS UNIX system The values in this section of the table must match those used in the table the Table for AT&T GIS UNIX and LU6.2, as indicated.				
12	Link station name		**GISCONN**	
13	Network name	2	**NETID**	
14	CP name	3	**GISPU**	
15	Remote LU		**GISLU**	
16	Application TP	5	**MQSERIES**	
17	Mode name	7	**#INTER**	
18	CPI-C symbolic destination name		**GISCPIC**	
19	Remote network address	8	**10007038E86B**	

Table 2. Configuration worksheet for SNAP–IX (continued)

ID	Parameter Name	Ref.	Example	User Value
20	Node ID to receive	9	**03E 00018**	
Connection to an HP—UX system The values in this section of the table must match those used in the Table for HP—UX and LU6.2, as indicated.				
12	Link station name		**HPUXCONN**	
13	Network name	2	**NETID**	
14	CP name	3	**HPUXPU**	
15	Remote LU	7	**HPUXLU**	
16	Application TP	8	**MQSERIES**	
17	Mode name	17	**#INTER**	
18	CPI-C symbolic destination name		**HPUXCPIC**	
19	Remote network address	5	**10005FC5D83**	
20	node ID to receive	6	**05D 54321**	
Connection to an AS/400 system The values in this section of the table must match those used in the Table for AS/400 and LU6.2, as indicated.				
12	Link station name		**AS4CONN**	
13	Network name	1	**NETID**	
14	CP name	2	**AS400PU**	
15	Remote LU	3	**AS400LU**	
16	Application TP	8	**MQSERIES**	
17	Mode name	17	**#INTER**	
18	CPI-C symbolic destination name		**AS4CPIC**	
19	Remote network address	4	**10005A5962EF**	
Connection to an OS/390 or MVS/ESA™ system without CICS® The values in this section of the table must match those used in the Table for OS/390 and LU6.2, as indicated.				
12	Link station name		**MVSCONN**	
13	Network name	2	**NETID**	
14	CP name	3	**MVSPU**	
15	Remote LU	4	**MVSLU**	
16	Application TP	7	**MQSERIES**	
17	Mode name	10	**#INTER**	
18	CPI-C symbolic destination name		**MVSCPIC**	
19	Remote network address	8	**400074511092**	
Connection to a VSE/ESA™ system The values in this section of the table must match those used in the Table for VSE/ESA and LU6.2, as indicated.				
12	Link station name		**VSECONN**	

Table 2. Configuration worksheet for SNAP–IX (continued)

ID	Parameter Name	Ref.	Example	User Value
13	Network name	1	**NETID**	
14	CP name	2	**VSEPU**	
15	Remote LU	3	**VSELU**	
16	Application TP	4	**MQ01**	**MQ01**
17	Mode name		**#INTER**	
18	CPI-C symbolic destination name		**VSECPIC**	
19	Remote network address	5	**400074511092**	

Explanation of terms

1 Configuration file name

This is the unique name of the SNAP–IX configuration file. The default for this name is `sna_node.cfg`.

Although it is possible to edit this file it is strongly recommended that configuration is done using xsnadmin.

2 Control point name

This is the unique Control point name for this workstation. In the SNA network, the Control point is an addressable location (PU type 2.1). Your network administrator will assign this to you.

3 Node ID to send

This is the unique ID of this workstation. On other platforms this is often referred to as the *Exchange ID* or *XID*. Your network administrator will assign this ID for you.

4 Network name

This is the unique ID of the network to which you are connected. It is an alphanumeric value and can be 1-8 characters long. The network name works with the Control point name to uniquely identify a system. Your network administrator will tell you the value.

5 Local APPC LU

An LU manages the exchange of data between transactions. The local APPC LU name is the name of the LU on your system. Your network administrator will assign this to you.

6 APPC mode

This is the name given to the set of parameters that control the APPC conversation. This name must be defined at each partner system. Your network administrator will assign this to you.

7 Invokable TP

MQSeries applications trying to converse with this workstation will specify a symbolic name for the program to be run at the receiving

end. This will have been defined on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See the *MQSeries Intercommunication* book for more information.

8 Local MAC address
This is the network address of the token-ring card. The address to be specified is found in the ether value displayed in response to the `ifconfig tr0` command issued at a root level of authority. (Tr0 is the name of the machine's token-ring interface.) If you do not have the necessary level of authority, your Sun Solaris system administrator can tell you the value.

9 Port name
This is a meaningful symbolic name that is used to associate the definitions with a network interface (in this case, a Token-Ring adapter). A separate Port must be defined for each physical device attached to the workstation.

10 Full path to executable
This is the path and name of the script file that invokes the MQSeries program to run.

11 Local queue manager
This is the name of the queue manager on your local system.

10 Link station name
This is a meaningful symbolic name by which the connection to a peer or host node is known. It defines a logical path to the remote system. Its name is used only inside SNAP–IX and is specified by you. The connection must be associated with an existing Link and owned by one local node. You must define one connection for each partner or host system.

18 CPI-C symbolic destination name
This is a name given to the definition of a partner node. You choose the name. It need be unique only on this machine. Later you can use this name in the MQSeries sender channel definition.

20 Node ID to receive
This is the unique ID of the partner workstation with which you will be communicating. On other platforms this is often referred to as the *Exchange ID* or *XID*. For a connection to a host system any values except `000 FFFFF` and `FFF FFFFF` may be specified. Your network administrator will assign this ID for you.

Establishing a session using SNAP–IX

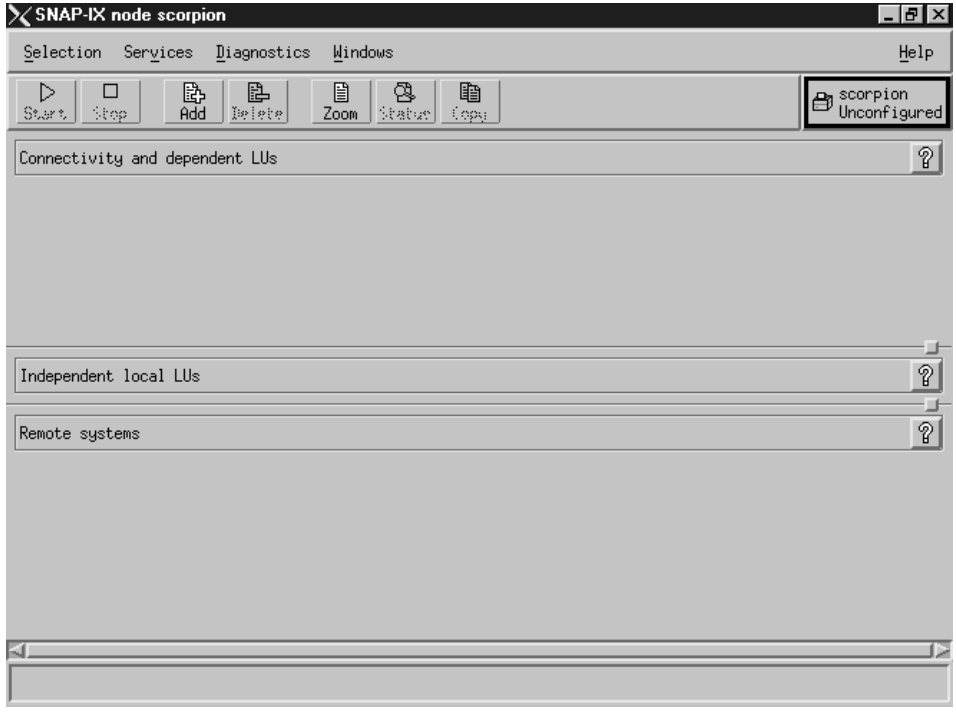
The following information guides you through the tasks you must perform to create the SNA infrastructure that MQSeries requires. This example creates the definitions for a partner node and LU on OS/2.

Use **sna start** followed by **xsnaadmin** to type the SNAP–IX configuration panels. You need root authority to use **xsnaadmin**.

SNAP–IX configuration: SNAP–IX configuration involves the following steps:

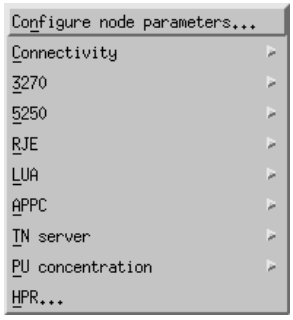
1. Defining a local node
2. Adding a Token Ring Port
3. Defining a local LU

The SNAP–IX main menu, from which you start, is shown here:

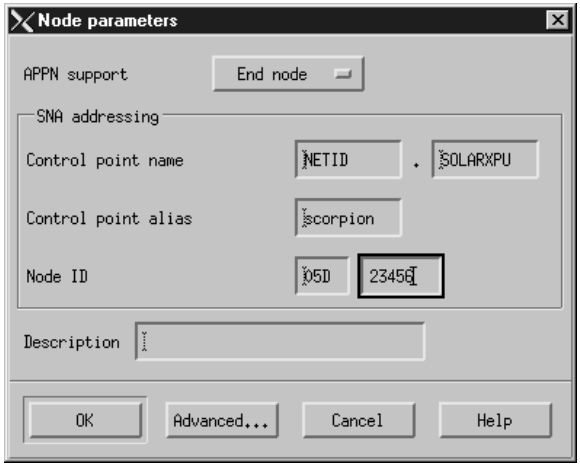


Defining a local node:

1. From the SNAP–IX main menu, click the **Services** pull-down:



2. Click **Configure node parameters**. The following panel is displayed:

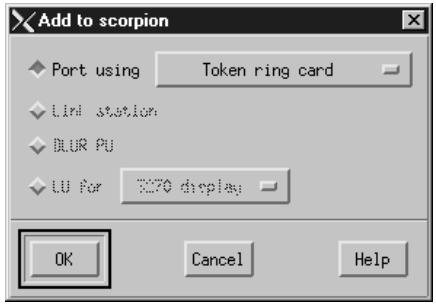


3. Complete the **Control point name** with the values **Network name** (**4**) and **Control point name** (**2**).
4. Type the **Control point name** (**2**) in the **Control point alias** field.
5. Type the **Node ID** (**3**).
6. Click **End node**.
7. Click **OK**.

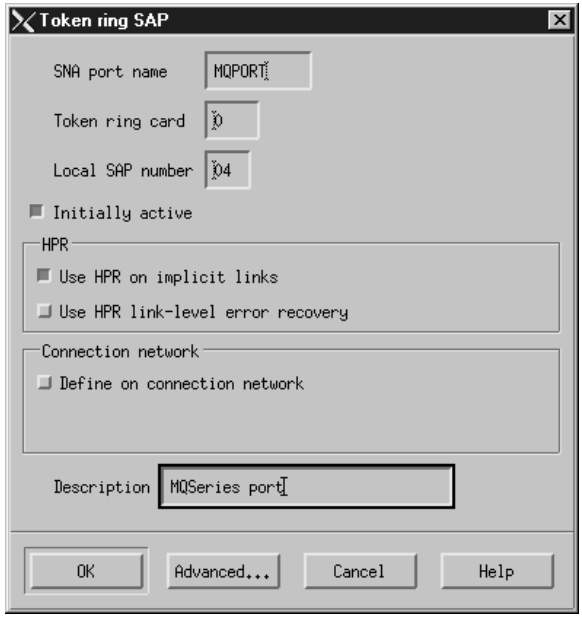
A default independent local LU is defined.

Adding a Token Ring Port:

1. From the main SNAP–IX menu, click **Connectivity and dependent LUs**.
2. Click **Add**. The following panel is displayed:



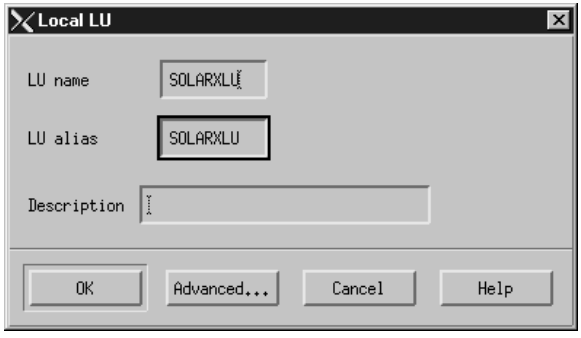
3. Click **Token Ring Card** and click **OK**. The following panel is displayed:



4. Type the **SNA port name** (**9**).
5. Type a **Description** and click **OK** to take the default values.

Defining a local LU:

1. From the main SNAP–IX menu, click **Independent local LUs**.
2. Click **Add**. The following panel is displayed:



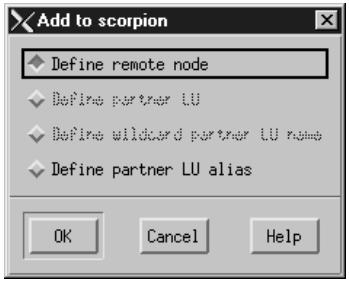
3. Type the **LU name** (**5**) and click **OK**.

APPC configuration: APPC configuration involves the following steps:

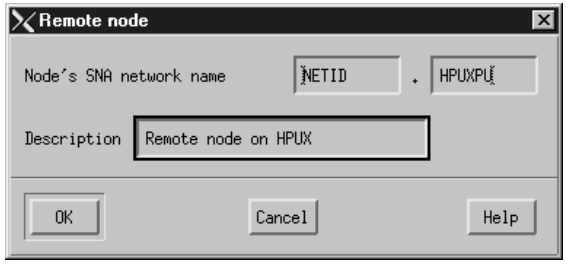
1. Defining a remote node
2. Defining a partner LU
3. Defining a link station
4. Defining a mode
5. Adding CPI-C information
6. Adding a TP definition

Defining a remote node:

1. From the main SNAP–IX menu, click **Remote systems**.
2. Click **Add**. The following panel is displayed:



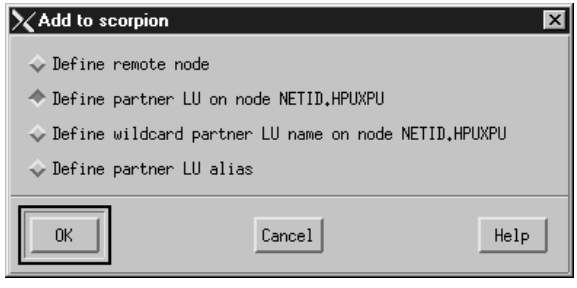
3. Select the **Define remote node** check box and click **OK**. The following panel is displayed:



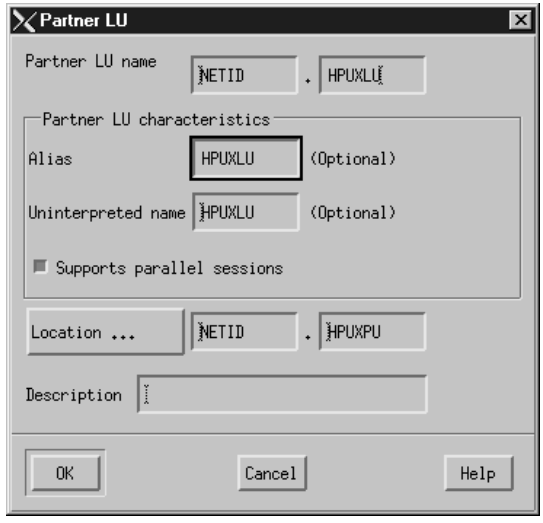
4. Type the **Node's SNA network name** (**13**) and a **Description**.
5. Click **OK**.
6. A default partner LU with the same name is generated and a message is displayed.
7. Click **OK**.

Defining a partner LU:

1. From the main SNAP–IX menu, click **Remote systems** and click the remote node.
2. Click **Add**. The following panel is displayed:



3. Select the**Define partner LU on node node name** check box.
4. Click **OK**. The following panel is displayed:

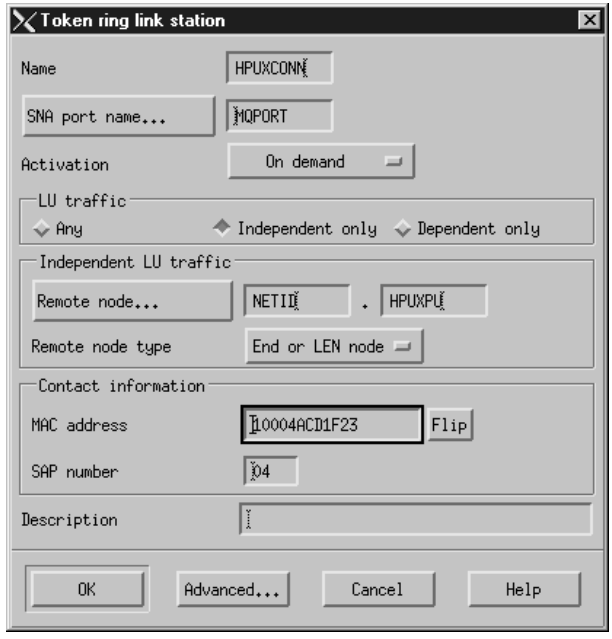


5. Type the **partner LU name** (**15**) and click **OK**.

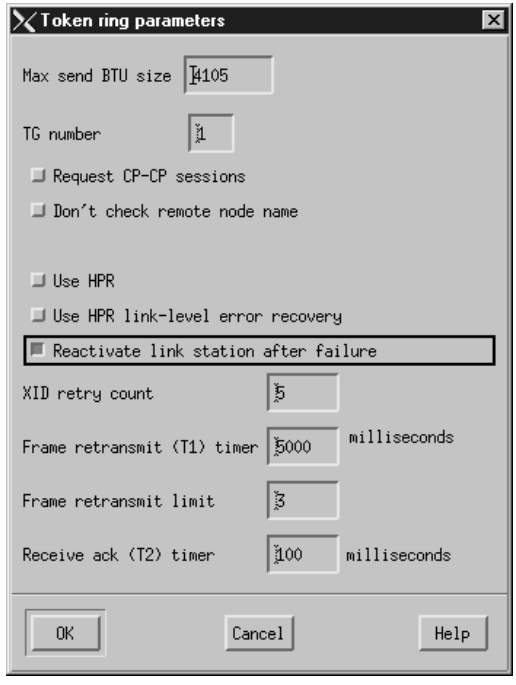
Defining a link station:

1. From the main SNAP–IX menu, click **Connectivity and dependent LUs**.
2. Click the MQPORT port.
3. Click **Add**. The following panel is displayed:

4. Select the **Add link station to port MQPORT** check box.
5. Click **OK**. The following panel is displayed:



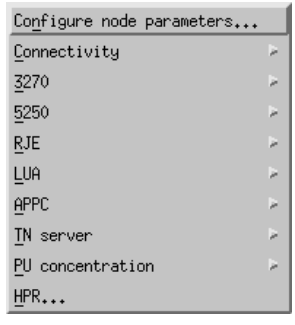
6. Type the **Name** of the link station (**12**).
7. Set the value of **Activation** to "On demand".
8. Select the **Independent only** check box.
9. Click **Remote node** and select the value of the remote node (**14**).
10. Click **OK**.
11. Set the value of **Remote node type** to "End or LEN node".
12. Type the value for **MAC address** (**19**) and click **Advanced**. The following panel is displayed:



13. Select the **Request CP-CP sessions**. check box
14. Select the **Reactivate link station after failure**. check box
15. Click **OK** to exit the Advanced panel.
16. Click **OK** again.

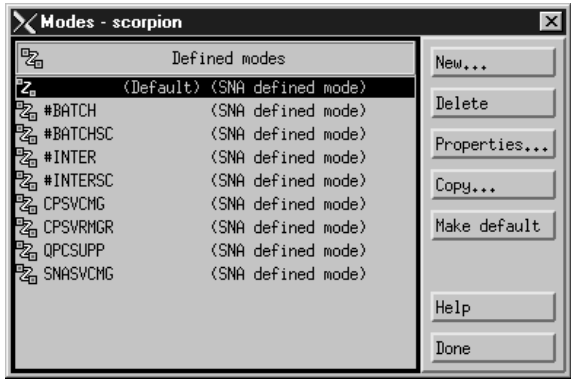
Defining a mode:

1. From the SNAP–IX main menu, click the **Services** pull-down: The following panel is displayed:

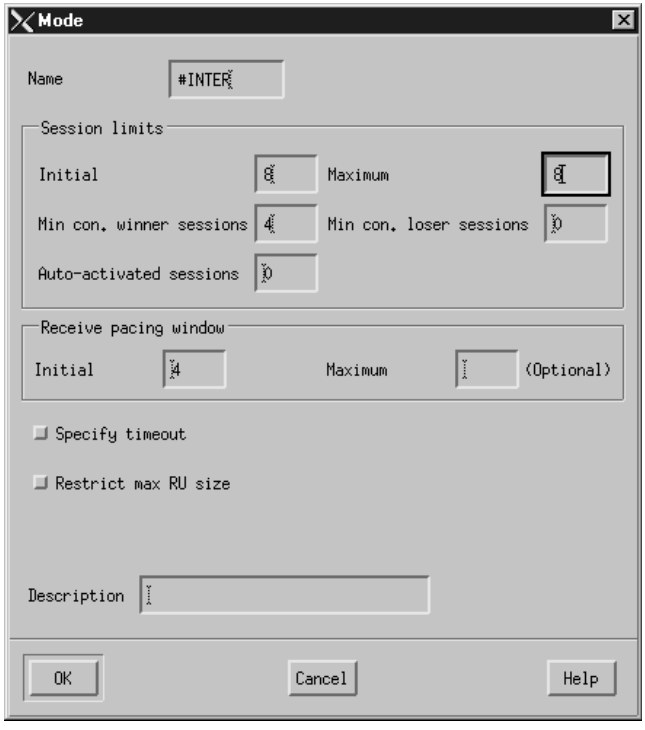


2. Click **APPC**. The following panel is displayed:

3. Click **Modes**. The following panel is displayed:



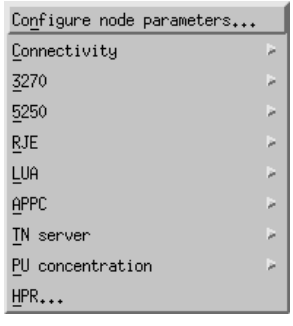
4. Click **Add**. The following panel is displayed:



5. Type the **Name** to be given to the mode (**17**).
6. Set the values of **Initial session limit** to 8, **Min con. winner sessions** to 4, and **Auto-activated sessions** to 0.
7. Click **OK**.
8. Click **Done**.

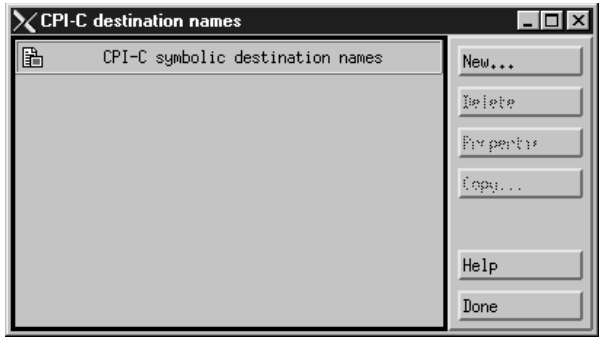
Adding CPI-C information:

1. From the SNAP–IX main menu, click the **Services** pull-down:

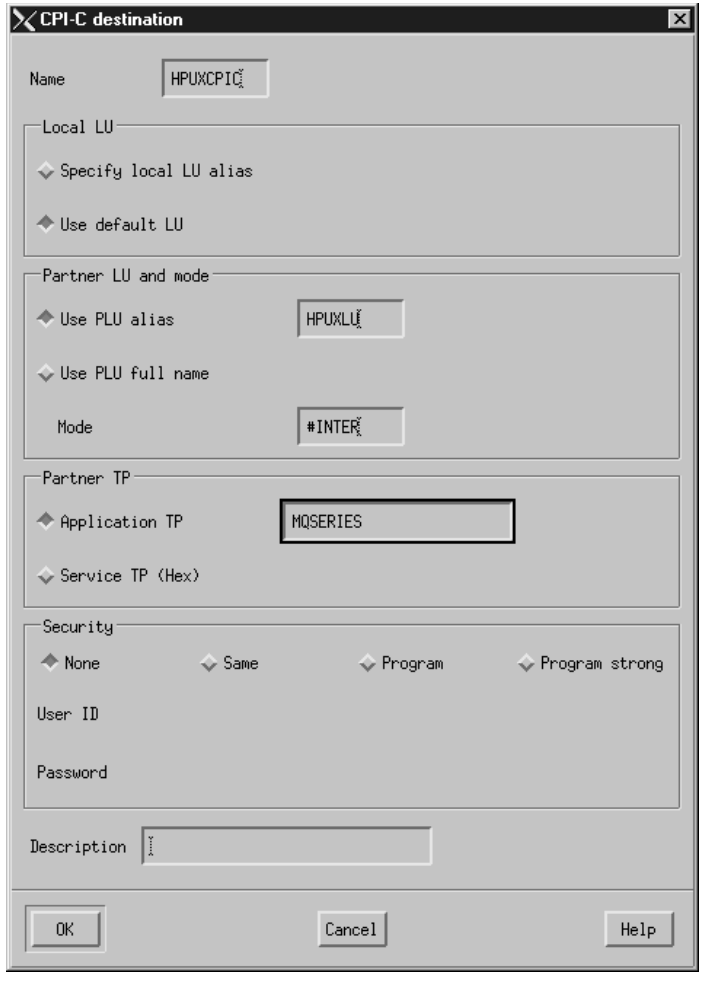


2. Click **APPC**. The following panel is displayed:

3. Click **CPI-C**. The following panel is displayed:



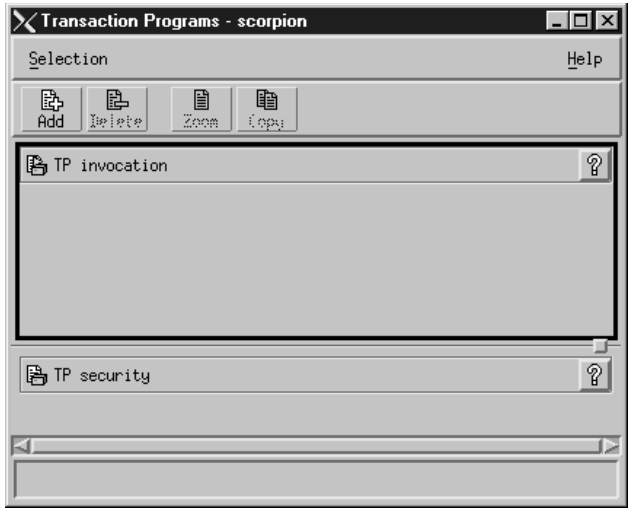
4. Click **Add**. The following panel is displayed:



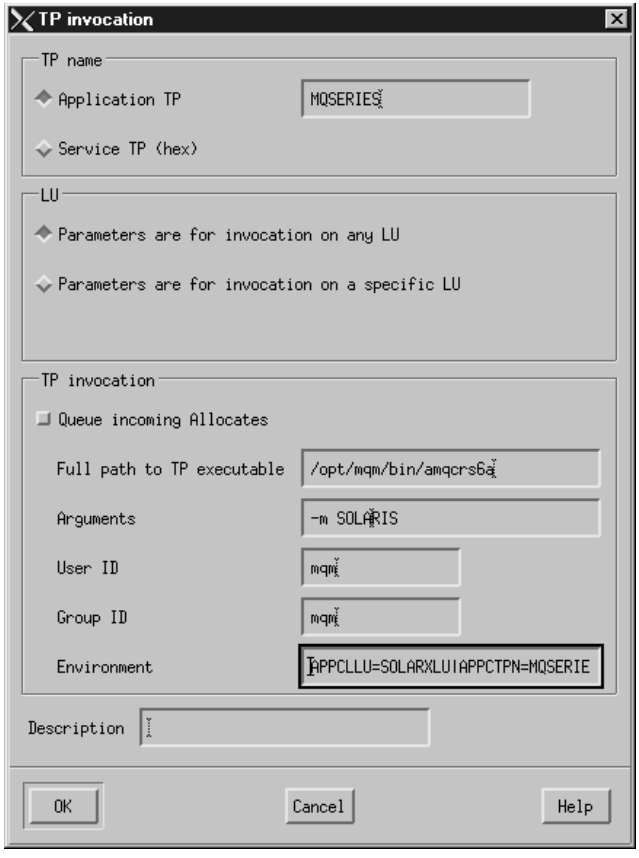
5. Type the **Name** (**18**). Select the **Application TP** check box and type the value (**16**). Select the **Use PLU alias** check box and type the name (**15**). Type the **Mode** name (**17**).
6. Click **OK**.

Adding a TP definition using SNAP–IX Release 6: To add a TP definition:

1. Click the **Services** pull-down and click **APPC** as for CPI-C information.
2. Click **Transaction Programs**. The following panel is displayed:



3. Click **Add**. The following panel is displayed:



4. Type **TP name** (**7**) in the **Application TP** field.
5. Clear the **Queue incoming Allocates** check box.
6. Type **Full path to executable** (**10**).
7. Type **-m Local queue manager** (**11**) in the **Arguments** field.
8. Type **mqm** in the **User ID** and **Group ID** fields.
9. Type environment variables **APPCLLU=local LU** (**5**) and **APPCTPN=Invokable TP** (**7**) separated by the pipe character in the **Environment** field.
10. Click **OK** to save your definition.

SNAP–IX operation: The SNAP–IX control daemon is started with the **sna start** command. Depending on the options selected at installation, it may already be running.

The **xsnaadmin** utility controls SNAP–IX resources.

Logging and tracing are controlled from here. Log and trace files can be found in the /var/opt/sna directory. The logging files sna.aud and sna.err can be read using a standard editor such as vi.

In order to read the trace files **sna1.trc** and **sna2.trc** they must first be formatted by running the command **snatrcfmt -f sna1.trc -o sna1** which produces a sna1.dmp file that can be read using a normal editor.

The configuration file itself is editable but this is not a recommended method of configuring SNAP–IX.

The APPCLLU environment variables must be set before starting a sender channel from the Sun Solaris system. The command can be either entered interactively or added to the logon profile. Depending on the level of BOURNE shell or KORN shell program being used, the command will be:

```
export APPCLLU=SOLARXLU      5    newer level
```

or

```
APPCLLU=SOLARXLU             5    older level
export
```

What next?: The connection is now established. You are ready to complete the configuration. Go to "MQSeries for Sun Solaris configuration" in Chapter 18 of the *MQSeries Intercommunication* book.

MQSeries for Sun Solaris sender-channel definitions using SNAP-IX SNA

```
def ql (OS2) +                                    F
    usage(xmitq) +
    replace

def qr (OS2.REMOTEQ) +                            D
    rname(OS2.LOCALQ) +                           E
    rqmname(OS2) +                                C
    xmitq(OS2) +                                  F
    replace

def chl (SOLARIS.OS2.SNA) chltype(sdr) +          G
    trptype(lu62) +
    conname('OS2CPIC') +             14
    xmitq(OS2) +                                  F
    replace
```

New function for AIX only

This section introduces the new function that applies only to MQSeries for AIX, V5.2.

Support for draft 10 threads

Version 5.2 of MQSeries for AIX uses the POSIX standard threading library (which was not available on AIX V4.2) to match the implementation on other UNIX platforms. Existing MQSeries appplications built on AIX 4.2 using the draft 7 level of POSIX threads are not affected by this new implementation and will continue to run unchanged. However, MQSeries exits and installable services should be recompiled and relinked using the xlc_r compiler on AIX 4.3 to use the final level of the pthread standard definition (also known as the draft 10 level). It is recommended that any new threaded applications on AIX 4.3 also be written to use this level of the pthreads standard.

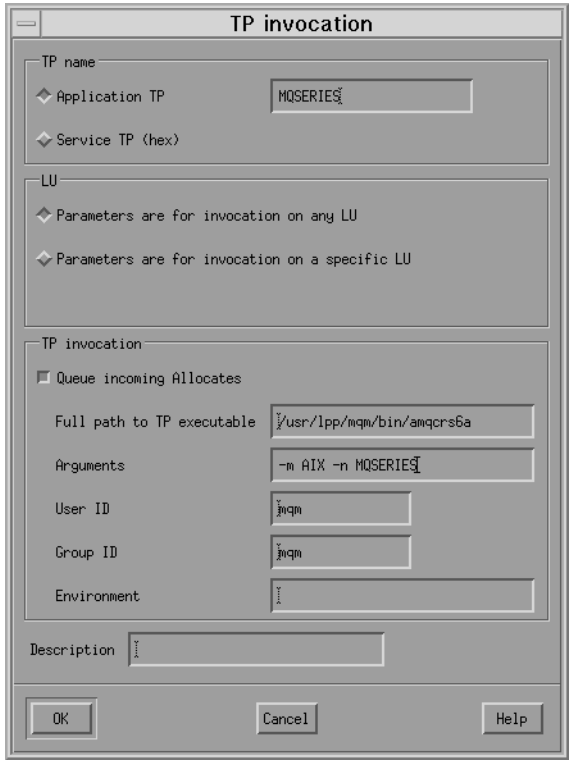
Enhanced support for Communications Server for AIX V5

Communications Server for AIX V5 support has been enhanced to enable you to use the graphical interface to configure transaction programs invokable by MQSeries. This description should be read in conjunction with Chapter 14 of the *MQSeries Intercommunication* book.

Note: MQSeries no longer supports Communications Server for AIX V4.

The section "Defining a transaction program" in Chapter 14 of the *MQSeries Intercommunication* book refers to MQSeries versions earlier than V5.2. For V5.2, you define a transaction program as follows:

From the main window, click **Services**, **APPC**, and **Transaction programs ...** The following panel is displayed:



1. Type **TP name** (**6**) in the **Application TP** field.
2. Clear the **Queue incoming Allocates** check box.
3. Type the **Full path to executable** (**7**).
4. Type **–m Local queue manager** in the **Arguments** field.
5. Type **mqm** in the **User ID** and **Group ID** fields.
6. Enter environment variables **APPCLLU=local LU** (**4**) and **APPCTPN=Invokable TP** (**6**) separated by the pipe character in the **Environment** field.
7. Click **OK**.

Refer to "Explanation of terms" in Chapter 14 of the *MQSeries Intercommunication* book for a description of the terms marked with a reference number.

Chapter 4. New function in MQSeries for AS/400 V5.2 only

This chapter introduces the new function that applies only to MQSeries for AS/400, V5.2. It contains these sections:

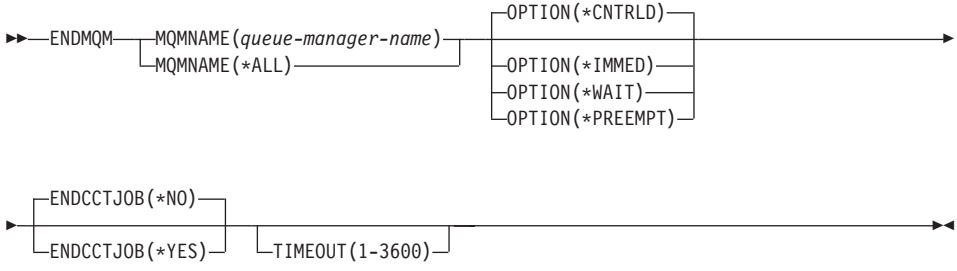
CL commands

This section describes the new ENDCCTJOB option to the ENDMQM (End message queue manager) CL command. For other additions and modifications to the CL commands, see:

- "RFRMQMAUT command for MQSeries for AS/400" on page 6
- "RCDMQMIMG command for MQSeries for AS/400" on page 22

ENDCCTJOB option to the ENDMQM (End message queue manager) CL command

The ENDCCTJOB option is added to the ENDMQM CL command. The function of this option is to quiesce the queue manager. The syntax of the ENQMQM command is now:



The description of the parameters for the ENDCCTJOB option is:

***NO** The queue manager or queue managers are ended. No further action is taken.

***YES** The following tasks are performed:

- Stop all channels (except receiver, client-connection and cluster-receiver channels) defined for the queue manager.

- Record media images for all objects defined for the queue manager.
- End the queue manager.
- Terminate processes connected to the queue manager.
- Destroy all shared memory and semaphores used by the queue manager.

Exception handling

This section describes exception handling in the MQSeries for AS/400, V5.2 product. It should be read in conjunction with the general discussion of "UNIX signal handling on MQSeries V5.2 products" on page 52 in Chapter 3. New function in V5.2 UNIX® systems only. For AS/400, the sections "Unthreaded applications" and "Threaded applications" apply, rather than the corresponding sections in Chapter 3. New function in V5.2 UNIX® systems only.

MQSeries for AS/400 uses ILE/C condition and cancel handlers as its exception processing mechanisms. Because of this, applications must not use the ILE/C signal() API when connected to MQSeries. The signal() API is implemented by ILE to handle ILE/C conditions as if they were signals, and can interfere with the ILE/C condition handlers used by MQSeries.

Sigaction() and sigwait() are safe to use with MQSeries, because they do not interact with ILE conditions at all. The ILE condition and cancel handler APIs are also safe to use in all circumstances. These APIs, when used together, will handle the same combination of exception conditions as signal().

Unthreaded applications

Each MQI function sets up ILE/C condition and cancel handlers for the duration of the MQI function.

Threaded applications

This section describes how AS/400 handles exceptions for threaded applications.

Synchronous signals

Synchronous signals arise as exceptions in a specific thread. AS/400 allows ILE/C condition and cancel handlers to be set up to process exceptions for each thread. MQSeries sets up its own exception handlers for the duration of each MQI function.

Asynchronous signals

MQSeries does not make use of any asynchronous signals in threaded applications. However, client applications might do so.

Access to MQSeries objects

If you are an AS/400 user with *ALLOBJ authority, and your MQSeries authorizations are controlled by the OAM, MQSeries V5.2 automatically allows you access to all MQSeries objects through the MQI. It is no longer necessary for you to edit the OAM authorization files to obtain the access you require. See Chapter 5 in the *MQSeries for AS/400 System Administration* book for more information about security considerations and the Object Authority Manager (OAM).

See the *OS/400 Security — Reference V4R4* SC41–5302 book for a complete description of the *ALLOBJ authority.

Support for nonthreaded listener

With MQSeries V5.2 the STRMQMLSR command is changed to submit by default a nonthreaded listener job called AMQCLMAA that starts each channel as a separate job (AMQRMCLA). With V5.1 the STRMQMLSR command submits a listener job RUNMQLSR and all channels started for that listener run as threads within that listener.

You control whether listeners run threaded or nonthreaded with the *ThreadedListener* parameter in the qm.ini file. This parameter is added to the `CHANNELS` stanza:

ThreadedListener=No | Yes
: This attribute controls whether listeners run threaded or nonthreaded. The default is No.

Refer to the *MQSeries for AS/400 System Administration* book for more information about the qm.ini file.

Nonthreaded listeners have the following advantages:

- They are more scalable than threaded listeners.
- They start nonthreaded channel jobs which run slightly more quickly than threaded channels. This makes nonthreaded channels suitable for heavily loaded batch processing environments.
- Nonthreaded channel exits do not need to be thread safe, which makes it possible to write exit programs in languages that do not offer full support for threading.

Threaded listeners have the following advantage:

- They start channels as threads which start more quickly than nonthreaded channels. This makes them suitable for conversational applications where channels are expected to start and end frequently.

Nonthreaded listener

Appendix. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make

improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

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

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	AS/400	CICS
IBM	IBMLink	MQSeries
MVS/ESA	OS/2	OS/390
OS/400	System/390	VSE/ESA
Websphere		

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

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

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom

- By fax:
  - From outside the U.K., after your international access code use 44–1962–870229
  - From within the U.K., use 01962–870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC34-5761-01

Spine information:

MQSeries®

MQSeries V5.2 Release Guide

Version 5.2