Version 8 Release 0

IBM MQ for HPE NonStop V8

Version 8 Release 0

IBM MQ for HPE NonStop V8

Note
Before using this information and the product it supports, read the information in "Notices" on page 101.

This edition applies to version 8 release 0 modification 1 of IBM MQ for HPE NonStop (product number 5724-A39) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Overview

IBM® MQ Version 8 functionality is now available on HPE NonStop platforms.

IBM MQ has been available on several HPE NonStop platforms up to Version 5.3.1 (with various fixpacks). These releases are not supported on the HPE NonStop X platform. For HPE NonStop X, there is a release available that is based on IBM MQ Version 8.0. This release is available for HPE NonStop servers with Itanium processors (J-Series OS starting from J06.20) and x86 processors (L-Series OS starting from L16.05). This new release of IBM MQ will eventually replace WebSphere MQ V5.3.1 on all supported HPE NonStop platforms (although WebSphere MQ for HP NonStop Server V5.3.1 will continue to be supported for some time).

IBM MQ V8 has many new features when compared with WebSphere MQ V5.3.1, most of which are available in the NonStop release. While all applications running on HPE NonStop using WebSphere MQ V5.3.1 can run unchanged on IBM MQ for HPE NonStop V8, there are differences regarding installation and administration of the product. This documentation covers these operational aspects of IBM MQ for HPE NonStop V8.

IBM MQ for HPE NonStop V8 is similar to the product on Linux and UNIX platforms. Most of the tools and options available on Linux and UNIX platforms work in exactly the same way on HPE NonStop. This documentation covers only the differences between IBM MQ for HPE NonStop V8 compared to Linux and UNIX platforms.

As many NonStop IBM MQ users are familiar with WebSphere MQ V5.3.1 on that platform, differences between IBM MQ for HPE NonStop V8 and WebSphere MQ for HP NonStop Server V5.3.1 are explicitly identified in this documentation.

Chapter 2. What's new

What's new in various releases of IBM MQ for HPE NonStop V8.

What's new for IBM MQ for HPE NonStop V8.0.1

New features in IBM MQ for HPE NonStop V8.0.1.

IBM MQ for HPE NonStop V8.0.1 supports the following new features:

Enhanced migration utility
The import/export migration utility has been enhanced to support migration between IBM MQ for HPE NonStop V8 queue managers (for example, for moving a queue manager from a J Series platform to NonStop X). You can also move queue data from an IBM MQ for HPE NonStop V8 queue manager back to an WebSphere MQ for HP NonStop Server V5.3.1 queue manager. This is meant to be used in scenarios where a fallback from Version 8 to Version 5.3 is required. See Migrating between IBM MQ versions for details.

Partitioned queue files
IBM MQ for HPE NonStop V8 now supports partitioned queue files. Partitioning rules are set using the enhanced **runnscnf** tool. (There is no support for using other platform-specific tools such as FUP.) See Chapter 12, “Migrating to alter or partition queue files,” on page 73 for details.

Online queue file migration
Queue files can be moved to a different disk, partitioned, repartitioned, or resized while the queue manager is running and while the queue is in active use. This includes system queues. Queue file migration can be also done while the queue manager is down (offline migration). Usually migration is started using the **runnscnf** tool, but **altmqfls** can also be used. See Chapter 12, “Migrating to alter or partition queue files,” on page 73 for details.

Installer capable of upgrading existing IBM MQ for HPE NonStop V8 installation
The new installer introduced with IBM MQ for HPE NonStop V8.0.1 can be used to install a fresh instance of the product, or to upgrade an existing installation. Upgrading an existing installation preserves all queue manager configuration and data. See Chapter 4, “Installing and upgrading,” on page 9 for details.

Tuning agent processes
You can edit the tuning stanza in a queue manager `qm.ini` file to control the agent processes used by the queue manager. Information is now provided on the available attributes, see “Tuning agent processes” on page 25 for details.

What's new for IBM MQ for HPE NonStop V8.0.3

New features in IBM MQ for HPE NonStop V8.0.3.

IBM MQ for HPE NonStop V8.0.3 supports the following new features:

New SetSignalManager class for runnscnf
The SetSignalManager is a type of process used to provide the MQGMO_SETSIGNAL service. Multiple SetSignalManager objects can be configured. See "Class SetSignalManager" on page 42.

Timeout feature for SetSignalManager
The InactivityTiimeout attribute of the SetSignalManager class can be configured to make the behaviour of SET_SIGNAL similar to the version 5.3 equivalent. See "Class SetSignalManager" on page 42.

What's new for IBM MQ for HPE NonStop V8.0.4

New features in IBM MQ for HPE NonStop V8.0.4.

IBM MQ for HPE NonStop V8.0.4 supports the following new and changed features:

Publish/subscribe now supported
IBM MQ for HPE NonStop V8.0.4 now supports publish/subscribe messaging, as provided by distributed platforms. See Publish/subscribe messaging.

CERTVPOL attribute now supported
In IBM MQ for HPE NonStop V8.0.4 you can now set the CERTVPOL attribute for queue managers. By default, the attribute is set to `RFC5280` to apply only the RFC 5280 compliant certificate validation policy. See ALTER QMGR.

New EmitEMSforExpiry property for runnscnf QueueManager class
The `EmitEMSforExpiry` property controls whether to emit EMS messages about processed expired messages. See "Class QueueManager" on page 40.

Can now install into existing OSS folder
You can now install IBM MQ into an existing, non-empty OSS folder. See "Installation procedure" on page 10.

Can now suppress MQGET error 2208 after Cache Manager failure
A new class `QueuePattern` property, `IgnoreNPMsgLossErrorOnMQGET`, has been added. Set this property to `T` to suppress the MQGET error 2208 after Cache Manager failure. See "IBM MQ processes unique to IBM MQ for HPE NonStop V8" on page 17 and "Class QueuePattern" on page 41.

Restricted support now available in TNS non-native applications for MQOD structures with version number greater than 1
In previous releases, TNS non-native applications could not have MQ object descriptor (MQOD) structures in an MQOPEN call with a version greater than 1. IBM MQ for HPE NonStop V8.0.4 supports MQOD structures with a version greater than 1, but does not accept MQOD structures containing any MQ object records (MQORs/MQRRs). See MQOD - object descriptor.

Chapter 3. Enhancements and limitations

IBM MQ for HPE NonStop V8 is implemented with NonStop fundamentals in mind, while preserving as many of the features of IBM MQ for distributed platforms as possible.

In addition to other features, the performance-related characteristics of the NonStop platform have been considered. This results in a product that, from a feature and usability perspective, is close to the general IBM MQ product, even though the internal engine is different. The consequences are documented in these sections.

IBM MQ logger

There is no transactional logger in IBM MQ for HPE NonStop V8.

IBM MQ on distributed platforms like Linux, UNIX, and Windows, has its own internal transactional engine, and its own transaction logging mechanism. On the HPE NonStop platform, the internal transactional engine is replaced by a tight integration with the platform-specific transactional engine, TMF by HPE. This replacement means that most configuration, and all queue data files, are protected by TMF, and can be replicated by audit trail reading based replication products like RDF or Shadowbase. For information on configuration files, see Chapter 7, "Configuring IBM MQ," on page 27.

No options or features relating to the IBM MQ logger are supported on the HPE NonStop platform. This exclusion includes options regarding size and location of logs, the difference between linear and circular logging, and so on. Trying to use logger related options results in an error message.

The IBM MQ logger as a transactional logger must not be confused with the IBM MQ error logs. These are present in the appropriate directories as usual.

XA server support

There is no XA server support in IBM MQ for HPE NonStop V8.

XA is a technology standard that enables heterogeneous transactions crossing platform boundaries.

The IBM MQ for HPE NonStop V8 server does not support XA transactions initiated by clients running on other platforms. This limitation also applied to the WebSphere MQ for HP NonStop Server V5.3 version on the platform.

IBM MQ for HPE NonStop V8 comes with an IBM MQ client environment. The IBM MQ client on HPE NonStop integrates with TMF. The client is able to initiate XA transactions that communicate with an IBM MQ server, which in turn supports the server side XA protocol. By using the IBM MQ client you can write an application that starts a TMF transaction on the HPE NonStop platform, then calls IBM MQ functions like MQPUT and MQGET via the client interface connected to an MQ server on platforms like Linux and Windows. This approach provides transactional consistency between the TMF transaction and the operations on queues on the remote server. Because the IBM MQ implementation on HPE

NonStop platforms does not support XA, this mode of operation is not possible when the MQ client on NonStop connects to a queue manager running on a NonStop platform.

Crypto hardware

Crypto hardware is not supported on IBM MQ for HPE NonStop V8.

Because crypto hardware is not supported, the SSLCRYP property is not supported for HPE NonStop when you use the **DISPLAY QMGR** command.

OSS and Guardian support

IBM MQ for HPE NonStop V8 is an OSS based product.

OSS stands for Open System Services, which is HPE NonStop's "UNIX personality". Administration and monitoring is achieved by using OSS based tools. All IBM MQ queue manager processes are OSS processes. Error logs, FFST files (detailed error descriptions for problem analysis), traces, and so on. are all located in the OSS file tree.

Applications that use IBM MQ API functions can run under OSS or Guardian without any functional differences. Tools used to configure and monitor the product (**runmqsc**, **strmqm**, and so on) are delivered both as Guardian and as OSS programs.

All dynamically changing queue manager data is located in TMF audited files in the Guardian file space.

Support for non-native applications

IBM MQ for HPE NonStop V8 supports non-native (code 100) Guardian MQI applications.

Such applications must be re-linked, see Chapter 13, "TNS non-native application support," on page 81 for details. Code 100 applications can be re-linked even if source code for the application is not available. Support for non-native applications is limited to the features and functions of the MQI offered by IBM MQ V5.3. Existing MQI applications can continue to be used with IBM MQ for HPE NonStop V8 without the need to migrate to native code.

Note that IBM MQ for HPE NonStop V8 does not support MQAI non-native applications.

High availability features

By default, IBM MQ for HPE NonStop V8 runs in fault tolerant mode.

Running in fault tolerant mode means that the central queue manager process, the execution controller (EC), runs as a NonStop process pair. In the case of a CPU outage, the EC backup process initiates a takeover and starts a new backup. Connected applications receive an error, but can immediately reconnect. After a short takeover period, new connections are possible and applications can continue. Transactions in progress might be aborted due to the takeover mechanism (depending on the transaction activities underway at the time of the failure). For details see, Chapter 8, "High availability (HANSQM)," on page 53.

Data storage

All queue data and all implicit configuration data and metadata is stored in TMF-protected (audited) Enscribe files.

Implicit configuration data is data generated internally by the queue manager when queue manager objects such as queues or channels are created, deleted, or altered. All this information is TMF protected. Explicit configuration data is configuration data given by the user in OSS based `.ini` files (`qm.ini`, `qmproc.ini`). This data is not TMF protected. For replication scenarios, `.ini` files must be copied to the replicated site either manually or by using appropriate tools or scripts. These files usually only change occasionally.

For each IBM MQ queue there is exactly one Enscribe queue data file. Queue data files can be stored on different Guardian volumes or subvolumes based on queue name patterns. For details see Chapter 7, "Configuring IBM MQ," on page 27.

For IBM MQ for HPE NonStop V8.0.1 onwards, queue data files can be partitioned. See "Partitioning" on page 77 for details.

In a replication environment the `qm.ini` file must always be copied to the replicated site after a queue manager is created or deleted (that is, **`crtmqm`** or **`dltmqm`** is executed).

Application-initiated TMF transactions

IBM MQ for HPE NonStop V8 supports application-initiated TMF transactions in the same way that WebSphere MQ for HP NonStop Server V5.3 on NonStop does.

An MQPUT or MQGET operation with syncpoint option set, and called while a user-initiated TMF transaction is active, results in the queue manager inheriting the application-initiated transaction. All file operations to the queue data files are completed under the control of the application-initiated transaction.

When no application-initiated transaction is active, IBM MQ implicitly starts a TMF transaction when required. When functions like MQGET or MQPUT are called and the SYNCPOINT option is set, MQCMIT or MQBACK must be called to terminate the implicitly started transaction. The rules are the same as for IBM MQ on other platforms. At any point in time either an application-initiated transaction, or an implicitly started MQ transaction might be active. Trying to mix those will result in an error with reason code `MQRC_UOW_MIX_NO_SUPPORTED (2355)`.

The general platform-specific rules are the same as for WebSphere MQ V5.3 on NonStop, so existing applications using application-initiated TMF transactions will continue to work without change.

CPU assignment

IBM MQ V8 makes substantial use of shared memory.

Shared memory on HPE NonStop platforms is only supported within one CPU. Therefore all processes of a queue manager (with a few exceptions documented in Chapter 7, "Configuring IBM MQ," on page 27) run in a single CPU. This limitation is automatically enforced by the product.

If the default high availability configuration is used, a backup process of the execution controller (EC, program `amqzxma0`) is started in a different CPU than the main queue manager. The backup EC also prestarts a few other processes running in its CPU, see Chapter 8, "High availability (HANSQM)," on page 53 for details. The resource consumption of these processes in the backup CPU is extremely low. There is no active checkpointing with the exception of the set signal manager. See the Chapter 10, "MQGET SET SIGNAL," on page 63 for details.

Application programs can run in any CPU.

IBM MQ tools such as **strmqm**, **crtmqm**, and **runmqsc** can be started in any CPU. Be aware that some of these programs might internally be restarted on the CPU of the queue manager, which is completely transparent. It might, however, be important to know that if you run (for example) **runmqsc** explicitly in CPU 2, and the queue manager runs in CPU 1, **runmqsc** will be moved from CPU 2 to CPU 1. If, for example, in that scenario CPU 1 fails, the `runmqsc` instance also fails, although it was initially started on CPU 2.

It is possible to create multiple queue managers from a single installation. These queue managers can be configured to run on different CPUs. See Chapter 7, "Configuring IBM MQ," on page 27 for details.

Bindings options

IBM MQ for HPE NonStop V8 supports three different kinds of bindings.

Bindings are the means of communications between applications and the queue manager. The following bindings are available:

Shared bindings
: Shared bindings are only available when the application runs on the same CPU as the queue manager, and internally use shared memory.

Isolated bindings
: Isolated bindings use a socket-based communication mechanism between applications and queue manager.

Guardian bindings
: Guardian bindings use Guardian based interprocess messages ("$RECEIVE messages") for communication between application and queue manager. Guardian bindings are not available on any other platform than NonStop.

The default binding mechanism for all user applications on NonStop is Guardian bindings. This mechanism provides good performance and CPU independence for applications.

Using other types of binding does not allow the inheriting of application-initiated transactions by queue managers. Any applications using BEGINTRANSACTION/ ENDTRANSACTION explicitly, or applications running as servers and inheriting TMF transactions from their clients, must use Guardian bindings to connect to IBM MQ (see "Application-initiated TMF transactions" on page 7).

IBM MQ internal applications might use other kinds of bindings for internal reasons. These applications are identified by their location in the bin directory of the installation tree (`.../opt/mqm/bin`). Under no circumstances store any user applications in that directory. User applications started from that directory are not supported.

Chapter 4. Installing and upgrading

How to install or upgrade IBM MQ for HPE NonStop V8.

Deliverables

IBM MQ for HPE NonStop V8 drivers are delivered as OSS executable programs.

The names of the files end in `.run`, for example:

- `mqs-8.0-hpe-nsi64.run` for J-Series OS
- `mqs-8.0-hpe-nsx64.run` for L-Series OS

The actual names of the files may vary depending on the exact version delivered.

In addition, there is a file with the same name but extension `sha1`. This file contains an `sha1` checksum of the deliverable and can be used to check the completeness and integrity of the deliverable.

Planning

Before installing the product, some planning is recommended.

The installation requires the following resources:

- About 2 GB of space in a single tree in the OSS file system; the files system should have capacity for 20,000 additional files. The 2 GB does not include space for errorlogs, FFST files, traces, and so on. The potential size of that dynamic data cannot be determined in advance.
- About 1 GB of space on a Guardian subvolume on an audited (TMF protected) Guardian disk. Installation on an SMF virtual volume is supported.
- Additional disk space for queue files, depending on the need and profile of your application.

IBM MQ can be installed multiple times on the same NonStop system. Each installation must have its own OSS directory and Guardian sub-volume and needs the same amount of free storage space.

IBM MQ can be installed on a NonStop system that already has the WebSphere MQ for HP NonStop Server V5.3 Server product, or the IBM MQ for HPE NonStop V8 client product installed.

You must check certain system parameters before installing or running IBM MQ for HPE NonStop V8.

The following TMF `BEGINTRANS` values should be set to at least these values or higher:

`RecRMCOUNT` 256

`RMOpenPerCPU` 1024

`BranchesPerRM` 512

IBM MQ might install with lower settings, but failures could occur in high load situations.

Installation procedure

To install IBM MQ for HPE NonStop V8, complete the following steps:

1. Login to OSS with a user id whose primary group is MQM. For example, MQM.USER or a Safeguard alias of that user.
2. Load the IBM MQ installation program in an OSS directory and make it executable. For example:

   ```
   chmod +x ./installer_name.run
   ```

   Where *installer_name* is the name of the installation program, for example, `ibm-mq-hpe-nsx64-8.0.1`.
3. Choose a new or empty OSS directory and an empty Guardian sub-volume for the new IBM MQ installation. The Guardian sub-volume must be on an audited (TMF protected) disk. The OSS and Guardian locations are passed to the IBM MQ installation program as shown in the next step.
4. Start the installation by running the IBM MQ installation program. The OSS and Guardian locations for the new IBM MQ V8 installation are passed to the IBM MQ installation program using the `-i` and `-g` options:

   ```
   ./installer_name.run -i OSS_directory -g Guardian_sub-volume
   ```

   The Guardian sub-volume can be specified in various forms:

   ```
   vol.subvol
   vol/subvol
   \$vol.subvol (Note: the $ must be escaped)
   /G/vol/subvol
   ```

For example:

```
./mqs-8.0-hpe-nsi64.run -i /home/user/mqv8 -g vol.mqv8
```

This command installs IBM MQ for HPE NonStop V8 into the `mqv8` OSS directory in the user's home directory, and into the `$VOL.MQV8` Guardian sub-volume.

Note: If you want to install IBM MQ into an existing non-empty OSS directory, you can specify the command line option `-f`. The existing directory that you specify must not contain `opt` or `var` directories.

The installation displays its progress as it runs. Depending on the power, size, and current workload of your system, installation typically takes less than 10 minutes.

Upgrade procedure

You can upgrade existing IBM MQ for HPE NonStop V8 installations while preserving existing configurations and data (including queue managers, queues, and queue data).

By default the upgrade procedure creates a backup of any files that it is about to change. The backups are located in the folder *OSS_directory*`/opt/mqm/backup`. The log files created while upgrading are located in the folder *OSS_directory* and begin with the string `mq_`.

You must end all queue managers before running an upgrade.

To upgrade, enter the following command:

```
installer_name.run --upgrade -i OSS_directory [-d] [-b]
```

Where:

- *installer_name* is the name of the installation program, for example, `ibm-mq-hpe-nsx64-8.0.1`.
- *OSS_directory* is the path to the existing installation. The path can be absolute or relative to the current working directory.
- You specify `-d` to delete the back up files of the current installation.
- You specify `-b` to skip making a backup of changed files during the upgrade.

The upgrade can take 60 minutes or longer on busy machines. You can track the progress of the upgrade by entering a command to tail the upgrade log file, for example:

```
tail -f ~/MQ8a/mq_8010-20170926-110637-upgrade.log
```

Where `MQ8a` is the OSS directory, and `mq_8010-20170926-110637-upgrade.log` is the most recent log file.

By default the upgrade process creates a backup of the currently installed release of the product. This part of the upgrade procedure consumes about 80% of the upgrade time. You can skip the backup by using the `-b` argument. Note that you cannot restore to previous versions without a backup created by upgrade.

If you attempt to use upgrade to restore the current release (for example, you attempt to upgrade a Version 8.0.1 installation to Version 8.0.1), the existing backup might act to prohibit the upgrade. In this case, specify the `-d` parameter to delete the backup files.

The following example shows a successful upgrade procedure:

```
$ ibm-mq-hpe-nsx64-8.0.1.run --upgrade -i MQ8a
MQ version installed: 8000
installer version: 8010
Checking for opened files in /home/mqm.user/MQ8a/opt/mqm

#-------------------------------------------------------------------------------
#  IBM MQ Server 8.0.1 for HPE NonStop X
#

#  Fixpack            : 8.0.1.0
#  Architecture       : nsx64
#  Build              : p800-L170925-182855
#
#  MQ Install Path    : /home/mqm.user/MQ8a
#                       $MQAS.MQ8A    (/G/mqas/mq8a)
#
#  MQ owner           : MQM.USER   12,34
#
#  System Name        : CS4
#  RVU                : L16.05
#  UNAME              : NONSTOP_KERNEL NSX-G
#  Default TCPIP      : $ZTC0
#  Locale             : C
#
#  Tue Sep 26 2017 11:06:37 MET DST
#-------------------------------------------------------------------------------
```

```
Checking for opened files in /home/mqm.user/MQ8a/opt/mqm                [OK]
Making backup. See log-file in the root directory for progress          [OK]
Upgrading installation                                                  [OK]
Installing SSL libraries                                                [OK]
Setting Guardian and OSS tree attributes                                [OK]
Generating message catalogs                                             [OK]
MQ upgrade successful                 [12:25 elapsed]                   [OK]
```

Restore procedure

You can restore to a previous version of IBM MQ for HPE NonStop V8.

If you have previously upgraded IBM MQ for HPE NonStop V8, you can restore to the previous version provided that you let the upgrade procedure create a backup. You must use the same installer program that you used to upgrade.

To restore, enter the following command:

installer_name.run --restore -i *OSS_directory* [-d]

Where:

- *installer_name* is the name of the installation program, for example, ibm-mq-hpe-nsx64-8.0.1.
- *OSS_directory* is the path to the existing installation. The path can be absolute or relative to the current working directory.
- You specify -d to delete the back up files after successful restoration.

Provided that you have not used any of the new feature installed by the upgrade, the restored system is equivalent to a new installation of the previous version, with configurations and data of queue managers and queues preserved.

Troubleshooting installation

After the installation completes, the installation log is available in the OSS directory specified by the -i option in the installation command.

Any problem observed during installation is documented in the log file. If the installation fails and assistance is needed to resolve the issue, IBM Support might ask for the installation log to identify the cause of the problem.

Chapter 5. Working with IBM MQ

Working with IBM MQ includes creating queue managers and routine tasks such as stopping and starting queue managers.

It also includes tasks such as compiling channel exit programs.

Creating a queue manager

Before creating a queue manager, you should understand requirements such as required disk space.

Planning disk space

Queue managers require disk space in the OSS file system and on Guardian discs.

Creating a queue manager creates directory entries in the OSS file system in the installation directory within the `var` subdirectory. The initial amount of data stored there during creation of the queue manager is small (about 1 MB). However this data can grow rapidly if severe problems occur. In that case the `AMQERRnn.LOG` files grow and (potentially many and potentially large) `FFST` files might be created in the appropriate error directories. So it is advisable to make sure there is enough space in the file system (number of files and space on disk) before creating queue managers.

Queue data files and some configuration files are stored in a configurable Guardian subvolume. The name of a queue data file is chosen by IBM MQ internally. The name always starts with the letter "Q". The **runnscnf** configuration tool (see Chapter 7, "Configuring IBM MQ," on page 27) has an option to show which logical queue is represented by which physical file (you can also use the **dspmqfls** tool for this purpose).

Each queue is by default initially created with 512 primary extents, 5008 secondary extents, and `maxextents` set to 50. This means the queue will initially use 1 MB of disk space. As soon as the file grows beyond 1 MB, it is extended in steps of roughly 10 MB. When the `maxextents` limit of 50 (filesize of roughly 51 MB) is reached, IBM MQ automatically tries to increase `maxextents` for the queue, so that it can grow further. The actual space needed depends on the profile of your application.

In addition to the queue data files, several configuration files are stored in the Guardian subvolume of the queue manager. These names start with letters "AMQ". The total disk space used by the configuration files is less than 10 MB, even if thousands of queues are created.

By default, all queue files are created in the same Guardian subvolume specified when creating the queue manager (see "Running crtmqm" on page 15). It is possible to configure IBM MQ for HPE NonStop V8 to store sets of queue files (determined by a pattern of the logical name) on different volumes/subvolumes. This configuration is done before the queues are actually created. First you decide which queue files are supposed to go onto which disk, then define the appropriate rules (see the Chapter 7, "Configuring IBM MQ," on page 27 for details). Then you create the queue files, so they are located at the correct place from the beginning.

It is also possible to define rules for primary/secondary/maxextents allocation on the basis of patterns for logical queue names.

Partitioned queue files are supported in IBM MQ for HPE NonStop V8.0.1.

Environment variables and PATH

You might need to set up environment variables and add to the PATH before working with IBM MQ queue managers.

All IBM MQ tools are located in the OSS directory *install_dir*/opt/mqm/bin, where *install_dir* is the installation directory. Tool location can either be qualified to point to that directory, or the directory can be included in the $PATH environment variable.

On Guardian, the IBM MQ tools are located in the IBM MQ installation subvolume.

IBM MQ for HPE NonStop V8 does not (normally) require any environment variables to be defined for IBM MQ administration tools or IBM MQ application programs to run (provided that application programs have been linked with the IBM MQ V8 libraries).

When running application programs that were previously compiled and linked with WebSphere MQ for HP NonStop Server V5.3, an _RLD_FIRST_LIB_PATH environment variable or define is required, as shown:

For OSS:

```
export _RLD_FIRST_LIB_PATH=install_dir/opt/mqm/lib: install_dir/opt/mqm/lib64
```

For Guardian:

```
add define =_RLD_FIRST_LIB_PATH,class search, subvol0 $vol.subvol
```

where $vol.subvol is the Guardian sub-volume chosen when IBM MQ was installed.

Environment variables used by WebSphere MQ for HP NonStop Server V5.3 (MQNSKOPTPATH and MQNSKVARPATH) are ignored by IBM MQ for HPE NonStop V8.

CPU considerations

All IBM MQ processes of a queue manager are running in a single CPU with some exceptions.

Cache manager process (executable amqcache)
The cache manager process stores all non-persistent messages in main memory. So, if your application profile makes heavy use of non-persistent messages, some scalability can be achieved by configuring IBM MQ to run the cache manager process in a different CPU than the queue manager itself. It is also possible to have multiple cache manager process instances for a single queue manager. You can configure a cache manager process for a set of logical queues given by a queue name pattern.

Set signal manager (executable amqssmgr)
The set signal manager process is involved when your application uses the MQGET SET SIGNAL feature. So, if this feature is heavily used by the application, configuring the set signal manager to run in a different CPU than the queue manager can potentially improve throughput. Like the

cache manager, the set signal manager is configured per logical queue name pattern. Note that the set signal manager process can optionally be run in fault tolerant mode as a NonStop process pair. In that case, checkpointing occurs and CPU usage of the backup process is in the same order of magnitude as for the primary. Both CPUs for the set signal manager (primary and backup) can be configured independently.

Running crtmqm

You run the command **crtmqm** to create a queue manager.

Note: As IBM MQ for HPE NonStop V8 uses TMF-enabled files for queue data storage, logger related **crtmqm** options are not supported. These are the following options:

- `-lc`
- `-ld`
- `-lf`
- `-ll`
- `-lp`
- `-ls`

You specify the Guardian sub-volume by using the –ng option of the **crtmqm** command.

As with the previous WebSphere MQ for HP NonStop Server V5.3, **crtmqm** chooses a new (empty) Guardian sub-volume if the `-ng` option is not specified on the command-line.

The Guardian sub-volume provided by `-ng` can be written in various forms:

```
vol.subvol
vol/subvol
\$vol.subvol (Note: the $ must be escaped)
/G/vol/subvol
```

Additionally, the `-ng` option accepts a trailing "+" wildcard in the sub-volume specification. When the Guardian sub-volume contains this wildcard, it is treated as a sub-volume prefix. Specified this way, the **crtmqm** command chooses an empty Guardian sub-volume with a name that begins with the specified value. For example:

```
crtmqm -ng data09.myqmgr+ MYQMGR
```

This command causes the **crtmqm** command to choose an empty Guardian sub-volume with a name starting with `$DATA09.MYQMGR`. The chosen sub-volume might be, for example, be `$DATA09.MYQMGR0` if that sub-volume was empty at the time the **crtmqm** command was run.

Note: The **crtmqm** option –x specifies the maximum number of uncommitted messages the queue manager supports. This corresponds to the TMF locklimit per transaction. So make sure that the TMF limit (which can be configured per disk) is at least as large as the limit used by IBM MQ. The default value for –x for IBM MQ is 10000, while the default lock limit for TMF is 5000. So either increase the TMF limit (this can be done online, no system interruption is required) or specify –x with a lower number than your current TMF configuration. Please consult your system administration, HPE, or IBM if in doubt.

Starting and ending queue managers

You start queue managers by using the **strmqm** command and end queue managers by using the **endmqm** command.

Because of the different implementation of IBM MQ for HPE NonStop V8, certain features of **strmqm** and **endmqm** are not supported.

Unsupported strmqm options

The NonStop implementation is different to other IBM MQ implementations in the following areas:

High availability

Instead of a standby queue manager running on a different system (MIQM, multiple instance queue manager), as with Windows or Linux, IBM MQ on NonStop implements the high availability NonStop queue manager (HANSQM) concept. See Chapter 8, "High availability (HANSQM)," on page 53 for details. Since HANSQM administration is different from MIQM, certain **strmqm** parameters available for other platforms are not supported on NonStop.

No logger

As IBM MQ for HPE NonStop V8 uses TMF enabled files for queue data storage, logger related **strmqm** options are not supported.

Specifically, the following **strmqm** options are not supported on NonStop:

- `-a` (MIQM related option)
- `-r` (MIQM related option)
- `-x` (MIQM related option)

Unsupported endmqm options

The following options for **endmqm** are not supported on HPE NonStop servers:

- `-s`
- `-x`

Both options are meant to be used in connection with Multi Instance Queue Managers, available on distributed platforms. On the NonStop platform fault tolerance is achieved by the HANSQM feature. See Chapter 8, "High availability (HANSQM)," on page 53 for details.

Starting a queue manager

As on all other platforms, a queue manager is started using **strmqm** after it has been created with **crtmqm**.

On NonStop platforms, the following extra considerations apply:

Backup process automatically started

If not configured otherwise (see Chapter 8, "High availability (HANSQM)," on page 53 for details), a full queue manager is started on one CPU, and the backup EC process plus several other queue manager processes are started on another CPU.

CPU assignment

If not configured otherwise (see Chapter 8, "High availability (HANSQM)," on page 53 for details), the primary queue manager is started on the CPU where **strmqm** is run.

You can explicitly specify which CPU to start the primary queue manager by using the `-cpu` option with **strmqm**. For example, if you want your primary queue manager QM1 to start on CPU 3, enter the following command:

```
run –cpu=3 strmqm QM1
```

To show the current CPU assignment for all queue managers of a given installation, use the **dspmq** command with option `–x`.

Ending a queue manager

On NonStop platforms, queue managers are ended by using the **endmqm** command as on all other platforms.

The following special considerations apply:

- The **endmqm** command always ends the primary and the backup instance of the given queue manager, regardless of the CPUs these are running on.
- Never try stopping a queue manager by using a **kill -9** command on queue manager processes. Because the EC process runs with a backup process, using the **kill** command can result in a takeover, which then might start another backup in a yet different CPU.
- If a hard kill of a queue manager is required for some reason, first stop the EC process pair using the TACL **STOP** command with the process name, as in **STOP $QMAB**. The process name of the EC is shown by the **dspmq –x** command. (If **dspmq -x** does not work, look for the process started from the executable `amqzxma0`). After stopping the EC process pair, usually all processes of that queue manager on both CPUs should be finished. If that is not the case, you can then stop all process pairs from executable `amqssmgr` (SET SIGNAL manager) and finally `kill -9` all potentially remaining processes.

  Note: This procedure is an emergency mechanism, and under normal circumstances should never be required.
- After ending a queue manager and looking at the process list (by using the **ps** command), you might find one or more instances of the **dspmq** program remaining, at most one instance in each CPU. This is normal and expected behavior. Since the **dspmq** program is common for the installation, and does not depend on a specific queue manager, these programs might run in the background. They may go away after a while, and will always go away when **dltmqm** is run. It is also safe to stop these instances manually; if an instance is required by any queue manager function, it is automatically restarted.

IBM MQ processes unique to IBM MQ for HPE NonStop V8

Several new process types have been added to IBM MQ to provide NonStop specific functionality.

Cache Manager Supervisor: `amqcchsv`

The cache manager supervisor process (executable `amqcchsv`) has the following tasks:

1. On demand, start any of the other NonStop specific process types documented in this topic. If any of these processes fails for whatever reason, the cache manager supervisor restarts it automatically.
2. At queue manager end, stop all potentially remaining processes. If certain processes of a queue manager fail (for example, the execution controller EC), typically a takeover to the backup instance running on a different CPU is initiated. To ensure consistency, it is important that no processes of the previous primary instance are running at that point. The cache manager supervisor takes care of that.

CPU consumption of this process is usually extremely small.

Cache Manager: `amqcache`

The cache manager (executable `amqcache`) stores non-persistent messages in main memory. You can configure multiple instances of this process with logical names, and can assign different queues to use different instances of the cache manager for scalability. If there are large queues with non-persistent messages, memory consumption of the cache manager can become large. If large memory consumption of the cache manager is observed, use **runmqsc** to view the current depth of queues, to see which queues might cause that. As the cache manager is a 64-bit process, its memory use can go up to 12 GB (the default setting for 64-bit processes on NonStop).

Due to the nature of the `amqcache` process, resource consumption (CPU and memory) might be significant.

Note:

If the cache manager fails, all non-persistent messages held by this instance are lost. Under general IBM MQ rules, non-persistent messages are not guaranteed. When the cache manager is restarted (which is done automatically), it immediately starts storing new non-persistent messages. At that point, the queue manager still knows about the lost non-persistent messages. If the application tries to retrieve one of those messages, for example, the first one on a queue, it receives an MQ error (2208 MQRC_FILE_SYSTEM_ERROR). The queue manager then forgets about that lost message. The application must be prepared to receive as many of these errors as there are lost messages due to the `amqcache` failure. After these errors have been received, the application will start receiving normal non-persistent messages again. So a failure of the `amqcache` process from an application perspective will mean a certain number of recognized errors and then continued business.

If required, you can suppress 2208 MQRC_FILE_SYSTEM_ERROR messages. You can specify that MQGET does not return this error, but instead returns the next available message or a different error than 2208 (for example, 2033 MQRC_NO_MSG_AVAILABLE).

To suppress error 2208 on MQGET, set the property `IgnoreNPMsgLossErrorOnMQGET` in the tool `runnscnf`:

```
class QueuePattern;
object myBigQueuePattern;
set pattern myBigQu*;
set IgnoreNPMsgLossErrorOnMQGET T;
```

where `myBigQueuePattern` is a name of a QueuePattern, and `myBigQu*` is a pattern matching queue names.

Setting IgnoreNPMsgLossErrorOnMQGET to F restores the default behavior.

Any changes only take in effect after a queue manager restart.

Config Manager: amqconfg

The amqconfg process keeps a cache of certain configuration settings made using **runnscnf** (see Chapter 7, "Configuring IBM MQ," on page 27 for details), and provides that to the queue manager. It is a restartable process, so if it fails it will be restarted and rebuild the cache. No outage should be observed.

Setsignal Manager: amqssmgr

The setsignal manager process (executable amqssmgr) is involved in all activity regarding the MQGET SET SIGNAL feature. You can configure multiple instances of this process with logical names and can assign different queues to use different instances for scalability. The process can optionally be configured to run as a process pair.

Note: An application using the MQGET SET SIGNAL feature typically waits for messages on $RECEIVE, indicating that **MQGET** can be called and will get a message. If no interprocess message appears on $RECEIVE, the application might wait forever. For this reason it is important to run amqssmgr as a checkpointing NonStop process pair. Even if the primary instance fails (for example, due to a CPU outage) the backup instance is still able to notify applications of the event, and so trigger continuation of business. Depending on your application, you should in addition use application timeouts so that even in outage situations applications never become completely unresponsive.

Due to the nature of the process, resource consumption (CPU and memory) might be significant.

Consistency Manager: amqcnmgr

The consistency manager, amqcnmgr, is started together with the queue manager, but is only used in situations where a takeover from primary to backup occurs. In that case, amqcnmgr ensures that the knowledge of the running queue manager regarding open interactions becomes consistent over time.

For example, assume that Application A starts a TMF transaction and then does an MQPUT operation that completes without error, the TMF transaction is not yet ended. Application B (which might be a queue manager internal application like a channel agent) has an MQGET for that queue outstanding. At that point of time, the queue manager fails and the backup takes over. The MQGET of application B will fail, application B will reconnect to the backup instance of the queue manager, reopen the queue, and reissue the MQGET. Because the MQPUT transaction of application A is not yet finished, the backup instance does not know anything about the message put by A, the corresponding records in the Enscribe file are still locked by the transaction. When now A ends the transaction, the queue manager must get notification about the MQPUT done by A, so that it can present the message to B. When B does the MQOPEN of the queue, the amqcnmgr process is involved and its task is to inform the queue manager about the outcome and the consequences of any outstanding transactions.

Technically, the amqcnmgr process could be started on a different CPU than the queue manager, but this does not make much sense. In normal operations, the

amqcnmgr process does not do any work at all, and even in takeover situations it is only involved for a short time. Resource consumption is not an issue.

Guardian Bindings Agent: amqzlga0

Each application connecting to the queue manager also connects to an agent doing the queue manager related work for that application. Depending on the kind of bindings used, different types of agents are used. When using shared bindings, the agent used is amqzlaa0. A process of this kind is usually always started with the queue manager, because this kind of connection is used by all internal components of the queue manager.

For isolated bindings, the corresponding agent executable is amqzlsa0. As isolated bindings are not recommended on NonStop, you will typically never see this process.

For Guardian bindings, the corresponding agent is amqzlga0. Since Guardian bindings are available only on NonStop, this process is only available on NonStop. The amqzlga0 agent is always threaded; non-threaded agents are not supported with IBM MQ V8 on NonStop.

Each amqzlga0 agent is able to handle up to 10 concurrent connections. If the application uses more concurrent connections, IBM MQ automatically starts more instances of the amqzlga0 process. If your application uses many concurrent IBM MQ connections, you will see many amqzlga0 instances. These processes do the main IBM MQ work, so they use significant CPU resources.

All amqzlga0 instances must run on the queue manager CPU. Although applications communicate with amqzlga0 using Guardian interprocess messages, and so can run on any CPU, the amqzlga0 instances use shared memory with other IBM MQ processes, and so must run on the queue manager CPU only.

Compiling channel exit programs on HPE NonStop systems

You can compile channel exit programs for IBM MQ on your HPE NonStop system.

In the following examples, exit is the channel library name and ChannelExit is the function name. These names are used by the channel definition to reference the exit program using the format described in MQCD- channel definition.

Sample compiler and linker command for channel exits on HPE NonStop I (ia64)

```
$ c89 -Wshared -Wsystype=oss -Wlp64 -Wextensions -D_PUT_MODEL_ -I$MQINST/opt/mqm/inc
  -Weld="-export_all -set data_model lp64" -L$MQINST/opt/mqm/lib64 -lmqds64_r -lput
  -o $MQINST/var/mqm/exits64/exit_r exit.c
```

Sample compiler and linker command for channel exits on HPE NonStop X (x86)

```
$ c89 -Wshared -Wsystype=oss -Wlp64 -Wextensions -D_PUT_MODEL_ -I$MQINST/opt/mqm/inc
  -Wxld="-export_all -set data_model lp64" -L$MQINST/opt/mqm/lib64 -lmqds64_r -lput
  -o $MQINST/var/mqm/exits64/exit_r exit.c
```

Triggering Guardian programs and TACL scripts from IBM MQ

You can trigger Guardian programs and TACL commands or scripts from IBM MQ by using the `mqtrig` script.

You reference the `mqtrig` script from the `APPLICID` parameter of an MQSC process definition (see DEFINE PROCESS). The `APPLICID` parameter specifies the type of object that is triggered using the following arguments:

-c $*vol.subvol.command* [*args...*]
Start a TACL command

-cv $*vol.subvol.command* [*args...*]
Start a TACL command with echo

-p /G/*vol/subvol/program* [*args...*]
Start a Guardian program

-5.1 Start TACL commands in IBM MQ 5.1 compatibility mode

`mqtrig` passes PARAM (environment) variables to the Guardian program or TACL command. These PARAMs represent the values from the MQTMC2 structure generated by the Trigger Monitor. PARAM variables are only generated for MQTMC members that contain data. The PARAMs that can be passed are listed in the following table:

Table 1. PARAM variables

PARAM name	MQSC Attribute	MQTMC Member
TRIGVER	-	Version (currently always set to 2)
TRIGQNAME	-	Qname
TRIGPROCESS	Queue PROCESS attribute	ProcessName
TRIGDATA	Queue TRIGGERDATA attribute	TriggerData
TRIGAPPLID	Process APPLICID attribute	ApplId
TRIGENVDATA	Process ENVRDATA attribute	EnvData
TRIGUSERDATA	Process USERDATA attribute	UserData
TRIGQMGRNAME		QMgrName

Guardian programs

Guardian programs are triggered by using the `-p` option of `mqtrig`.

```
mqtrig -p /G/vol/subvol/program [args ...]
```

No TACL shell is started when running `mqtrig` in this mode.

The Guardian program is given the TRIG PARAMS described in Table 1 and is also passed the MQTMC2 structure and ENVRDATA tokens as arguments, after any other program arguments.

When you specify the `mqtrig` command as the `APPLICID` parameter of the **`DEFINE PROCESS`** MQSC command, the DISPLAY PROCESS for that process might return the following:

```
display process(myprocess)
AMQ8407: Display Process details.
DESCR( )
APPLICID(mqtrig -p /G/vol/subvol/program arg1 arg2)
USERDATA(Hi there) ENVRDATA(my env data)
PROCESS(MYPROCESS) ALTDATE(2018-10-30)
ALTTIME(13.33.16) APPLTYPE(UNIX)
```

TACL commands

TACL commands can be Guardian programs, TACL routines or TACL macros. TACL commands can be triggered using the `-c` or `-cv` option of `mqtrig`, and these commands run under control of a TACL shell started by `mqtrig`:

```
mqtrig -c "\$vol.subvol.command [args ..]"
```

The TACL command is given the TRIG PARAMS described in Table 1, but is not passed the MQTMC2 structure and ENVRDATA tokens as arguments.

IBM MQ 5.1 compatibility mode

The MQ 5.1 trigger monitor passed the TMC to all TACL commands, but because of the TACL command line limit, only a portion of the TMC was delivered to the TACL command. `mqtrig` provides all useful fields within the TMC as PARAMs and does not pass the TMC at all on the TACL command line. If you want to preserve the MQ 5.1 behavior, you can use the `mqtrig -5.1` argument to force the TMC to be passed on the command line.

Chapter 6. Administering IBM MQ

Enhancements have been made to IBM MQ to make administering IBM MQ for HPE NonStop V8 easier.

Enhancements to `runmqsc`

For IBM MQ for HPE NonStop V8, enhancements have been made to the **runmqsc** command.

`FC` command

Support for the FC (fix command) syntax has been added so that **runmqsc** includes command history features that are similar to those offered by the TACL shell. You use the following **runmqsc** command:

`FC [num | string ]`

Use the **FC** command to edit and re-submit a command from the command history. **FC** accepts an optional command number or command string prefix argument. If neither argument is provided, **FC** presents the most recent command for editing. FC has the same command editing features as the TACL shell **FC** command:

! [*num* | *string* **]**
: The ! command re-executes a command from the command history. The ! command accepts an optional command number or command string prefix argument that can be used to select which command should be re-executed. If neither argument is provided, the ! command re-executes the most recent command.

H [*num* **]**
: The H command displays commands from the command history. The display includes the command numbers associated with each command. The command numbers can be used with either the **FC** or ! commands to select a particular command for processing.

  The H command accepts an optional numeric argument that specifies how many commands should be listed. If no numeric argument is provided, the H command displays the 10 most recent commands from the command history.

Additional information displayed

`runmqsc` has been enhanced for IBM MQ for HPE NonStop V8 to show the following additional or adapted information:

DIS QS(..) TYPE(HANDLE) ALL
: This command shows the TMF transaction id as the UOW id.

Guardian process ids
: Several commands show Guardian process ids (*process name*/*CPU*,*PIN*) in addition to, or instead of, OSS pids.

Specifying TCP/IP Transport for channels and listeners

For IBM MQ for HPE NonStop V8, enhancements have been made that allow you to specify the TCP/IP transport.

You can specify the TCP/IP transport process used by outgoing channels and listeners in the channel LOCLADDR attribute, the listener -i or -g options, or the queue manager qm.ini file.

LOCLADDR attribute

The Channel LOCLADDR attribute has been extended to support the optional specification of a TCP/IP transport process name. To specify a TCP/IP transport name using LOCLADDR, append a /transport argument to any existing LOCLADDR value, or specify /transport alone if there is no existing LOCLADDR value.

For example:

- LOCLADDR(/ztc0)
- LOCLADDR(hostname/ztc0)
- LOCLADDR(2.45.60.20/ztc0)
- LOCLADDR((1234)/ztc0)
- LOCLADDR((2000,3000)/ztc0)

The transport name following the / is case-insensitive and a leading $ is optional.

Listener switches

You can use the TCP/IP listener -g switch to specify a TCP/IP transport name to be used by the listener when listening for new incoming connections.

The transport name specified by -g is case-insensitive and a leading $ is optional. For example:

```
runmqlsr -t tcp -g ztc5 -p 2550
runmqlsr -t tcp -g \$ZTC0 p 2660
listener runmqlsr -i <ipaddr>
```

You can also use the TCP/IP listener -i switch to specify a TCP/IP transport using the same syntax described for the channel LOCLADDR attribute. Note that listeners ignore any port specifications in their -i value because the listening port is provided by the -p switch. For example:

```
runmqlsr -t tcp -i 2.45.60.20/ztc0 -p 2550
runmqlsr -t tcp -i /ztc0 -p 2550
```

The transport name specified by -i is not case sensitive and a leading $ is optional.

qm.ini file

If the channel LOCLADDR attribute does not have a TCP/IP transport specified, then the channel uses the TCP/IP transport specified by the qm.ini default transport value:

```
TCP:
   Transport=$ZTC0
```

Enhancements to dspmq

For IBM MQ for HPE NonStop V8, enhancements have been made to the **dspmq** command.

The dspmq –x command has been changed on NonStop to show CPU and PIN for the primary and backup EC of all running queue managers of that installation.

EMS messages

For IBM MQ for HPE NonStop V8, additional diagnostic messages have been added.

EMS messages have been implemented in addition to the IBM MQ error log and FFST mechanisms to display low level information (like Guardian file names and Guardian error codes) for failing Guardian API calls. For experienced NonStop operators, these messages can provide required information faster than FFST analysis. See “EMS” on page 91.

Tuning agent processes

You can edit the tuning stanza of a queue manager qm.ini file to control the agent processes used by the queue manager.

You can configure the following attributes in the qm.ini file tuning stanza:

GuardianAgentCapacity=*agentmaxthreadcount*

The **GuardianAgentCapacity** attribute configures the maximum number of worker threads allowed within each **amqzlga0** agent process. Each IBM MQ application connection is serviced by a worker thread within an **amqzlga0** agent process, and the **GuardianAgentCapacity** attribute controls how worker threads are distributed across agent processes.

The default value for GuardianAgentCapacity is 10 threads.

GuardianAgentsMax=*agentmaxcount*

The **GuardianAgentsMax** attribute configures the maximum number of **amqzlga0** agent processes that the queue manager can start. After the queue manager has started this number of agent processes, any further connection requests requiring a new agent process are rejected.

The default value for **GuardianAgentsMax** is 100 agent processes.

GuardianAgentIdleTimeout=*agentidleseconds*

The **GuardianAgentIdleTimeout** attribute configures the amount of time, in seconds, that an idle **amqzlga0** agent process waits for more work before ending.

The default value for **GuardianAgentIdleTimeout** is 300 seconds.

Chapter 7. Configuring IBM MQ

The HPE NonStop version of IBM MQ is different from the product on other platforms. Several NonStop-specific features have been added to the core product, while other features have been removed.

IBM MQ for HPE NonStop V8 is available for J-Series and L-Series HPE NonStop operating systems.

For HPE NonStop servers with J-Series operating system, WebSphere MQ Version 5.3 is also available. Although the two versions of IBM MQ are different in many respects, applications written for WebSphere MQ for HP NonStop Server V5.3 will run on IBM MQ for HPE NonStop V8. System management of IBM MQ for HPE NonStop V8 is, however, different from system management of WebSphere MQ Version 5.3.

The following topics describe NonStop-specific configuration options for IBM MQ for HPE NonStop V8, and how these options can be used. In some cases, where WebSphere MQ for HP NonStop Server V5.3 provides tools for configuring NonStop-specific properties, tools with the same interface are also available for IBM MQ for HPE NonStop V8. The new versions of the old tools have reduced functionality in some areas, however, because certain parameters are no longer needed or available for IBM MQ for HPE NonStop V8.

Most NonStop-specific parameters can be configured with a new program (`runnscnf`), which was not available on NonStop with WebSphere MQ Version 5.3, and is also not available on any other platform. Usage and options of `runnscnf` are described in this documentation. In some cases, **`runnscnf`** is an alternative to the tools that are compatible with WebSphere MQ for HP NonStop Server V5.3.

Note that this documentation only describes tools and options that are specific to NonStop, and not part of the general product. For tools that are part of IBM MQ V8 in general (like **`crtmqm`** or **`strmqm`**), see IBM MQ Knowledge Center https://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.helphome.v80.doc/WelcomePagev8r0.htm. If there are significant differences between a general tool and the NonStop version of that tool (for example, as for **`dspmqfls`**) these differences are described in this document.

There are also some area where you can use HPE NonStop tools to configure the NonStop system to run IBM MQ.

Configuration methods

There are several methods available to configure both general and platform-specific configuration parameters, either generally for an IBM MQ installation, or for a specific queue manager.

Parameters or settings applicable to distributed platforms as well as NonStop are accessible on NonStop in the same ways as for the other platforms. All general IBM MQ documentation that applies to both NonStop and other platforms is valid for NonStop. There is no NonStop-specific documentation that duplicates the general IBM MQ documents. This consideration applies to tools like **`runmqsc`**, **`crtmqm`**, **`dltmqm`**, and so on.

.ini files

.ini files contains values that are used to initialize queue managers.

There are two .ini files used for queue manager configuration:

mqs.ini

This file is available in the directory *var_installation_path*/var/mqm. The file contains settings applicable to the queue manager installation as a whole.

qm.ini

This file is available in the directory *var_installation_path*/var/mqm/qmgrs/*queue_manager_name*. The file contains settings applicable to the specific queue manager. For example, the path and the name for a configuration file for a queue manager called QMNAME is: *var_installation_path*/var/mqm/qmgrs/QMNAME/qm.ini.

All NonStop-specific settings in these .ini files can alternatively be configured by using **runnscnf**. Note that **runnscnf** stores information in an audited Enscribe file, so users of audit trail-based replication products get automatic replication of all settings made with runnscnf. If a setting is in an .ini file and also set with **runnscnf**, the **runnscnf** setting takes precedence.

NonStop-specific tools inherited from WebSphere MQ for HP NonStop Server V5.3

For reasons of compatibility with WebSphere MQ for HP NonStop Server V5.3.x, certain tools used with that release on NonStop servers are also available on IBM MQ for HPE NonStop V8.0.

These tools are **altmqfls**, **dspmqfls**, **altmqusr**, and **dspmqusr**. These tools are described in specific topics (see "altmqfls" on page 45, "dspmqfls" on page 46, "altmqusr" on page 47, and "dspmqusr" on page 48.

Compared to WebSphere MQ for HP NonStop Server V5.3, some of the existing tools have different or limited functionality. Different functionality is explicitly documented where applicable.

Environment variables

Some IBM MQ settings have traditionally been configured by using environment variables.

For compatibility with WebSphere MQ for HP NonStop Server V5.3, some of these environment variables are still supported by IBM MQ for HPE NonStop V8. However, all of these settings can also be configured by using **runnscnf**. Using **runnscnf** has the additional advantage that spelling of names and range of values are checked, so that the number of potential errors is reduced.

runnscnf

runnscnf is the new tool available with IBM MQ for HPE NonStop V8.

runnscnf is a command line utility similar to **runmqsc** or other prompting command line tools common on the platform. See "The **runnscnf** tool" on page 30 for information about usage and features.

Areas of NonStop-specific configuration

Certain areas of configuration are specific to IBM MQ on HPE NonStop.

Location and size of Guardian data files

IBM MQ for HPE NonStop V8 stores all dynamic data in audited Enscribe files. The data includes dynamically changing persistent queue data and object descriptions (such as channel definitions). The Guardian subvolume of these files and (for queue data files) primary extents, secondary extents, and maxextents can be configured by using **runnscnf**. See "The **runnscnf** tool" on page 30 for details.

Processes for non-persistent messages

Non-persistent queue messages are kept in main memory and managed by specific processes called cache managers. You can configure multiple per-queue manager cache managers. Cache managers can run on a different CPU than the queue manager, providing some scalability. Each cache manager is responsible for storing non-persistent messages of a configurable set of IBM MQ queues. The relationship between cache managers, IBM MQ queues, CPU, and so on, can be configured by using **runnscnf**.

Set signal settings

IBM MQ for HPE NonStop V8 supports the MQGET SET_SIGNAL feature in the same way as WebSphere MQ for HP NonStop Server V5.3. The signal can be used from programs as for WebSphere MQ for HP NonStop Server V5.3. As the implementation for Version 8 is different from Version 5.3, there are some settings available from **runnscnf**. The MQGET SET_SIGNAL feature in IBM MQ V8 is implemented using a new process class called `SetSignalManager`. Multiple `SetSignalManager` processes can be configured, with one process doing the work for one queue. `SetSignalManager` processes can run on different CPUs. `SetSignalManager` processes are configured by using **runnscnf**.

EMS subsystem

IBM MQ for HPE NonStop V8 can optionally issue more, and more detailed, EMS messages than WebSphere MQ for HP NonStop Server V5.3. The extent of EMS messages issued, and the collector used, can be configured by using **runnscnf**.

Queue manager global settings

There are certain NonStop-specific settings that are global to the queue manager. These settings include the home terminal being used, the CPU, the priority of certain NonStop-specific processes, tuning parameters, CPU set assignment for each queue manager, settings for fault tolerance, default TCP transport, and so on. All these settings are available by using **runnscnf**.

User name mapping

User names within IBM MQ have a length limit of 12 characters per name. NonStop user names can be up to 17 characters long. On NonStop, names consist of group name, a dot ('.'), and a user name, like: MYGROUP1.USER1. Group name and user name can each be up to 8 characters long, so the name of a regular NonStop user can be up to 17 characters long and thus does not fit in the data structures used by IBM MQ to store user identification.

WebSphere MQ for HP NonStop Server V5.3 provided the tools **dspmqusr** and **altmqusr** to create a mapping between NonStop-specific user names and names used within IBM MQ. This mapping can be used to establish a one-to-one relationship between IBM MQ internal names and NonStop user names. These tools are available in IBM MQ for HPE NonStop V8, with similar syntax and functionality to the WebSphere MQ for HP NonStop Server V5.3 tools. See“altmqusr” on page 47 and “dspmqusr” on page 48.

IBM MQ user names are sometimes referred to as Principal Names. The mapping described here is stored in an internal database. The **crtmqm** command creates this database and adds an entry for the user who ran the installation script (the installation owner). The principal created is always ‘mqm’, for compatibility with other IBM MQ implementations.

After you have created a queue manager, you can create entries in the database for other users of the queue manager.

You can use IBM MQ standard mechanisms to authorize NonStop users outside of the MQM group (which is reserved for IBM MQ administration) to use certain features and/or resources of an IBM MQ queue manager. For this authorization to be effective, the following conditions must be met:

- A mapping must be present in the internal database.
- Authorization must be explicitly granted to all resources accessed (see the **setmqaut** command).

NonStop users within the MQM group (other than the installation owner) can administer (create, start, stop, secure, and so on) any queue manager within the installation. No user name mapping is required to do so.

Applications require a mapping entry to use a queue manager (put, get, browse, and so on). If no mapping entry is found, application access is refused. No explicit or additional authorization (via `setmqaut`) is required, however.

The `runnscnf` tool

runnscnf is a tool that is used to change all kinds of different settings based on a common concept.

The tool uses the following terminology:

class

The term “class” means a set of objects of the same kind that have a common set of properties. By knowing the class of an object, **runnscnf** knows which changeable or viewable properties are available for the object, so only valid properties for that object and only valid values for the properties of an object can be entered. For supported classes and their properties see “Classes” on page 38. Class names are not case sensitive, so you can enter a class name in lower case, upper case, or any mixture of cases. Classes are defined by IBM MQ, you cannot introduce new classes or delete existing classes.

object

An “object” within **runnscnf** is the name of a member of the appropriate class. Examples of objects are instances of the CacheManager (a type of process that stores non-persistent messages in main memory), named

QueuePatterns (allowing rules for file size and file placement in the Guardian space), and so on. Names of objects are typically chosen by the user. In general, names of objects are case sensitive. For certain classes there is a fixed set of object names supported. See "Classes" on page 38.

Objects exist only as long as there is at least one configured or system maintained property for this object. When you set a property for an object, this object exists in the database maintained by `runnscnf`. When you delete the last property of an object, the object is also deleted.

If you want to use special characters in object names (like _, -, &, + and so on), the object name must be quoted in `runnscnf`. Without quoting, object names can consist only of characters and digits.

property

Each object has a set of named properties that can be changed and viewed by using **runnscnf**. A property of an object has a name and a value. For example, a property of the CacheManager CacheMan1 object is the CPU where the process is supposed to run. In that example, the class of the Object is "CacheManager", the name of the object is "CacheMan1", the name of the property is "CPU", and the value of the property is the CPU where the process is supposed to run. All objects of a given class have the same set of available properties. The values of the properties always have a default, which can be overwritten by using **runnscnf**. **runnscnf** only shows values that are explicitly set by the user.

This concept allows a common, simple way of changing properties for different objects in a common, unified way.

runnscnf has two different modes of operation, local and global. In global mode, properties are set for all queue managers within a given installation. In local mode, a specific queue manager must be specified, and settings apply only to that queue manager. If a setting is given for a specific queue manager, it overwrites any potential global setting of the same property. The mode can be switched between local and global at any time.

Settings made in **runnscnf** do not work on existing objects, the settings apply only to newly created objects. So, for example, **runnscnf** cannot be used to change attributes of an existing queue file, it can only be used to set parameters for a new queue file to be created. All settings made with **runnscnf** are persisted in a file. Settings remain even after the queue manager is deleted, so if you recreate a queue manager with a name known to **runnscnf**, the previous settings remain in effect. If you want to delete the settings completely, you must use **dltmqm** with the `-c` option.

Using `runnscnf`

The **runnscnf** tool is used by IBM MQ administrators to set and view configuration properties.

runnscnf is implemented as a prompting command interpreter. When the tool is started, it shows a banner and prompts, the user can then enter commands and view responses.

The tool can accept input files (I/O redirection), so it can be used in scripts. See "**runnscnf** command reference" on page 32 for the syntax of the tool.

Commands and other keywords are not case sensitive. Object names (like queue names) are case sensitive if, and only if, they are given as quoted strings.

Class names and property names are stored and displayed in a case sensitive form for better readability. However, you can specify them in any case on the command line.

The syntax descriptions use the following conventions:

[ ]

Indicates optional parameters. Parameters that are not enclosed in square brackets are required.

|

Indicates mutually exclusive parameters. You can use the parameter before or after the separator. You cannot use all options.

{ }

Indicates a set of mutually exclusive parameters when a parameter is required.

runnscnf can be started with the following command line option:

```
runnscnf [ queue_manager ]
```

Where *queue_manager* specifies the queue manager to operate on. If the name has special characters like spaces in it, it must be quoted. Specifying a queue manager is equivalent to giving the MODE LOCAL command with the queue manager as specified on the command line, see "MODE" on page 34.

If no argument is given on the command line, **runnscnf** works on the default queue manager. If no default queue manager is defined, a warning is issued, the mode is local, and no current queue manager is set.

In addition to setting configuration properties, **runnscnf** can also display specific information about queue managers. See "**runnscnf** command reference," and "Examples of using **runnscnf**" on page 43 for details.

Multiple commands can be given on one line, which makes scripting easier. Multiple commands on one line can be separated by a semicolon (;) character for clarity. Line breaks within one command are not allowed and cause a syntax error.

All objects and their properties are stored in an audited Enscribe database file (AMQPARMS). So when replication tools like RDF or Shadowbase are used for replicating to a backup system, changed settings are automatically replicated.

runnscnf command reference

Reference information for the **runnscnf** command is given in the following topics.

class

Set the class name to use.

Purpose

Set the class name to use. This setting stays valid until another class is set. So, if you set multiple properties of an object of a class, the name of the class and the object only needs to be set once.

For the set of available classes, see "Classes" on page 38.

Syntax

```
CLASS ClassName
```

Example

```
runnscnf Command Interface
Version 1.10, 2017-01-18

NSCNF>class QueuePattern
CLASS set to QueuePattern
NSCNF>
```

comment

You can add comments to scripts of **runscnf** commands.

Purpose

Any line starting with a star (*) character is treated as a comment.

Syntax

```
* any string
```

DEL

Delete the specified property of the current object.

Purpose

Note: An object only exists in the database as long as properties of the object are configured. So when you delete the last property of an object, the object is also implicitly deleted.

Syntax

```
DEL PropertyName
```

Example

```
NSCNF>del PrimaryExtents
Property deleted successfully.
```

DIS

Display Enscribe file names.

Purpose

Display the Enscribe file names for all queues matching the given pattern.

Syntax

```
DIS QueueNamePattern
```

Example

The following example shows the Enscribe file names of all local queues within the currently selected queue manager that start with the character "Q":

```
NSCNF>dis Q*
------------------------------------------------------------
Queue:                Q1
Guardian file:        $MQWA.MQ8W64Q0.QBZ8D4P3
```

FC

Edit the previous command.

Purpose

This is the well-known FC (fix command) command used to edit and potentially execute the last command again.

Syntax

```
FC
```

HELP

Give help for the **runsncf** commands.

Purpose

Show a syntax description of the available commands.

Syntax

```
HELP
```

LIST

List properties.

Purpose

List a set of properties as specified by the three (optional) parameters. Wildcard notation ("*" for any substring and "?" for any single character) can be used. If *Class*, *Object,* or *Property* are omitted, this is treated like a "*" wildcard. So LIST(,,) lists all properties for all objects in all classes in the currently selected (local or global) parameter file. This is the same as LIST without the parenthesis.

If the option ALL is given, the command in addition shows the creation and modification timestamp of the properties. It also gives the name, group, and user number of the user who last changed the property, and the name of the program doing the change.

Syntax

```
LIST([ Class ],[ Object ], [ Property ]) [ ALL ]
LIST [ ALL ]
```

MODE

Sets the **runscnf** mode to local or global.

Purpose

Set the mode:

- `GLOBAL` for installation wide settings
- `LOCAL` for queue manager-specific settings applied to the queue manager specified by *QueueManagerName*.

If **runnscnf** is started and no **MODE** command is given, local mode for the default queue manager is assumed. If no default queue manager is defined, the mode is local with no queue manager set.

Note: The mode set is valid only for the current **runnscnf** session.

Syntax

```
MODE { GLOBAL | LOCAL QueueManagerName }
```

OBEY

Executes the **runnscnf** commands in the specified file.

Purpose

Executes the **runnscnf** commands in the file specified by *filename*. *filename* must be the name of an existing text file. Obey files can be nested for up to ten levels, that is, an obey file can contain an obey command calling another file.

Syntax

```
O filename
OBEY filename
```

OBJECT

Set the object name to use.

Purpose

Set the object name to use. This setting stays valid until another object is set.

If the object name has special characters such as spaces in it, it must be quoted using double quote characters (“).

Syntax

```
OBJECT ObjectName
```

PREPARE MIGRATE

Prepare an online or offline migration for a set of queues.

Purpose

The PREPARE MIGRATE command is used to calculate a set of queues that should be migrated to new files, based on the **runnscnf** configured rules. The command is given a pattern for queue names. It then checks for all queues in the current queue manager matching the given pattern (if the actual queue file matches the configured parameters of the file).

The command checks the following parameters:

Subvolume
Is the physical file for that queue located in the configured subvolume?

Primary extent size
Does the queue file's primary extent size match the configured primary extent size?

Secondary extent size
Does the queue file's secondary extent size match the configured secondary extent size?

Number and location of partitions
Does the number and location of the files' actual partitions match the configured partition scheme?

If any of these checks results in a mismatch, the queue is a candidate for migration. The **PREPARE MIGRATE** command lists the candidates as a preparation for the **START MIGRATE** command. PREPARE MIGRATE only calculates and shows the list of candidates, it does not start any migration and does not change any configuration. If the list of queues to be migrated is too long, and would result in too many concurrent migrations being started, the **PREPARE MIGRATE** command can be repeated using a different pattern matching to produce a smaller list. Use this method to perform a number of smaller migrations.

Syntax

```
PREPARE MIGRATE QueueNamePattern
```

Example

The following example calculates the list of all local queues within the currently selected queue manager that start with the character “Q” and need to be migrated based on the current **runnscnf** settings:

```
NSCNF>PREPARE MIGRATE Q*
```

SET

Set the value of the specified property.

Purpose

Sets the property given by *Propertyname* to the value given by *Propertyvalue*. Mode, class, and object must have been set before. If the property could not be set, an error explaining the reason is shown.

Syntax

```
SET Propertyname Propertyvalue
```

SHOW

Show current settings.

Purpose

Shows the current settings of MODE, CLASS, and OBJECT. SHOW always displays the settings in the internal case sensitive form.

Syntax

```
SHOW
```

START MIGRATE

Starts the migration of a set of queue files.

Purpose

The **START MIGRATE** command is used to start the process of changing location, partitioning, or extent sizes of a set of queue files. The set of queue files affected is calculated by the **runnscnf PREPARE MIGRATE** command. Before using **START MIGRATE**, you must run a **PREPARE MIGRATE** command first.

START MIGRATE starts a migration process (executable file `amqoqmig`) for each queue to migrate. Existing queue files cannot be changed, but must be recreated and copied. This is done by `amqoqmig`. Depending on the number of queues in the set of queues to migrate this can have a performance impact.

Syntax

```
START MIGRATE
```

STATUS MIGRATE

Check the status of the migration of a set of queue files.

Purpose

Use the **STATUS MIGRATE** command to check the progress of migrations that you have started using the **PREPARE MIGRATE** and **START MIGRATE** commands. For each migration in progress, the following information is displayed:

- Name of the queue being migrated
- Old and new physical filenames
- Start time of migration
- Time migration was restarted (if it was interrupted)
- Process name of the migration process
- Number of records already moved
- Status of the migration

Syntax

```
STATUS MIGRATE
```

Classes

The following table lists supported classes.

Table 2. Classes

Name	Meaning	Available globally?
CacheManager	A CacheManager is a type of process used to store non-persistent messages in main memory. Multiple CacheManager objects can be configured.	N
CacheManagerSupervisor	The CacheManagerSupervisor is a single process starting and monitoring the CacheManagers. It is automatically started when the queue managers starts.	N
Processes	Certain properties can be configured per process.	N
Queue	A Queue is represented by a physical Enscribe file.	N
QueueManager	The queue manager itself. The object name in this case is always CurrentQMGR.	Y
QueuePattern	A QueuePattern defines certain properties available for all queues matching the pattern. **Note:** The name of the pattern is just a name (for example, "LargeQueues" or "TempQueues"), the value of the pattern itself (usually having wildcard characters) is specified using the property named "Pattern". See "Examples of using **runnscnf**" on page 43 for examples.	Y
SetSignalManager	A SetSignalManager is a type of process used to provide the MQGMO_SETSIGNAL service. Multiple SetSignalManager objects can be configured.	N

Class CacheManager

Class CacheManager has the properties listed in the following table.

Table 3. CacheManager properties

Name	Datatype	Meaning
CPU	Number	The CPU to run this CacheManager on.
Priority	Number	Priority for cache manager process.
ProcessName	Processname	Process name of CacheManager. This is automatically set by the system and cannot be changed.

Note: CacheManager processes do not need to run on the same CPU as the IBM MQ kernel processes. This gives at least a minimum of scalability.

Class CacheManagerSupervisor

Class CacheManagerSupervisor has the properties listed in the following table.

Table 4. CacheManagerSupervisor properties

Name	Datatype	Meaning
CPU	Number	The CPU to run this CacheManagerSupervisor on.
Priority	Number	Priority for cache manager supervisor process.
ProcessName	Processname	Process name of CacheManagerSupervisor. This will automatically set by the system and cannot be changed.

Note: CacheManagerSupervisor processes do not need to run on the same CPU as the IBM MQ kernel processes. However, because this process type uses only a small amount of resources, there is no benefit in moving it to a different CPU from the IBM MQ kernel. When the IBM MQ kernel ends (because of shut down or any kind of malfunction), the CacheManagerSupervisor and all CacheManagers also automatically end.

Class Processes

Class Processes has the properties listed in the following table.

Table 5. processes properties

Name	Datatype	Meaning
PrimaryCPU	CPU	Preferred CPU of primary process.
BackupCPU	CPU	Preferred CPU of backup process.
AllowedCPUs	CPU list	Comma-separated list of CPUs that can be used to start a process instance on.
ProcessName	Processname	Optional process name of the EC process.

The only object of class Processes is EC, the execution controller process. By default, this process runs as a NonStop process pair, with primary on the CPU given as PrimaryCPU, and backup on the CPU given as BackupCPU. If any of these CPUs is not available, IBM MQ chooses another one of the AllowedCPUs. If a CPU where either the primary or the backup instance runs fails, another primary or backup instance is started automatically on an available CPU from the list of allowed CPUs.

If you do not want to run the EC as a process pair (not recommended), you can achieve this by setting PrimaryCPU to the same CPU number as BackupCPU. In that case, the queue manager does not run in fault tolerant mode. If the EC fails, or the CPU used by the EC fails, the queue manager is completely unavailable.

Class Queue

Class Queue has the properties listed in the following table.

Table 6. Queue properties

Name	Datatype	Meaning
CacheManager	String	Logical name of the cache manager for this queue.
SetSignalManager	String	Logical name of the SetSignalManager for this queue.

Note the following points:

- By using different CacheManagers for different queues, the administrator can distribute the load for non-persistent message processing across CPUs.
- The settings for extents must be made before the queue is created, a subsequent change has no effect.
- To set primary and secondary, and maxextents extents, use objects of class QueuePattern (see "Class QueuePattern" on page 41).
- If the value of the SetSignalManager property for an existing queue of a running queue manager is changed, the new value takes effect when the queue is opened by any IBM MQ process. Do not confuse this with MQOPEN calls by applications. An MQPEN call might or might not result in a new (OS level) open to the queue. The new setting of the SetSignalManager property is used when a new OS level open to the queue occurs. All new settings take effect immediately when a queue manager is restarted. Due to this rule, multiple set signal managers might be running for a single queue until the queue manager is restarted if the property SetSignalManager for this queue was changed while the queue manager was running.
- As the cache manager maintains non-persistent messages for queues in its process memory, changing the CacheManager property for existing queues while the queue manager is running is not possible. So the CacheManager property for a queue can be changed if the queue manager is in stopped state or the queue does not yet exist.

Class QueueManager

Class QueueManager has the properties listed in the following table.

Table 7. QueueManager properties

Name	Datatype	Meaning
EmitEMSforExpiry	Bool	Control whether to emit EMS messages about processed expired messages.

Note the following points:

- An EMS message about processed expired messages is emitted every 10,000th expired message by default, or every *X* seconds since the last processed expired message. The time *X* seconds is configured by setting `ExpiryInterval` in stanza `TuningParameters` in the queue manager specific configuration file `qm.ini`. The value `ExpiryInterval` is also used for the `expirer` task.
- If you want to turn off the EMS messages for expiry, set the value of the property `EmitEMSforExpiry` to `f`. Set `EmitEMSforExpiry` to `t` to enable the EMS

messages again. Changing `EmitEMSforExpiry` takes effect after a queue manager restart. The EMS message has the event number 5 and the unique id `EMS_probeId_1050`.

Class QueuePattern

Class QueuePattern has the properties listed in the following table.

Table 8. QueuePattern properties

Name	Datatype	Meaning
CacheManager	String	Logical name of the cache manager for all queues matching this pattern.
IgnoreNPMsgLossErrorOnMQGet	Bool	Suppresses MQGET error 2208 after Cache Manager failure. See "IBM MQ processes unique to IBM MQ for HPE NonStop V8" on page 17.
Level	Number	Pattern matching starts with patterns of the highest level going down levels. When the first match is found, matching ends. The allowed range for levels is 0 to 999, gaps are allowed.
Maxextents	Number	Maxextents used when creating a queue file for a queue matching this pattern.
NumberPartition	Number between 1 and 16	Number of partitions for queue files matching this pattern
Pattern	String	Pattern a queue name needs to match.
PrimaryExtents	Number	Number of primary extents used when creating a queue file matching this pattern.
SecondaryExtents	Number	Number of secondary extents used when creating a queue file matching this pattern.
SetSignalManager	String	Logical name of the SetSignalManager for all queues matching this pattern.
Subvolume	Subvolume	The subvolume a queue file matching this pattern is created on. The subvolume must be on an audited disk.
VolumePartition2... VolumePartition16	Volume	Disk volume for the partition, only allowed if NumberPartitions is set to more than 1.

Note the following points:

- A pattern can consist of all characters allowed for MQ object names plus the meta characters * and ?. * matches any substring, while ? matches a single character.
- Objects of class QueuePattern can be added or changed while the queue manager is running, but this has an effect only on newly created queues.
- Any match for a local pattern prevents matching using global patterns.
- Adding a Pattern * in mode GLOBAL sets defaults for all queues in all queue managers created from this installation.

It is important to understand that the name of an object of class QueuePattern is not the pattern itself, but only a name. The following example should make the concept clearer.

Assume that your organization has a set of queues that you expect to become really large for application reasons. These queues should get a secondary extent size of 8000. You decide that the names of these queues will all start with letters QL, so a pattern matching the names of large queues is QL*. To use **runnscnf** to configure IBM MQ, you create an object of class QueuePattern. The name of that object might be LargeQueues. Set the following attributes of that object to achieve your goal:

Table 9. Example properties

Property	Value
Pattern	QL*
SecondaryExtents	8000

Partitions must be specified in increasing number sequence, that is, specify `VolumePartition2` first, then `VolumePartition3`, and so on. Volumes for all partitions must be different. The subvolume is automatically set to the subvolume of the primary partition (given by the `Subvolume` parameter).

Deleting `VolumePartition`*N* properties must be done in decreasing number sequence, starting with the highest partition number for this pattern. For example, delete `VolumePartition9`, then `VolumePartion8` and so on. Deleting in decreasing number sequence ensures that there are no gaps in the sequence of partition volumes.

For more details about partitioning, and changing existing queues, see Chapter 12, “Migrating to alter or partition queue files,” on page 73.

Class SetSignalManager

Class SetSignalManager has the properties listed in the following table.

Table 10. SetSignalManager properties

Name	Datatype	Meaning
PrimaryCPU	CPU	The CPU where the primary SetSignalManager process will be started

Table 10. SetSignalManager properties (continued)

Name	Datatype	Meaning
BackupCPU	CPU	The CPU where the backup SetSignalManager process will be started. If set to the same CPU as PrimaryCPU, SetSignalManager runs as a single instance
Priority	Number	Priority for SetSignalManager process
ProcessName	Processname	Process name of SetSignalManager. This is automatically set by the system and cannot be changed
InactivityTimeout	Number	The number of minutes after a message has been sent to a process that SetSignalManager waits before doing a cleanup. The default is 1, the maximum 3000000 (approximately 6.5 years)
CloseOnTimeout	Boolean	By default the SetSignalManager closes its FILEOPEN on a target process after InactivityTimeout minutes have passed, set to False if processes should stay open

Note:

- By default, SetSignalManager processes run as an HP NonStop process pair and do not need to run on the same CPU as the IBM MQ kernel processes. This arrangement gives scalability and robustness.
- Setting CloseOnTimeout to False can increase the memory consumption of the process.
- Increasing the value for InactivityTimeout can result in a higher peak memory consumption of the process.

Examples of using `runnscnf`

Use the examples to help you understand how to use **`runnscnf`**.

Setting maximum file size for a set of queues

You can use **`runnscnf`** to set a maximum file size for a set of queues.

Assume you have a set of potentially large queues in queue manager QM1 and you want the secondary extents be set to 8000 when creating a queue. If the names of these large queues all start with QL, a pattern for that type of name is created and the parameters for that pattern specify a secondary extent of 8000. In the example the name of the pattern is LargeQueues. You use the following set of commands:

```
/home/ssd/mqm.alexi/mq8E64/opt/mqm/bin:runnscnf QM1
runnscnf Command Interface
Version 1.10, 2017-01-18

NSCNF>class QueuePattern
CLASS set to QueuePattern
NSCNF>object LargeQueues
OBJECT set to LargeQueues
NSCNF>set Pattern QL*
NSCNF>set SecondaryExtents 8000
```

To check that the command worked, enter the following command:

```
NSCNF> list (Que*,,)
```

This command lists all properties of all objects of all classes starting with "Que":

```
-------------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             Pattern
Value:                QL*
-------------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             SecondaryExtents
Value:                8000
NSCNF>
```

Showing Guardian file names of queues

You can use **runnscnf** to discover the Guardian file names for queues.

Assume you have queues Q1 and Q2 within queue manager QM1 and want to know the Guardian file names of these queues:

```
/home/ssd/mqm.alexi/mq8E64/opt/mqm/bin:runnscnf QM1
runnscnf Command Interface
Version 1.10, 2017-01-18

NSCNF>dis Q*
-----------------------------------------------------------------------
Queue:                Q1
Guardian file:        $MQWA.MQ8W64Q0.QRIHR2AZ
-----------------------------------------------------------------------
Queue:                Q2
Guardian file:        $MQWA.MQ8W64Q0.QRILEB89
NSCNF>
```

Configuring Fault Tolerant EC

You can use **runnscnf** to configure a fault tolerant EC.

Assume you have created queue manager QM1 and want to start it in fault tolerant mode, with the primary process running in CPU 1 and the backup process running in CPU 2. The following example shows the **runnscnf** commands used to do that:

```
runnscnf QM1
runnscnf Command Interface
Version 1.12, 2017-03-08

*** Warning: Could not open ConfigManager (error = 14) - queue manager may not be started.
NSCNF>class processes
CLASS set to Processes
NSCNF>object EC
OBJECT set to EC
```

```
NSCNF>set PrimaryCPU 0
NSCNF>set BackupCPU 1
NSCNF>
runnscnf finished.
```

The configuration is completed before the queue manager is started. Then the queue manager QM1 can be started:

```
strmqm QM1
IBM MQ queue manager 'QM1' starting.
IBM MQ queue manager 'QM1' started using V8.0.0.3.
```

Use **dspmq** with –x to show the actual EC processes (primary and backup):

```
dspmq –x
QMNAME(QM1)
STATUS(Running)
    PROCESS($X4RM) PRIMARY(0,1105) INSTANCE(1)
    PROCESS($X4RM) BACKUP(1,5469) INSTANCE(2)
```

The instance numbers (1 and 2) are used internally by IBM MQ to distinguish the two instances. If the CPU of the primary EC process fails, there is an automatic takeover and the old backup becomes the new primary. So the role of an instance can change within the lifetime of the instance. The instance number does not change over the lifetime of the instance. If instance 1 fails, instance 2 becomes the new primary and a backup is started. The new backup then has instance number 1.

altmqfls

For reasons of compatibility with older versions of IBM MQ on NonStop, the **altmqfls** command is supported on the platform.

Purpose

You use the **altmqfls** command to set those attributes of an IBM MQ object to be created that are specific to IBM MQ for HPE NonStop.

Most of the parameters available for the WebSphere MQ for HP NonStop Server V5.3 version of the command are not available in IBM MQ V8. In IBM MQ for HPE NonStop V8, **altmqfls** cannot be used to alter the properties of an existing queue file, it is only used to define properties for a new queue. Settings made with **altmqfls** in IBM MQ for HPE NonStop V8 can be checked with **runnscnf**. If you set a parameter for a queue with **altmqfls**, internally an object of class QueuePattern with the name of the queue is created. The Pattern property for that object is also set to the name of the queue, so that the pattern applies only to that specific queue.

Note: The **altmqfls** command can be used in IBM MQ V8 when the queue manager is either running or not running. When **altmqfls** is used while the queue manager is running, an online migration is started. See “Online migration” on page 75 for details. When **altmqfls** is used on an existing queue while the queue manager is not running, an offline migration is started. See “Offline migration” on page 75 for details. **altmqfls** waits for the end of the migration and reports the result.

Syntax

```
altmqfls [ --qmgr QMgrName] --type ObjectName [--volume VolName] [--qsize]
(..) ObjectName
```

Parameters

ObjectName

The name of the new IBM MQ object.

--type *ObjectType*

The type of the IBM MQ object, which can only be ql or qlocal.

--qmgr *QMgrName*

Optional. The name of the queue manager to which the IBM MQ object belongs. If no queue manager name is specified, the default queue manager is used.

--volume *VolumeName*

Optional. The name of the volume where the file that is associated with the new local queue is to be created. This parameter can be specified only with the *ObjectName*, --type *ObjectType*, and --qmgr *QMgrName* parameters. It is not allowed in combination with any other parameters.

--qsize (*QPriExt,QSecExt,QMaxExt,OPriExt,OSecExt,OMaxExt*)

Optional. Properties of the file extents of the queue file associated with the local queue. You must specify at least the first three properties whenever you use this parameter. The remaining three properties have no meaning for IBM MQ for HPE NonStop V8 and are silently ignored. The size of an extent is expressed in pages.

- *QPriExt* The size of the primary extent of the queue file
- *QSecExt* The size of a secondary extent for the queue file
- *QMaxExt* The maximum number of extents for the queue file

Return Code

- 0 - Command completed normally
- 10 - Command completed but not entirely as expected
- 20 - An error occurred during processing

dspmqfls

The **dspmqfls** command is provided for compatibility with WebSphere MQ V5.3. Due to the difference in the storage of IBM MQ objects, the information displayed is of limited use in IBM MQ for HPE NonStop V8.

Purpose

Use the **dspmqfls** command to display the real file system names for all IBM MQ objects that match a specified criterion. You can use this command to identify the files associated with a particular object.

In IBM MQ for HPE NonStop V8, all object descriptions are stored in the file `AMQOBJMD` located in the Guardian installation directory (subvolume) of the specific queue manager.

Syntax

`dspmqfls [-m QMgrName] [-t ObjType] GenericObjName`

Parameters

GenericObjectName

The name of the object. The name is a string with no flag and is a required parameter. Omitting the name returns an error.

This parameter supports a wild card character * at the end of the string.

Note that you must quote expressions containing wild card characters.

--type *ObjectType*

Optional. The object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

*** or all**
All object types, which is the default.

q or queue
A queue.

ql or qlocal
A local queue.

qa or qalias
An alias queue.

qr or qremote
A remote queue.

qm or qmodel
A model queue.

qmgr A queue manager object.

prcs or process
A process.

ctlg or catalog
An object catalog.

nl or namelist
A namelist.

--qmgr *QMgrName*
Optional. The name of the queue manager to examine files for. If you omit this name, the command operates on the default queue manager. The object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

Return Code

- 0 - Command completed normally

altmqusr

Use the **altmqusr** command to create, alter, or delete an entry in the principal database of a queue manager.

Purpose

Each entry in the principal database maps an IBM MQ principal to a NonStop OS user ID.

Syntax

```
altmqusr  -m QMgrName -p principal [-u NonStopOSUserID] [-r]
```

Parameters

-m *QMgrName*

The name of the queue manager whose principal database is to be updated.

-p *principal*

The IBM MQ principal whose entry in the principal database is to be created, altered, or deleted.

-u *NonStopOSUserID*

Optional. A NonStop OS user ID or Safeguard alias. If the IBM MQ principal does not already have an entry in the principal database, a new entry is created to map the principal to the specified NonStop OS user ID, or to the NonStop OS user ID corresponding to the specified Safeguard alias. If the principal already has an entry, the entry is altered so that the principal maps to the new NonStop OS user ID.

Note that Safeguard aliases are not stored in the principal database. If you specify a Safeguard alias for this parameter, only the corresponding NonStop OS user ID is stored in the principal database.

-r Optional. The entry for the IBM MQ principal is deleted from the principal database.

Return Code

- 0 - Successful operation
- 36 - Invalid arguments supplied
- 69 - Storage not available
- 71 - Unexpected error

Examples

The following command maps the IBM MQ principal mquser1 to the NonStop OS user ID MQTEST.FRED:

```
altmqusr -m MT02 -p mquser1 -u MQTEST.FRED
```

The following command maps the IBM MQ principal mquser2 to the NonStop OS user ID for which user01 is a Safeguard alias:

```
altmqusr -m MT02 -p mquser2 -u user01
```

The following command deletes the entry for the IBM MQ principal mquser1 from the principal database:

```
altmqusr -m MT02 -p mquser1 -r
```

Related commands

"dspmqusr" - Display IBM MQ user information

dspmqusr

Use the **dspmqusr** command to display information about IBM MQ principals that have entries in the principal database of a queue manager.

Purpose

You can use **dspmqusr** to display information about a specified principal, or to display information about all the principals in the database.

For each IBM MQ principal, the command displays the following information:

- The principal itself
- The NonStop OS user ID to which the principal maps
- The user groups to which the NonStop OS user ID belongs

Syntax

```
dspmqusr   -m QMgrName [-p principal]
```

Parameters

-m *QMgrName*

The name of the queue manager whose principal database is to be queried.

-p *principal*

Optional. The IBM MQ principal whose entry in the principal database is to be queried. If you omit this parameter, the command displays information about all the principals that have entries in the principal database of the queue manager.

Return Code

- 0 - Successful operation
- 36 - Invalid arguments supplied
- 69 - Storage not available
- 71 - Unexpected error

Examples

This example shows what **dspmqusr** displays for a newly created queue manager:

```
dspmqusr -m UNM
Principal    Userid  Username              Alias GroupName          GroupType
               0.1
mqm           47.11  MQM.MANAGER              n    MQM                   a
nobody         0.0
```

This example shows what **dspmqusr** displays after additional principals have been added to the principal database using **altmqusr**:

```
dspmqusr -m UNM
Principal    Userid  Username              Alias GroupName          GroupType
               0.1
mqm           47.11  MQM.MANAGER              n    MQM                   a
mquser1      250.1   MQTEST.USER0001          n    MQTEST                a
                                                  MQM                   s
mquser2      251.1   GROUP.FRED               n    GROUP                 a
nobody         0.0
```

Principal mquser1, which maps to user ID MQTEST.USER0001, has been added. MQTEST.USER0001 is a member of two user groups, MQTEST and MQM.

Principal mquser2, which maps to user ID GROUP.FRED, has also been added. GROUP.FRED is a member of only one user group called GROUP.

Related commands

"altmqusr" on page 47 - Alter IBM MQ user information

Configuring NonStop for IBM MQ

There are some areas of HPE NonStop that you need to configure for IBM MQ.

IBM MQ for HPE NonStop V8 TMF Configuration

IBM MQ uses TMF resource managers (RMs) to coordinate transactions.

Within TMFCOM, there are two important configuration value that users of IBM MQ on NonStop need to consider:

RMOPENSPERCPU
Limits the total number of TMF resource managers (RMs) that can be opened by all processes within a single NonStop CPU.

BRANCHESPERRM
Limits the number of TMF transactions that can be exported to each TMF RM.

You must ensure that the `RMOPENPERCPU` and `BRANCHESPERRM` configuration parameters are set to appropriate values for your configuration that are sufficiently large to accommodate the anticipated TMF transaction traffic for MQ and other non-MQ uses of TMF.

The following table lists IBM MQ processes that will open TMF RMs:

Table 11. Processes that open TMF RMs

MQ process type	MQ program	Number of TMF RMs	CPU
Guardian Agent processes	amqzlga0	*n* per process	HOME CPU of qmgr
Shared Agent processes	amqzlaa0	*n* per process	HOME CPU of qmgr
Cache Manager processes	amqcache	1 per process	configurable using runnscnf
Config Manager processes	amqconfg	1 per process	HOME CPU of qmgr

Each IBM MQ agent process (amqzlga0 and amqzlaa0) open *n* RMs, where *n* is calculated as follows:

```
n = 1 + ( 1000 / BRANCHESPERRM )  rounded up to a whole number
```

Within TMFCOM, `BRANCHESPERRM` defaults to 128, so a default TMF configuration would cause each IBM MQ agent process to open 9 RMs in the queue manager's HOME CPU.

Within TMFCOM, `RMOPENPERCPU` defaults to 128.

The **ALTER BEGINTRANS** command of TMFCOM is used to change these configuration values. The default values may be sufficient for a small deployment, but users should consider increasing these values significantly for medium or large deployments.

Usage notes

1. The number of agent processes (of either type) used by each queue manager is dynamic and depends on the number of applications connected to the queue manager, as well as on the tuning parameters specified in each queue manager's `qm.ini` file.
2. Agent processes run in the HOME CPU of each queue manager, and when multiple queue managers use the same HOME CPU, their processes will share the configured value of `RMOPENPERCPU`.
3. Cache Manager processes run in the CPU configured by **runnscnf**, see "The **runnscnf** tool" on page 30.

Chapter 8. High availability (HANSQM)

One of the main features of the NonStop platform is high availability and tolerance of any kind of single failure. Therefore the implementation of IBM MQ must also be able to tolerate failures.

Most failures in a NonStop system (like disks, controllers, power, and so on) are automatically and transparently handled either by the hardware or by the operating system. However, failure of a CPU must still be handled by mechanisms within applications. These topics describe how IBM MQ for HPE NonStop V8 deals with these situations and what is different compared to IBM MQ 5.3 on NonStop.

A basic understanding of the difference between IBM MQ and an ordinary NonStop business application is helpful. An ordinary NonStop business application (especially in the Pathway context) consists of a number of context-free servers providing business logic and database access. Servers are typically single threaded, so a CPU failure will bring down a subset of these servers; all transactions currently worked on by any of the failed processes will fail and can be restarted. TMF guarantees the overall consistency of the database.

IBM MQ behaves differently to ordinary applications. To provide the complex functionality and the throughput required by applications, IBM MQ keeps a lot of fast changing information in memory. In a standard implementation on a distributed platform like Linux or UNIX this information is kept in shared memory, so that all components, even if running in different CPUs, have immediate access to that information. In a “shared everything” environment this design works well.

As NonStop servers are built as a “shared nothing” architecture, sharing information across CPUs is not as easy. NonStop provides a fast message system to transport information across CPU boundaries, but due to the operating system overhead, this is several orders of magnitude slower than direct memory access. An application using shared memory running transparently distributed in multiple NonStop CPUs would need to transport each individual change across CPU boundaries using an IPC message. This is the classic NonStop checkpointing process pair approach, as it was also implemented in IBM MQ 5.3 on NonStop.

With IBM MQ V8 there are several considerations:

- Implementing highly complex checkpointing process pairs is an extremely complex task and as such implementation errors might have severe consequences. This is high risk.
- IBM MQ V8 makes more use of shared memory than IBM MQ 5.3, so there would be a lot more checkpointing activity in turn reducing performance and costing CPU cycles in the passive instance.
- Implementing checkpointing process pairs within IBM MQ V8 would require major architectural changes and so large differences compared to the sources on other platforms. Consequently, defect repair by porting solutions from distributed platforms to NonStop would be difficult and result in more risk and later delivery of fixes.

- In case of a CPU outage IBM MQ 5.3 still aborts some transactions and loses some connections. Although the main queue server in IBM MQ 5.3 runs as a checkpointing process pair, the agents do not, and a failing agent results in broken connections.

Based on these considerations a different design is implemented on IBM MQ for HPE NonStop V8.

If a CPU fails, applications must be able to reconnect immediately to a lost connection. A few retries within a few seconds are tolerable. However, applications must see a consistent state of the queue manager.

It is not required that each change in (shared) memory is immediately transported to the CPU of the standby instance. What is important is that the standby instance can recover all information needed to continue working as if there was no failure.

Based on that principle the following architecture is implemented:

- The execution controller (EC) runs as a process pair.
- At startup, the standby instance starts as many parts of the queue manager as it can which do not require dynamically changing information.
- There is no active checkpointing of information from the primary EC to the backup EC, so that the backup does not consume any resources.
- When a takeover occurs, the new primary recovers all information it needs from persistent files, because of the use of TMF this recovered state is consistent with the application view of the database. The new primary then completes the startup phase of the queue manager so that all components are available and also starts a new backup. So fault tolerance is also restored.
- The recovery of information is done on a queue by queue basis. As soon as a queue is opened, the required information for that queue is recovered. This recovery is the same mechanism that is used when the queue is opened after startup.

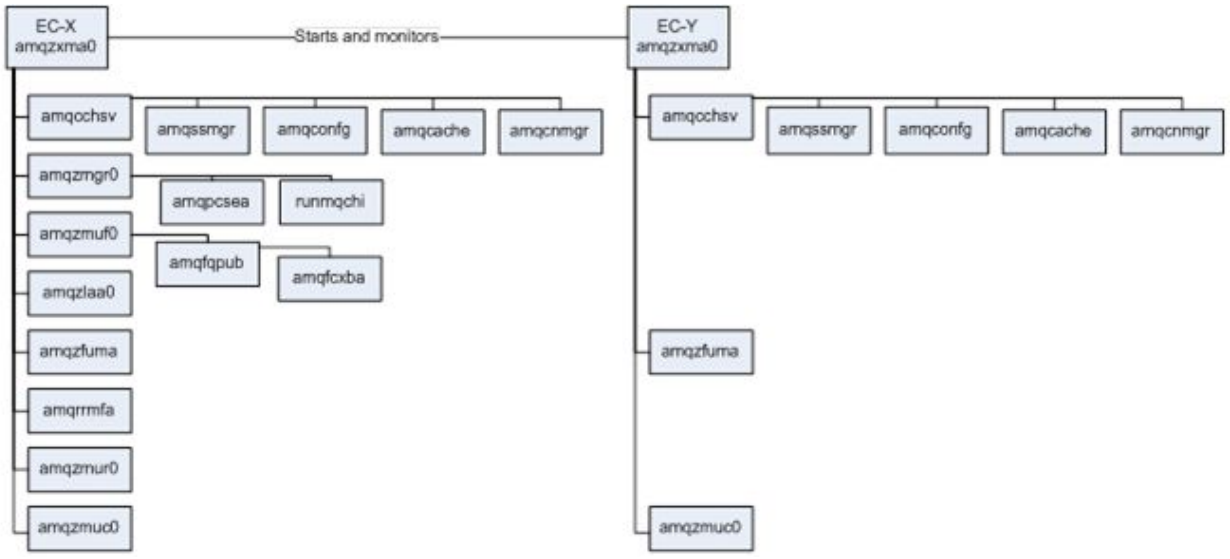
The information recovery mechanism is more expensive in terms of resources than a checkpointing backup, but it occurs only in case of a takeover. Under normal circumstances,resources are saved by not checkpointing (compared to the rare event of a CPU outage). In case of a CPU outage, applications can still continue working without human intervention.

When a queue manager is started, all standard queue manager processes are started in the EC primary CPU and a selection of queue manager processes is started in the EC backup CPU. The only process pair involved is for the EC itself, all other processes are not pairs in the NonStop sense, but are processes of different instances of two queue managers with the same name. One instance is active, and the other instance is passive (in standby mode).

The terminology used to describe that scenario is:

- Primary and backup for the EC processes (following standard NonStop terminology).
- Active and standby instance for the queue managers in the different CPUs (following standard MQ terminology).

The following diagram shows the process hierarchy after normal startup.



As the EC is an OSS process pair, the primary instance is indicated by a –X and the backup instance by a –Y.

Configuring for high availability

You can accept the default configuration for high availability, or specify some of the details yourself.

Default configuration

If no configuration is given, the following rule applies when you use **strmqm** to start a queue manager:

- The queue manager runs in fault tolerant mode, that is, with backup enabled.
- The primary EC process is started in the CPU where **strmqm** was run.
- The backup EC process is run in a different CPU, chosen by the NonStop.

Configuring CPUs and EC process name

For configuring queue manager CPUs and the EC process name, four configurable properties are available. All properties are configured using the **runnscnf** tool (see "The **runnscnf** tool" on page 30 for details). All properties are for class "Processes" and object EC. The individual properties are:

PrimaryCPU

Set the EC primary CPU. This CPU is chosen for the primary EC process when the queue manager is started, if that CPU is available at the time. All queue manager processes for the active instance run on that CPU.

If that property is not set, or set to -1, the CPU where **strmqm** is run is used.

BackupCPU

Set the EC backup CPU. This CPU is chosen for the backup EC process when the queue manager is started if that CPU is available at that time. All queue manager processes for the backup instance run on that CPU.

If that property is not set, or set to -1, any available, and allowed, CPU is used.

AllowedCPUs

A comma-separated list of CPUs. If this property is configured, queue manager processes (active and standby) are only be started in CPUs which are in this list. If configured, and >=0, the PrimaryCPU and the BackupCPU are implicitly added to that list. By using the AllowedCPUs property, you can restrict CPU usage for IBM MQ to a subset of all CPUs in the system.

ProcessName

The EC process name in the usual Guardian syntax ($...). If a process with that name is already running on the system when **strmqm** is executed, **strmqm** fails with an error message, and the queue manager is not started. If this property is not configured, a process name is chosen by the operating system. The process name chosen by the system is displayed by the following command:

```
dspmq –x
```

Disabling high availability

You can disable high availability for IBM MQ by setting the PrimaryCPU and BackupCPU to the same value in the configuration, even if both values are -1 (any CPU).

You can configure a non-fault-tolerant queue manager to run in a specific CPU by specifying:

```
PrimaryCPU=BackupCPU=CPU_to_run_queue_manager_in
```

You can configure a non-fault-tolerant queue manager to run in any available and allowed CPU by specifying:

```
PrimaryCPU=BackupCPU=-1
```

There is a difference between setting primary and backup CPU both to any CPU (-1) and not configuring primary and backup CPU. In the first case, a non-fault tolerant configuration is able to run in any allowed CPU, in the second case a fault tolerant configuration is able to run in any allowed CPU.

CPU failures

IBM MQ behaves differently according to which CPU fails.

Failure of primary MQ CPU

If the CPU running the primary EC (and all processes of the active MQ instances) fails, the following events occur:

- All processes (except set signal managers) of the active instance of that queue manager exit, even if they run on a different CPU.
- All non-persistent messages of that queue manager at that point in time are lost.
- All transactions started in that CPU are aborted by the system. This includes all transactions implicitly started by the queue manager, and all transactions started by application processes running in the failed CPU.

- A takeover sequence is initiated. The backup EC instance automatically becomes the new active instance, completes its initialization work and starts the remaining queue manager processes.
- The now primary EC starts a backup in an available CPU (if there is any).
- After the now active instance is completely up and running, applications can reconnect. Applications trying to reconnect before that point in time receive an error, but can retry after a short delay.

The time needed for that series of events depends on the type, size, workload, and so on, of your system. Typical times are between 1.5 and 10 seconds.

Failure of back up CPU

If the CPU running the backup EC (and all processes of the standby MQ instances) fails, the following events occur:

- All processes (except set signal managers of the standby instance of that queue manager) exit, even if they are currently running on a different CPU.
- All transactions started in that CPU are aborted by the system.
- The primary EC gets notification of the failure of its backup and starts a new backup in another available and allowed CPU (if possible). This instance initiates the standby queue manager.

Failure of other CPUs hosting IBM MQ processes

If a CPU fails where cache managers, or other IBM MQ processes, run, and which is not the main queue manager CPU, the failing processes are automatically restarted in an available CPU by the queue manager. If a cache manager was running on a failed CPU, all non-persistent messages for queues served by this cache manager are lost.

Applications must be prepared to get error messages when trying to retrieve messages from a failed cache manager. The application receives as many of these errors as messages were lost. Normal processing then continues.

Reload of a CPU

If a CPU is reloaded and the queue manager is configured for fault tolerance, but is currently running in non-fault-tolerant mode due to available/allowed CPUs, IBM MQ checks if a new backup in the reloaded CPU can be started (that is, if the reloaded CPU is an allowed CPU). If that is the case, a backup EC instance and a standby queue manager are started in the reloaded CPU.

Chapter 9. User name mapping

The implementation of IBM MQ on all platforms uses the concept of a principal to identify users.

A principal string can only be 12 characters long and is case-sensitive. The principal name is stored in the message data header to identify the user. The principal name can be passed around to other queue manager instances.

The HPE NonStop platform defines user names to be of the form *GROUP.USER*, where the maximum length can be 17 characters. In the general case, such names are too long and must be mapped to a principal to be handled internally.

The user name mapping feature of IBM MQ V8 allows the MQ administrator to set up mappings between principals (that is IBM MQ format user name) and NonStop user names. The mapping is one-to-one and unique; a principal can be mapped to one username only, and vice versa.

The feature provides two tools, called **altmqusr** and **dspmqusr**, for configuring these mappings. See “altmqusr” on page 47 and “dspmqusr” on page 48 for details.

User name mappings are specific to the queue manager for which they are created; each queue manager has its own mapping database. This database is allocated and filled with initial values when the queue manager is created using the **crtmqm** command.

The predefined, initial principals are:

mqm Mapped to the user executing the **crtmqm** command.

nobody
Mapped to the (numeric) user 0,0, which might or might not exist.

(12 space characters)
Mapped to the (numeric) user 0,1, which might or might not exist.

The principal nobody and (12 space characters) are not usable for applications, and are used as placeholders if a user is rejected and/or not authorized.

User identification

During application and IBM MQ connect processing, the queue manager obtains from NonStop OS the user ID under which the application is running.

The queue manager then queries the principal database to determine the corresponding principal, and passes this principal to the OAM for all subsequent authority checks on behalf of the application. These authority checks include checking whether the principal has the authority to connect to the queue manager. If the user ID has no entry in the principal database, the queue manager assigns the principal `nobody` to the user ID.

When an application sends a message, the queue manager sets the UserIdentifier field in the message descriptor to the principal associated with the application. The principal therefore travels with the message as the means of identifying the user who sent the message.

In general, all NonStop user names must get mapped to a principal in order to run applications. The mapping for the installation owner and the principal mqm is created automatically; all other mappings must be created manually.

The installation owner user ID

IBM MQ requires that a user group named MQM is defined. Any user in that group is allowed to install the base product. All files created during installation show this user ID as the file owner.

All users in the MQM group are allowed to create a queue manager instance; it is not required to use the user ID used for the initial installation. All dynamic files (queue files, configuration files and so on) are still owned by the user ID used for the installation.

However, queue managers created by two different MQM group members will differ in the initial mapping of the principal mqm; this entry will show the NonStop user name used to create this queue manager.

In this sense, the term "installation owner" refers to the creator of a particular queue manager and the principal mqm found in its mapping database.

This principal and its associated user name has full control on this queue manager, including the permission to run applications that connect to it.

User names in the MQM group

Users that are members of the MQM group are privileged users.

Users belonging to the MQM group can carry out all administrative work on all queue managers within the installation. This work includes:

- Create any queue manager (**crtmqm**)
- Start any queue manager (**strmqm**)
- Stop any queue manager (**endmqm**)
- Delete any queue manager (dltmqm)
- Configure queue manager attributes (**runmqsc**, **setmqaut**, altmqusr and so on)

However, not all users are allowed to run applications against all existing queue managers; only the "installation owner" can run applications initially.

To allow application processing by other MQM group members, you must create a mapping entry for each user. The mapping is created using the **altmqusr** command. Such a mapping is enough to allow access; no explicit authorization via the **setmqaut** command is required.

User names not in the MQM Group

All users outside the MQM group must be granted access to a queue manager explicitly.

At least three operations must be executed:

- Create a Principal by using the **altmqusr** command
- Allow access (connect authority) to the queue manager by using the **setmqaut** command
- Allow access (MQI operations) to entities of the queue manager by using the **setmqaut** command

For example, use the following commands to allow principal user1 access to the default queue manager and the local queue Q3:

```
MQ8i517x >altmqusr -p user1 -u dummy.user0001

MQ8i517x >setmqaut -p user1 -t qmgr +connect
The setmqaut command completed successfully.

MQ8i517x >setmqaut -p user1 -t q -n Q3 +put +get
The setmqaut command completed successfully.
```

User names for channels

When using channels between queue managers (server connects or client connects) a valid principal name must be established during connect processing.

The final principal name is created by IBM MQ based on configuration options for channels (attribute MCAUSER) and/or the CHLAUTH rules (attributes CLNTUSER and MCAUSER).

The final principal name is validated against the principal database and must be found in order to get a connection established. If the principal maps to a NonStop user outside the MQM group, the following rules must be met in addition:

- + connect authority for the queue manager
- +get +put (and so on) for any entities like queues

User names in IBM MQ security exits

The IBM MQ OAM security exits present principal names as part of the exit interface data structures. As such, they are limited to the size of a principal (12 characters).

The exact content of several fields in the data structures depend on the User ID of the command executed, and the content of the principal database.

For example, if the **dspmqaut** command is executed with the installation owner ID, the content is:

```
UserID:             [mqm         ]
EffectiveUserID:    [mqm         ]
UserIdentifier:     [mqm         ]
```

The values are the principal name ‘mqm’ for the installation owner

If the **dspmqaut** command is executed with another member of the MQM group (mqm.usr100) without principal mapping, the content is:

```
UserID:             [MQM.USR100  ]
EffectiveUserID:    [MQM.HUFF    ]
UserIdentifier:     [MQM.USR100  ]
```

The values are the native user names on the NonStop OS.

If the **dspmqaut** command is executed with another member of the MQM group (mqm.usr100) with an existing principal mapping, the content is:

```
UserID:             [user-100  ]
EffectiveUserID:    [MQM.HUFF    ]
UserIdentifier:     [user-100  ]
```

The values are either the mapped principal (for user mqm.usr100) or the native user name for the IBM MQ internal process.

Chapter 10. MQGET SET SIGNAL

The MQGMO_SET_SIGNAL option allows an application to request a notification from the queue manager when a message is available to be read from a queue.

The notification is sent by the queue manager to the application in the form of an MQIPC message on the application $RECEIVE queue.

The MQGMO_SET_SIGNAL option can be used against multiple queues concurrently and, when used this way, an application can wait for messages to arrive on many queues at the same time.

This function is offered by the **amqssmgr** process class, which is a proxy process between the calling application and the running queue manager. To guarantee high availability, this proxy is designed to run as a NonStop process pair. The proxy notifies the application of any queue manager events that might relate to MQGET calls with active MQGMO_SET_SIGNAL.

Using MQGET SET_SIGNAL

To use MQGET SET_SIGNAL, the application calls MQGET with a special option set in the MQGMO structure

The option is set as follows:

```
MQGMO gmo    = {MQGMO_DEFAULT}; /* get message options */
gmo.Options |= MQGMO_SET_SIGNAL;

MQGET(Hcon,          /* connection handle   */
        Hobj,        /* object handle       */
        &md,         /* message descriptor  */
        &gmo,        /* get message options */
        buflen,      /* buffer length       */
        buffer,      /* message buffer      */
        &messlen,    /* message length      */
        &CompCode,   /* completion code     */
        &Reason);    /* reason code         */

If ( (MQCC_OK == CompCode)
      && (MQRC_NONE == REASON) )
{
  /* A message was already on the target queue and delivered   */
  /* to the application.                                       */
} else if ( (MQCC_WARNING == CompCode)
            && (MQRC_SIGNAL_REQUEST_ACCEPTED == Reason) )
{
  /* No message was available on the target queue and the      */
  /* application will receive a notification on its $RECEIVE.  */
} else if ( (MQCC_FAILED == CompCode)
            && (MQRC_SIGNAL_OUTSTANDING == Reason) )
{
  /* There is already a SET_SIGNAL request outstanding for     */
  /* the used MQ Object Handle                                 */
} else
{
  /* Warning or Error outside of MQGET with SET_SIGNAL scope   */
  ...
}
```

MQGET fails with reason MQRC_OPTIONS_ERROR if both MQGMO_SET_SIGNAL and MQGMO_WAIT are set.

MQGET fails with reason MQRC_SIGNAL_OUTSTANDING if there still is an unserviced MQGET with SET_SIGNAL call for the supplied IBM MQ Object Handle.

Message format

After the MQGET call returns with Completion Code MQCC_WARNING and Reason Code MQRC_SIGNAL_REQUEST_ACCEPTED, an IBM MQ `amqssmgr` process sends a message on the calling application $RECEIVE to notify it on events corresponding to the message descriptor used in its MQGET call.

This message is of the type MQIPC (defined in cmqc.h) and is always 10 Bytes in size, independent of the architecture (Itanium/x86, 32bit/64bit):

```
typedef struct tagMQIPC {
        short   MsgCode;
        MQLONG  ApplTag;
        MQLONG  Status;
    } MQIPC;
```

The following table gives the meaning of the fields of the structure:

Table 12. structure field meaning

Field	Meaning
MsgCode	This field always has the value TRIGGER_RESPONSE defined in `cmqc.h`.
ApplTag	This field has same value as `gmo.Signal1` of the issued MQGET call, or 0 if the default value was used. This can be used to identify specific MQGET calls.
ApplTag	This field gives information about the event causing the message to be sent to the application.

The status field can have the values given in the following table:

Table 13. Status field meanings

Value	Meaning
MQRC_NONE	A message conforming to the message descriptor arrived at the target queue and is available to be fetched. Since multiple MQGET calls can access the same queue with the same message descriptor, it might happen that the message that initiated the MQIPC message is no longer on the queue when the application tries to retrieve it.
MQRC_NO_MSG_AVAILABLE	This status occurs if `gmo.WaitInterval` is not equal to MQWI_UNLIMITED and no message conforming with the issued message descriptor arrived within the specified interval.
MQRC_GET_INHIBITED	MQGET calls have been inhibited for the queue during an outstanding MQGET call with set SET_SIGNAL option.

Table 13. Status field meanings (continued)

Value	Meaning
MQRC_Q_MGR_QUIESCING	The queue manager is in the process of being gracefully shut down while the MQGET call with SET_SIGNAL is still outstanding.
MQRC_Q_MGR_STOPPING	The queue manager is shutting down while MQGET call with SET_SIGNAL is still outstanding.
MQRC_CONNECTION_BROKEN	The `amqssmgr` process associated with the MQGET call lost its connection to the queue manager, or its primary process died. This might happen because of system failures, like CPU-down events or IBM MQ internal errors. This status requires the application to at least reissue the MQGET call.

Active Checkpointing

Applications can wait indefinitely on their $RECEIVE for a notification from IBM MQ.

If the `amqssmgr` process dies unexpectedly, the application might never receive the notification. To prevent this from happening, `amqssmgr` is designed to run as a NonStop process pair and so not let user applications hanging.

On process start, the primary `amqssmgr` process spawns its backup process and checkpoints each incoming MQGET with SET_SIGNAL request to the backup.

Should a disastrous event occur, such as death of the Execution Controller or of the `amqssmgr` process itself (for example, due to CPU-down or IBM MQ internal errors), the backup process takes over, spawns a new backup process, replicates all outstanding MQGET calls to it, and then send a MQIPC message for each outstanding MQGET call with the Status field set to MQRC_CONNECTION_BROKEN. As such it can happen that a single application might receive multiple messages with the same content.

Independent of the cause of failure, if the queue manager is in a state in which MQCONN calls would succeed, new MQGET calls with SET_SIGNAL are accepted and handled.

Chapter 11. Migrating between IBM MQ versions

You can migrate queue managers and queue data between IBM MQ versions.

You can migrate queue managers from WebSphere MQ V5.3 to IBM MQ V8, from IBM MQ V8 to IBM MQ V8. You can migrate queue data from IBM MQ V8 to WebSphere MQ V5.3.

There are four OSS utilities provided for migration:

exportmqm
: Exports all the information and data from an IBM MQ V8 queue manager.

mig/exportmqm
: Exports all the information and data from a WebSphere MQ V5.3 queue manager.

importmqm
: Creates and configures an IBM MQ V8 queue manager.

mig/importmqm
: Imports queue data from an IBM MQ V8 queue manager back into a WebSphere MQ V5.3 queue manager.

Exporting a WebSphere MQ V5.3 queue manager by using `exportmqm`

Use `mig/exportmqm` to export a WebSphere MQ V5.3 queue manager.

The **exportmqm** utility exports all queue manager non-SYSTEM.* objects and their configurations, Channel Synchronization records, user and object security settings, SSL files (if present) and, if selected, all the messages on the application local queues.

The **exportmqm** utility has the following syntax:

```
exportmqm -m QueueManagerName
```

For example:

```
exportmqm -m QM1
```

Where QM1 is the name of the WebSphere MQ V5.3 queue manager being migrated.

The **exportmqm** utility uses four files located in the *install_path*`/opt/mqm/bin/mig` directory of the IBM MQ V8 installation. If there is no IBM MQ installation on the system where the queue manager is being exported from, these files must be copied to that system and the directory containing the files must be added to the PATH environment variable. If it is not present in PATH, the directory should be added to it, for example, `export PATH=$PATH:`*dirname* where *dirname* is the full path to the directory where these files are located.

exportmqm can be run on any H or J-Series system and with any supported version of WebSphere MQ V5.3.

To run **exportmqm**, you must source the IBM MQ `wmqprofile` file. This is because **exportmqm** requires that the `MQNSKOPTPATH`, `MQNSKVARPATH`, and `_RLD_LIB_PATH` environment variables are set.

The WebSphere MQ V5.3 queue manager must be running and its command server must also be running. The queue manager must be in a quiet state. This means that no application queues (that is, non SYSTEM.* queues) can be open, and all channels must be either inactive or STOPPED. The **exportmqm** utility checks for these conditions and does not continue if any of these conditions are not met.

The **exportmqm** utility also verifies that it has access to the `\tmp` directory on the system, and that it can run the required commands (for example, **mkdir, tar**). If any of the required commands cannot be run, **exportmqm** stops.

All checks and steps performed by **exportmqm** are recorded in a log file that is located in the local directory.

You are asked if you want to export messages that are on their application local queues. If you answer YES at the prompt, all messages in each queue are loaded into an individual file whose name contains the name of the queue.

As **exportmqm** is running, details of what it is doing are displayed. All the steps are also recorded in the log file.

exportmqm creates a compressed tarball file and a md5sum file in the directory from which it was run, for example `MQ5EXPORT.MQ1.20170329-083048-1778385097.tar.Z` and `MQ5EXPORT.MQ1.20170329-083048-1778385097.tar.Z.md5` . Copy these two files to the location where you plan to run the import utility. This location could be on another NonStop system or on the local system, depending on where the IBM MQ V8 installation is.

Exporting an IBM MQ V8 queue manager by using `exportmqm`

Use **exportmqm** to export an IBM MQ V8 queue manager.

The **exportmqm** utility exports all queue manager non-SYSTEM.* objects and their configurations, Channel Synchronization records, user and object security settings, SSL files (if present) and, if selected, all the messages on the application local queues.

The **exportmqm** utility has the following syntax:

```
exportmqm -m QueueManagerName
```

For example:

```
exportmqm -m QM1
```

Where QM1 is the name of the IBM MQ V8 queue manager being migrated.

The **exportmqm** utility uses four files located in the *install_path*`/opt/mqm/bin` directory of the IBM MQ V8 installation.

exportmqm can be run on any J or L-Series system and with any supported version of IBM MQ V8.

To run **exportmqm**, no additional changes have to be made to the environment, except that you must source the IBM MQ `wmqprofile` file. This is because **exportmqm** requires that the `MQINST` environment variable is set.

The IBM MQ V8 queue manager must be running and its command server must also be running. The queue manager must be in a quiet state. This means that no application queues (that is, non SYSTEM.* queues) can be open, and all channels must be either inactive or STOPPED. The **exportmqm** utility checks for these conditions and does not continue if any of these conditions are not met.

The **exportmqm** utility also verifies that it has access to the `\tmp` directory on the system, and that it can run the required commands (for example, **mkdir, tar**). If any of the required commands cannot be run, **exportmqm** stops.

All checks and steps performed by **exportmqm** are recorded in a log file that is located in the local directory.

You are asked if you want to export messages that are on their application local queues. If you answer YES at the prompt, all messages in each queue are loaded into an individual file whose name contains the name of the queue.

As **exportmqm** is running, details of what it is doing are displayed. All the steps are also recorded in the log file.

exportmqm creates a compressed tarball file and a md5sum file in the directory from which it was run, for example `MQ8EXPORT.MQ1.20170329-083048-1778385097.tar.Z` and `MQ8EXPORT.MQ1.20170329-083048-1778385097.tar.Z.md5` . Copy these two files to the location where you plan to run the import utility. This location could be on another NonStop system or on the local system, depending on where the IBM MQ V8 installation is.

Importing a WebSphere MQ V5.3 or an IBM MQ V8 queue manager by using `importmqm`

Use `importmqm` to import configuration and queue data exported from a WebSphere MQ V5.3 or IBM MQ V8 queue manager using `mig/exportmqm` or `exportmqm`.

The `importmqm` utility has the following syntax:

```
importmqm -f ExportedFileName [-m QueueManagerName]
```

Where *ExportedFileName* is the name of the file that you are importing from, and -m *QueueManagerName* optionally specifies a new name for the imported queue manager.

For example:

```
importmqm -f MQ5EXPORT.MQ1.20170329-083048-1778385097.tar.Z  -m newQM
```

`importmqm` imports object configuration, channel synchronization information, security settings, and, if selected, the messages on the application local queues. The

utility does not require any environment variables and must not be run in an environment where WebSphere MQ V5.3 environment variables are set.

Because `importmqm` uncompresses and untars the input file created by the `exportmqm` utility, and creates a directory of the same name, it must not be run in the same directory as the `exportmqm` utility.

The `importmqm` utility verifies that it has access to the `\tmp` directory on the system, and also that it can run the commands that are needed (for example, **mkdir, tar**). If any of these required commands cannot be run, the utility stops.

All checks and steps performed by the `importmqm` utility are recorded in a log file which is located in the local directory.

The `importmqm` utility uncompresses and untars the input tarball into a working directory located in the same directory where the utility is run.

The utility then creates and starts a queue manager with the same name as the source queue manager, or with the queue manager name specified by the -m argument. If the queue manager is imported under a new name, the Channel Synchronization records are not migrated, and remote channels might have to be reset.

You are asked if you want to import messages that were on the application local queues of the source queue manager. If you answer YES at the prompt, all messages in each queue file are loaded into the queue whose name is determined from the name of the file containing the messages in the working directory.

The utility creates all the non SYSTEM* objects that were in the source queue manager, creates principals in the principal database, sets authorizations for users and objects, loads the channel synchronization data records, and, if present, copies in the SSL files to the queue manager SSL directory. If a User Name in the principal database on the source queue manager is not also a user on the system where the import utility is being run, a message is displayed, and no attempt is made to add that user to the principal database. The utility continues to run.

As the utility `importmqm` runs, details of what it is doing are displayed. All the steps are also recorded in the log file.

After the `importmqm` utility has finished, the log file that is located in the same directory can be reviewed.

While cluster attributes for objects are preserved, the Full Repository status of the queue manager itself is not and has to be manually preserved by using the **runmqsc ALTER QMGR REPOS()** command.

Cluster channels are not imported either, and have to be defined manually to integrate an imported queue manager into a cluster. To speed up this process, **importmqm** generates a file named *queue_manager*`_cluster_channel_definitions.mqsc` in the working directory it is called from. This file contains the cluster channel definitions of the exported queue manager and can be used as input for **runmqsc**.

Importing IBM MQ V8 data back into WebSphere MQ V5.3 using `mig/importmqm`

Use **mig/importmqm** to import queue data back into a WebSphere MQ V5.3 queue manager.

You should ensure that back ups are available before you use the utility.

The `importmqm` utility imports all the messages on the application local queues.

The `importmqm` utility has the following syntax:

```
importmqm -f ExportedFileName -m QueueManagerName
```

Where *ExportedFileName* is the name of the file that you are importing from, and -m *QueueManagerName* specifies a name for queue manager you are loading the data to.

For example:

```
importmqm -f MQ8EXPORT.MQ1.20170329-083048-1778385097.tar.Z  -m newQM
```

The **importmqm** utility uses four files located in the `install_path/opt/mqm/bin/mig` directory of the IBM MQ V8 installation. If there is no IBM MQ installation on the system where the queue manager is being exported from, these files must be copied to that system and the directory containing the files must be added to the PATH environment variable. If it is not present in PATH, the directory should be added to it, for example, `export PATH=$PATH:`*dirname* where *dirname* is the full path to the directory where these files are located.

importmqm can be run on any H or J-Series system and with any supported version of WebSphere MQ V5.3.

To run **importmqm**, you must source the IBM MQ `wmqprofile` file. This is because `importmqm` requires that the `MQNSKOPTPATH`, `MQNSKVARPATH`, and `_RLD_LIB_PATH` environment variables are set.

The WebSphere MQ V5.3 queue manager must be running and its command server must also be running. The queue manager must be in a quiet state. This means that no application queues (that is, non SYSTEM.* queues) can be open, and all channels must be either inactive or STOPPED. The **importmqm** utility checks for these conditions and does not continue if any of these conditions are not met.

The **importmqm** utility also verifies that it has access to the directory on the system, and that it can run the required commands (for example, `mkdir, tar`). If any of the required commands cannot be run, **importmqm** stops.

All checks and steps performed by **importmqm** are recorded in a log file that is located in the local directory.

You are asked if you want to import messages that are on their application local queues. If you answer YES at the prompt, all messages in each queue are loaded from the archive to be imported.

As **importmqm** is running, details of what it is doing are displayed. All the steps are also recorded in the log file.

Since **importmqm** only imports queue data, channels may have to be reset.

Queue data can only be migrated back to queue managers of the same name.

Chapter 12. Migrating to alter or partition queue files

You can use **runnsncf MIGRATE** commands to alter the attributes of a queue file or partition a queue file.

On IBM MQ for HPE NonStop V8, queues are represented by Enscribe key sequenced files. For each queue there is exactly one file representing the queue and containing persistent messages.

You might need to change attributes of a queue file. If, for example, a disk is too busy, you might want to move a heavily-used queue file to a different disk. Because of disk space shortage or performance issues, you might also want to use partitioned queue files. Moving existing queue files to a different volume, partitioning, or de-partitioning existing queue files is called “queue file migration”. Queue file migration can be done while the queue manager is up and running and the queue is in use (online migration) or while the queue manager is stopped (offline migration). The migration is controlled either by **runnscnf** or, as an alternative, by **altmqfls** (if only one single queue is to be migrated).

Queue file migration creates a new physical file for the queue, and copies the data from the old file to the new file.

Queue file migration overview

When you use **runnscnf** to alter or partition a queue file, you define objects of type “pattern”.

Each named pattern has a property named `Pattern`. The value of the `Pattern` property is a string. The string can include wildcard characters ‘*’ and ‘?’ (‘*’ matches any substring , ‘?’ matches a single character). For each pattern, queue file attributes such as subvolume, partition scheme, extent sizes can be defined. The pattern mechanism gives a set of rules for queue file attributes. Whenever a new queue is created, the patterns are checked against the queue name. If the queue name matches a pattern, the queue file attributes are set according the configuration for that pattern. So the rule set defined by using **runnscnf** with `Pattern` objects is immediately applied to all new queues created with a matching name.

The queue file migration feature also enables the application of the pattern-based rules to existing queues in a queue manager. If there is a difference between the existing file attributes and the file attributes as they should be according to the rules, the queue is a candidate for migration. The queue file migration is a complex process involving creating a new physical queue file (potentially including several partitions), moving queue messages from the old to the new file, and keeping track of the status. The old queue file is removed when all messages have been moved and the old file is not in use anymore.

Complete the following steps when planning to change the attributes of existing queue files:

1. Set up the patterns and attributes for the change using **runnscnf**.
2. Calculate the set of queues that are now candidates for migration. This is done by using the PREPARE MIGRATE command in **runnscnf**. The **PREPARE MIGRATE** command requires a pattern to potentially limit the set of queues to consider. If you want a list of all queues as candidates for migration, use the * pattern:

   ```
   PREPARE MIGRATE "*"
   ```

   The **PREPARE MIGRATE** command calculates the set of migration candidates for a subsequent **START MIGRATE** command within the current **runnscnf** session.
3. Determine the set of queues that you want to migrate within one migration step. If the list of candidate queues that results from the **PREPARE MIGRATE** command is long, you might consider using a more restrictive pattern in **PREPARE MIGRATE** to get a smaller set of candidates.
4. Start the migration for the currently selected set by issuing the **START MIGRATE** command. For each element of the set calculated by the previous **PREPARE MIGRATE** command, START MIGRATE starts a migration process and sets several properties (use the **runnscnf STATUS MIGRATE** command to view the properties).
5. Optionally monitor migration progress using the **runnscnf STATUS MIGRATE** command.

You can use **runnscnf** to migrate while the a queue manager is running (online migration), or while it is stopped (offline migration).

You can alternatively use the **altmqfls** command to migrate queue files, but **altmqfls** has the following limitations:

- **altmqfls** only processes a single queue file at a time.
- **altmqfls** cannot configure partitioned queues.
- **altmqfls** waits until the migration process has completed before returning to TACL or the OSS shell.

When the queue file migration is finished, and the old queue file is not used by the queue manager anymore, the old queue file is deleted. The parameters indicating a migration that are visible using **runnscnf** are also be deleted. So, if **STATUS MIGRATE** does not display anything, this means that all previously started migrations have been successfully finished.

Changing the partitioning information of queue files is completed in the same way as any other queue file migration.

Using PREPARE MIGRATE

You use the **runnscnf PREPARE MIGRATE** command to create a list of queue files to migrate.

About this task

The PREPARE MIGRATE command completes the following tasks:

- Walks through the list of all queues within the queue manager that is currently selected by **runnscnf**.

- For each queue matching the pattern given in the **PREPARE MIGRATE** command, it calculates the attributes the queue file should have according to the **runnscnf** configuration. These parameters are:
  - Location (subvolume) for the queue file
  - Primary extent size
  - Secondary extent size
  - Number and location of partitions (if any)

  If there is a mismatch between the configured values and the real values (for example if the configuration was changed after the queue was created), the queue is put in the list of migration candidates.
- The list of migration candidates, together with information about the reason why this queue is a candidate, is displayed.

The **PREPARE MIGRATE** command does not change any persistent state in the system. Each new **PREPARE MIGRATE** command calculates everything from scratch. So, if the list printed by **PREPARE MIGRATE** has too many entries, or entries you do not want to migrate now, you can reissue the command using a different pattern so that the result set matches your expectations.

The **runnscnf** session remembers the result of the **PREPARE MIGRATE** command for a subsequent **START MIGRATE** command. **START MIGRATE** always works on the set of queues calculated by **PREPARE MIGRATE**. This separation of the **PREPARE MIGRATE** and **START MIGRATE** commands has the following advantages:

- Multiple migrations can be initiated and started in one step. If required, multiple queues can be migrated with just two commands.
- You can control of the number of migration processes running in parallel, so that the I/O load does not cause any system problems.

Procedure

To create the migrate list, enter the following command:

```
PREPARE MIGRATE QueueNamePattern
```

Offline migration

Queue files can be migrated while the queue manager is in the ended state. This is known as offline migration.

runnscnf starts a migration process for each queue to be migrated. These migration processes run in the background, even if **runnscnf** is ended. The status of active migrations can be checked using the **STATUS MIGRATE** command in **runnscnf**. This command can be issued from any **runnscnf** session for this queue manager.

If a queue manager is started while migrations are going on, the migration process is stopped and restarted as an online migration by the queue manager. This is reflected in the “Time restarted” information in the **STATUS MIGRATE** command output.

Online migration

Queue files can be migrated while the queue manager is running. This is known as online migration.

Online migrations are done in exactly the same way as offline migrations. During the migration the queue manager knows that, temporarily, the queue being migrated is represented by two physical queue files, potentially with a different number of partitions. All new messages go into the new queue file. Old messages are looked for in the old file first, if not found (because they have already been moved by the migration process) the queue manager looks for the message in the new queue file. Since all queue files are TMF protected, no inconsistencies can arise.

The I/O load for a queue being migrated is significantly higher than under normal circumstances. However, this should only be a concern if the queue to be migrated is very large (more than 10000 messages, or a large number of large messages). Queues with less than 10000 messages of size less than 25k are typically migrated within a few seconds.

If the queue manager is ended while online migrations are in process, the migrations are interrupted. When the queue manager is started again, all migrations continue automatically.

While using online migration, the `MQOPEN` options `MQOO_SET` and `MQOO_BROWSE` must be allowed against the migrating queues.

Migration example

The following example shows a complete **runnscnf** session for an online queue file migration.

In the first step a named pattern object “LargeQueues” is created. All queues matching the corresponding pattern “QueueL*” will be located on subvolume $MQAS.IBMMQM, and the primary extent size for these queues will be 600:

```
$ runnscnf QMb
runnscnf Command Interface
Version 1.12, 2017-03-08
NSCNF>class queuepattern
CLASS set to QueuePattern
NSCNF> object LargeQueues
OBJECT set to LargeQueues
NSCNF> set pattern QueueL*
Property Pattern set to QueueL*
NSCNF> set subvolume '$MQAS.IBMMQM'
Property Subvolume set to $MQAS.IBMMQM
NSCNF>set PrimaryExtents 600
Property PrimaryExtents set to 600
```

In the next step, all queues matching the pattern QueueL* are shown using the DIS command (so you can see which queues will be migrated):

```
NSCNF>dis QueueL*
Queue:                QueueL11
Guardian file:        $MQAS.MQ8A1AQ1.Q2ZCQVCB
------------------------------------------------------------------------
Queue:                QueueL12
Guardian file:        $MQAS.MQ8A1AQ1.Q2ZEUML4
```

```
------------------------------------------------------------------------
Queue:                QueueL21
Guardian file:        $MQAS.MQ8A1AQ1.Q2ZH5FEF
```

There are currently three queues matching the pattern. None of the three queues currently resides on the correct subvolume, $MQAS.IBMMQ. However, at this point only queues QueueL11 and QueueL12 are required to be migrated, so a pattern limiting the selection to these two queues is specified for the **PREPARE MIGRATE** command:

```
NSCNF>prepare migrate QueueL1*
Queues to migrate:
QueueL11
$MQAS.MQ8A1AQ1.Q2ZCQVCB               ->$MQAS.IBMMQM
QueueL12
$MQAS.MQ8A1AQ1.Q2ZEUML4               ->$MQAS.IBMMQM
NSCNF>
```

Based on the pattern QueueL1* **runnscnf** shows the required two migration candidates. Commands **START MIGRATE** and **STATUS MIGRATE** are used to start the actual migration and show the status:

```
NSCNF>start migrate
NSCNF>status migrate
------------------------------------------------------------
Migration of queue "QueueL11":
$MQAS.MQ8A1AQ1.Q2ZCQVCB -> $MQAS.IBMMQM.Q208GUIY
Started at:        2017-09-20 13:44:23.587
Migration process: \CS4.$X10F1:6411812041
Status:            Initializing
------------------------------------------------------------
Migration of queue "QueueL12":
$MQAS.MQ8A1AQ1.Q2ZEUML4 -> $MQAS.IBMMQM.Q208HMLP
Started at:        2017-09-20 13:44:23.624
Migration process: \CS4.$X10F2:6411812297
Status:            Initializing...
```

After the migration is finished, the queues are correctly at the new location:

```
NSCNF>dis QueueL*
------------------------------------------------------------------------
Queue:                QueueL11
Guardian file:        $MQAS.IBMMQM.Q208GUIY
------------------------------------------------------------------------
Queue:                QueueL12
Guardian file:        $MQAS.IBMMQM.Q208HMLP
------------------------------------------------------------------------
Queue:                QueueL21
Guardian file:        $MQAS.MQ8A1AQ1.Q2ZH5FEF
NSCNF>
```

Partitioning

You can use queue file migration to partition a queue file.

There are many reasons why you might want to partition queue files. For large queues, disk space could be an issue. For very busy queues, distributing a queue over several disks can distribute CPU load to multiple DP2 disk processes, and can also make use of larger disk cache for multiple drives.

With IBM MQ for HPE NonStop V8.0.1 you only have to consider the disks you want the partitions to reside on. You then configure a partitioning scheme together with a queue name pattern in the same way as you would do for locating a queue on a different subvolume (see Chapter 7, “Configuring IBM MQ,” on page 27 for

details). When you create a queue where the name matches the pattern, IBM MQ will create the queue file with the number of partitions and the location of the partitions as given by the rule.

Note that all volumes used in a partitioning scheme must be audited (TMF protected).

When a queue file is partitioned, IBM MQ will distribute new messages evenly across all partitions.

Partitioning examples

In the first step, a rule is set up for queues with three partitions:

```
NSCNF>class QueuePattern
CLASS set to QueuePattern
NSCNF>object LargeQueues
OBJECT set to LargeQueues
NSCNF>set VolumePartition2 $SSD2
Property VolumePartition2 set to $SSD2
NSCNF>set VolumePartition3 $SSD3
Property VolumePartition3 set to $SSD3
NSCNF>set VolumePartition4 $SMF01
Property VolumePartition4 set to $SMF01
NSCNF>list
.
.
.
Class:                QueuePattern
Object:               LargeQueues
Property:             NumberPartitions
Value:                3
------------------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition2
Value:                $SSD2
------------------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition3
Value:                $SSD3
------------------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition4
Value:                $SMF01
```

Next, a queue is created where the queue name matches the pattern configured for “LargeQueues”, the queue file has three partitions (plus the master partition). The master partition is on the normal queue file subvolume, additional partitions are on volumes $SSD2, $SSD3 and $SMF01. (If you want to delete partitions from the configuration, remember to start with the last partition):

```
NSCNF>list
.
.
.
Class:                QueuePattern
Object:               LargeQueues
Property:             NumberPartitions
Value:                3
```

```
-----------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition2
Value:                $SSD2
-----------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition3
Value:                $SMF02
-----------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition4
Value:                $SMF01
.
.
.
NSCNF>del VolumePartition4
Property deleted successfully.
NSCNF>list
.
.
.
Class:                QueuePattern
Object:               LargeQueues
Property:             NumberPartitions
Value:                2
-----------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition2
Value:                $SSD2
-----------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition3
Value:                $SMF02
.
.
.
NSCNF>del VolumePartition3
Property deleted successfully.
NSCNF>list
.
.
.
Class:                QueuePattern
Object:               LargeQueues
Property:             NumberPartitions
Value:                1
-----------------------------------------------------
Class:                QueuePattern
Object:               LargeQueues
Property:             VolumePartition2
Value:                $SSD2
.
.
.
NSCNF>del VolumePartition2
Property deleted successfully.
```

runnscnf reports any errors, for example:

```
NSCNF>set VolumePartition6 $SMF02
*** Error: Next partition number to be used is 5.
*** Error: $SMF02 is not an appropriate value for property
```

```
olumePartition6 of object in class QueuePattern
*** Explanation: Invalid partition number.

NSCNF>del VolumePartition3
*** Error: Delete is restricted to last partition VolumePartition4
*** Error: Property not deleted.
*** Explanation: Invalid partition number.

NSCNF>set VolumePartition5 $SSD2
*** Error: volume $SSD2 in use by partition 2.
*** Error: $SSD2 is not an appropriate value for property VolumePartition5 of object in class QueuePa
*** Explanation: Volume already used as partition.
```

Queue file migration limitations

There are some issues to consider when you use queue file migration.

When a queue file migration is finished, the old queue file must be deleted, and the migration status information that can be viewed by using **runnscnf** must be removed from the system. This deletion is done automatically by the migration mechanism. However, the old queue file can only be deleted when no process has opened it. This is true even if all messages from the old file have already been moved to the new file. As some of the IBM MQ processes might keep the old queue file open for an extended period of time, it can happen that a migration status is reported as “Finished” in the **runnscnf STATUS MIGRATE** display for a long time. This is normal, there is no activity on the old queue file, and it only contains one record of data. Whenever the queue manager is restarted, the situation is cleaned up automatically.

You cannot revert back to IBM MQfor HPE NonStop V8.0.0 after you have partitioned queue files.

You cannot revert back to IBM MQfor HPE NonStop V8.0.0 while a queue file migration is in progress.

Chapter 13. TNS non-native application support

You can run TNS non-native applications with IBM MQ for HPE NonStop V8.

The support for TNS non-native applications is designed to allow legacy WebSphere MQ for HP NonStop Server V5.3 TNS applications to run with IBM MQ V8.0. It is not designed to support the development of new IBM MQ application programs.

The TNS non-native support in IBM MQ for HPE NonStop V8 is limited to the IBM MQ APIs and features found in the WebSphere MQ for HP NonStop Server V5.3 product. New IBM MQ V8 API features are not available in the non-native TNS environment. Note that non-native applications do not support MQCONNX.

The TNS non-native binding library is called `MQMTNS` and is located in the IBM MQ installation Guardian sub-volume chosen when IBM MQ for HPE NonStop V8 was installed.

TNS non-native applications using IBM MQ for HPE NonStop V8 must run as Guardian processes. IBM MQ V8 does not support OSS TNS non-native applications.

IBM MQ for HPE NonStop V8 supports C, COBOL, and TAL languages in the TNS non-native environment.

IBM MQ for HPE NonStop V8 TNS non-native applications require a single Guardian "param" named `MQPATHTNS` to be defined. The `MQPATHTNS` param must specify the OSS *mqinstall* location for the IBM MQ installation.

Note: *mqinstall* is the OSS install location specified using the `-i` option when IBM MQ for HPE NonStop V8 was installed.

For example:

```
param MQPATHTNS /home/david/mq8Beta6
```

Installing TNS non-native Support

You must explicitly install support for TNS non-native applications on IBM MQ for HPE NonStop V8.

Use the `mqgsamptns` script (in `opt/mqm/samp/bin`) to copy the source code and build scripts for the TNS non-native sample programs into a Guardian sub-volume.

The `mqprofile` must be sourced into the OSS shell. At an OSS login prompt, type:

```
. OSS_directory/var/mqm/mqprofile
```

Then, from the *OSS_directory*`/opt/mqm/samp/bin` directory, type:

```
./mqgsamptns Sample_Guardian_sub-volume
```

where*Sample_Guardian_sub-volume* is an empty Guardian sub-volume that will contain the non-native IBM MQ sample source code.

Building TNS non-native MQ sample programs

You must compile and bind the TNS non-native IBM MQ sample programs.

To compile and bind a TNS non-native Guardian executable version of the IBM MQ sample programs, use the supplied `BCSAMP`, `BCOBSAMP`, and `BTALSAMP` TACL routines.

- `BCSAMP` compiles and binds a C-language TNS non-native IBM MQ sample
- `BCOBSAMP` compiles and binds a COBOL TNS non-native IBM MQ sample
- `BTALSAMP` compiles and binds a TAL TNS non-native IBM MQ sample

The Guardian `MQCSTM` must be obeyed in the TACL session before using these TACL scripts.

For example, to use `BCSAMP` to compile and link an IBM MQ sample:

```
volume Sample_Guardian_sub-volume
BCSAMP amqsputc
```

The `BCSAMP` TACL routine will compile and bind the TNS non-native sample into the same sub-volume.

Chapter 14. JAVA Support

IBM MQ classes for JMS

IBM MQ classes for Java™ Message Service (JMS) is the JMS provider that is supplied with IBM MQ. For the HPE NonStop platform, JMS applications are typically run as standalone programs. The IBM MQ JMS provider has the ability to run in a Java EE environment.

IBM MQ classes for JMS support bindings connections to a local queue manager, and client connections to a local or remote queue manager.

An IBM MQ JMS sample program and installation verification test (IVT) is available in the following directory:

mqinstall/opt/mqm/java/bin/IVTRun

For more information on how to run this test, see http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q031740_.htm in the IBM MQ Knowledge Center,

IBM MQ classes for Java

IBM MQ classes for Java allow a Java application to connect directly to an IBM MQ queue manager.

IBM MQ classes for Java support bindings connections to a local queue manager and client connections to a local or remote queue manager.

An IBM MQ Java sample program and installation verification test (IVT) is available in the following directory:

mqinstall/opt/mqm/samp/wmqjava/samples/MQIVP.class

For more information on how to run this test, see http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q030690_.htm in the IBM MQ Knowledge Center.

IBM MQ Java for HPE NonStop requires HPE NonStop Server for Java 7 or later.

Chapter 15. SSL Channels

SSL channels are available with multiple certificate support enabled using the IBM MQ V8 CERTLABL channel attribute. See the chapter on NonStop specific configuration for details about setting up SSL channels.

Queue manager SSL certificates

Each queue manager requires one or more certificate files and associated stash files.

Certificate files are in ASCII PEM format with a name that ends with `.pem`. A certificate file contains an X509 personal certificate with its associated private key. A stash file contains a masked form of the passphrase used to protect the certificate private key. Certificate files can be created using OpenSSL, the IBM MQ ikeyman tool on Linux and Windows, or can be generated by a public certificate authority. Stash files are created using the IBM MQ **amqrsslc** tool.

A queue manager stores its SSL related files in a directory that is specified by the queue manager SSLKEYR attribute. The SSLKEYR attribute defaults to *mqinstall*`/var/mqm/qmgrs/QMGRNAME/ssl`. A queue manager can store its SSL files in any directory by using **runmqsc** to alter the queue manager SSLKEYR attribute.

A queue manager has a default certificate that is always named `cert.pem`. Its associated default stash file is always named `Stash.sth`.

SSL Channels that do not have a CERTLABL specified, use the default queue manager certificate and stash file, with the following full names:

```
SSKLEYR/cert.pem       Default qmgr certificate and private key
SSLKEYR/Stash.sth       Default qmgr stash file
```

SSL channels that have a CERTLABL specified use a certificate file and a stash file with the following names:

```
SSKLEYR/CERTLABL.pem       qmgr certificate and private key for label CERTLABL
SSLKEYR/CERTLABL.sth       qmgr stash file for label CERTLABL
```

For example, a channel with a CERTLABL attribute of "MYLABEL" will use a certificate file and stash file with the following names:

```
SSKLEYR/MYLABEL.pem
SSLKEYR/MYLABEL.sth
```

Each queue manager additionally needs a Certificate Authority file named `trust.pem`. This file contains the public certificates of all the signing authorities used by the organization. The queue manager uses the `trust.pem` file to verify any certificates presented by remote queue managers that are communicating using SSL.

The `trust.pem` file has a fixed name of the form:

```
SSKLEYR/trust.pem
```

Each queue manager can optionally use a Certificate Revocation List (CRL) file that contains the latest list of certificates that have been revoked by their signing authority. The CRL file has a fixed name of the form:

SSKLEYR/crl.pem

Cipherspecs

A cipherspec is a name used to select algorithms for signing, key exchange, and encryption.

The cipherspec that a channel uses is specified by its SSLCIPH channel attribute. Channels that do not have a cipherspec in their SSLCIPH attribute are not SSL channels and send their message traffic in clear-text over the network.

The following is a list of valid cipherspecs with their associated protocol version:

Table 14. Valid cipherspecs

MQ Channel SSLCIPH	Protocol	FIPS/SuiteB
DES_SHA_EXPORT	SSL v3	
DES_SHA_EXPORT1024	SSL v3	
NULL_MD5	SSL v3	
NULL_SHA	SSL v3	
RC2_MD5_EXPORT	SSL v3	
RC4_56_SHA_EXPORT1024	SSL v3	
RC4_MD5_EXPORT	SSL v3	
RC4_MD5_US	SSL v3	
RC4_SHA_US	SSL v3	
TRIPLE_DES_SHA_US	SSL v3	
TLS_RSA_WITH_DES_CBC_SHA	TLS v1.0	
TLS_RSA_WITH_AES_128_CBC_SHA	TLS v1.0	FIPS
TLS_RSA_WITH_AES_256_CBC_SHA	TLS v1.0	FIPS
TLS_RSA_WITH_3DES_EDE_CBC_SHA (deprecated)	TLS v1.0	FIPS
TLS_RSA_WITH_NULL_SHA256	TLS v1.0	
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS v1.0	FIPS
TLS_RSA_WITH_AES_128_GCM_SHA256	TLS v1.0	FIPS
TLS_RSA_WITH_AES_128_GCM_SHA256	TLS v1.0	FIPS
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS v1.0	FIPS
ECDHE_RSA_AES_256_GCM_SHA384	TLS v1.2	FIPS
ECDHE_RSA_AES_128_CBC_SHA256	TLS v1.2	FIPS
ECDHE_RSA_AES_128_GCM_SHA256	TLS v1.2	FIPS
ECDHE_RSA_AES_256_CBC_SHA384	TLS v1.2	FIPS
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS v1.2	FIPS
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS v1.2	FIPS
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS v1.2	FIPS SuiteB 128

Table 14. Valid cipherspecs (continued)

MQ Channel SSLCIPH	Protocol	FIPS/SuiteB
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS v1.2	FIPS SuiteB 192

SSL v3 is now considered to be a weak protocol and cipherspecs that use it are disabled by default. To use SSL v3 cipherspecs, you must either define an AMQ_SSL_V3_ENABLE environment variable or specify the AllowSSLV3 attribute in the SSL stanza of the queue manager `qm.ini` file. For example:

```
export AMQ_SSL_V3_ENABLE=1
```

Alternatively, in the `qm.ini`:

```
SSL:
     AllowSSLV3=Yes
```

Chapter 16. Sample programs

Various sample programs are provided with IBM MQ for HPE NonStop V8.

OSS environment and TACL scripts

You must set up an environment on NonStop before you can use the sample files provided with IBM MQ

mqprofile script

The `mqprofile` script is automatically generated when the product is installed. The `mqprofile` script is intended to be sourced into an OSS shell. The script adds the IBM MQ installation `bin` directory to the PATH and defines the MQINST environment variable.

The MQINST environment variable is used by the IBM MQ sample build scripts and is not required to run IBM MQ.

To source the `mqprofile`, at an OSS login prompt, type:

```
. OSS_directory/var/mqm/mqprofile
```

MQCSTM

MQCSTM is a Guardian TACL obey file that sets the MQINST param that is needed by the Guardian IBM MQ sample build scripts. The MQCSTM file is automatically generated in the installation subvolume when the product is installed.

At a TACL prompt, type:

```
obey $vol.subvol.MQCSTM
```

Installing and building sample Programs

You must install and build the sample programs before you can use them.

Installing native Guardian Source for samples

You use the `mqgsamp` script (in `opt/mqm/samp/bin`) to copy the source code for the IBM MQ sample programs into a Guardian sub-volume. The `mqprofile` script must be sourced into the OSS shell. At an OSS login prompt, type:

```
. <OSS directory>/var/mqm/mqprofile
mqgsamp Sample_Guardian_sub-volume
```

where *Sample_Guardian_sub-volume* is an empty Guardian sub-volume that will contain the IBM MQ sample source code.

Building native Guardian IBM MQ sample programs

To compile and link the native Guardian executable versions of the IBM MQ sample programs, use the supplied BCSAMP, BLSAMP, and BTSAMP TACL routines.

- BCSAMP compiles and links a C-language Guardian MQ sample
- BLSAMP compiles and links a COBOL Guardian MQ sample
- BTSAMP compiles and links a pTAL Guardian MQ sample

The Guardian MQCSTM must be obeyed in the TACL session before using these TACL scripts.

Example:

To use BCSAMP to compile and link an IBM MQ sample:

```
volume Sample_Guardian_sub-volume
BCSAMP amqsputc
```

The BCSAMP TACL routine will compile and link the sample into the same sub-volume.

Building OSS IBM MQ sample programs

To compile and link the OSS executable versions of the IBM MQ sample programs, use the supplied `bcsamp` or `blsamp` shell scripts.

The OSS sample build scripts are located in *OSS_directory*`/opt/mqm/samp/bin`.

- `bcsamp` - compiles and links a C-language OSS IBM MQ sample
- `blsamp` - compiles and links a COBOL OSS IBM MQ sample

The OSS `mqprofile` file must be sourced into the OSS shell before using these scripts.

```
. OSS_directory/var/mqm/mqprofile
```

For example:

To use `bcsamp` to compile and link an MQ C-language sample:

```
. OSS_directory/var/mqm/mqprofile bcsamp amqsput0.
```

For example:

To use `blsamp` to compile and link a COBOL MQ sample:

```
. OSS_directory/var/mqm/mqprofile blsamp amq0put0.cbl
```

Chapter 17. Problem handling

There are actions you should take to handle problems with IBM MQ for HPE NonStop V8.

In case of errors or problems, the following information should be consulted:

IBM MQ error logs and FFST files
These are available at the location described in the main IBM MQ documentation.

EMS messages
For details about EMS messages, see "EMS."

If this information does not help you resolve the issue, you might want to contact IBM Support. In that case you might be asked to provide and SDCP file (see "sdcp tool" on page 92)

EMS

In addition to FFST reports and error logs MQ 8 supports another way for user notifications, the Event Management Service (EMS).

By default no EMS events are generated. If you want to enable MQ EMS events, you must ensure that the environment variable MQEMSEVENTS is defined in the context of MQ processes. The value is a three-character string, which is a decimal value interpreted as a bit map, as follows:

Table 15.

EMS message	Bit-map entry	MQEMSEVENT value
FFST	0x00000001	1
START / STOP	0x00000002	2
PERFORMANCE	0x00000004	4
CHANNEL	0x00000008	8
QUEUE MANAGER	0x00000010	16
MESSAGE	0x00000020	32
ERROR	0x00000040	64
MQ 8 events	0x00000080	128
ALL	0x000000FF	255

Thus, to switch on all EMS events for MQ, you must define the following environment variable in the environment from which any administration commands are issued:

```
/home/mqm: export MQEMSEVENTS=255
```

Using a different collector

You can use a different collector to the default collector.

By default IBM MQ V8 reports EMS events to $0. A different collector can be defined for IBM MQ V8 events by using the environment variable MQEMSCOLLECTOR:

```
/home/mqm: export MQEMSCOLLECTOR='$0'
```

Note the process name $0 is quoted to avoid environment variable substitution performed by the shell.

Messages

The System ID of all MQ 8 EMS events is `IBM.WMQS.V80`.

IBM MQ V8 supports following WebSphere MQ V5.3 EMS events:

- FFST message
- Queue Manager event
- Performance event
- Channel event
- Display Message Event
- Report Error Event

Following EMS events are introduced with IBM MQ V8:

- Guardian and OSS system call failures.
- Takeover events. Queue manager backup takes over.
- Memory allocation failure. Some IBM MQ process is running out of dynamic memory.
- Cache manager error.
- Process termination by ABEND().
- Other errors.

sdcp tool

`sdcp` is the Service Data Collection and Packaging tool.

`sdcp` is an OSS shell script that can be used to collect and package important data from IBM MQ for HPE NonStop V8. The primary purpose of `sdcp` is to reduce the time and effort required from users and IBM Support in gathering the most critical and commonly required data in support of the technical analysis of a PMR.

`sdcp` gathers data about the IBM MQ installation and the queue managers running in that installation. Data is also collected about the general operating environment on the HPE NonStop Server. Queue managers do not need to be stopped when running `sdcp`.

No customer application message data is collected to avoid inadvertently collecting sensitive information. Use of IBM MQ utilities to collect queue, channel, and authority information can be excluded if necessary. `sdcp` requires no interactive input from the user and makes no modifications to the system, IBM MQ installation, or the queue managers.

Usage

`sdcp` runs from the OSS shell and requires a logged-in user ID that is an administrator of the IBM MQ installation.

The command line interface is as follows:

```
sdcp [-d] [-e] [-f] [-w workdir] [-m queuemgr] [-p pmrNumber]
```

Where:

-d

Enables OSS shell debug output for the script

-e

Excludes the use of IBM MQ utility programs to collect configuration and status information about running queue managers. If not specified, IBM MQ command line utilities are used to collect this data for running queue managers.

-f

Enables the fully detailed collection of certain items, including the WHOHAS utility. The full level of detailed data collection can take over an hour on some systems and is usually not necessary. Only use this parameter if directed to by IBM Support. If not specified, a normal level of detail is collected.

-w

Is the name of an OSS directory where `sdcp` stores temporary files that it uses while running, and where the final output is created. If not specified, the current working directory is used.

-m

queuemgr is the name of the queue manager to collect data for. If not specified, `sdcp` collects data for all queue managers in the installation.

-p

pmrNumber defines the final archive and intermediate file name prefix, and must be formatted as an IBM Support PMR number, branch and country code; with each item being separated by a dot (.) or comma (,). For example, 11111.222.333.

Note: The use of the –p flag is recommended when collecting data for submission to IBM Support. Proper use of this flag will ensure that the file gets uploaded to the appropriate storage location.

For simple usage, just typing `sdcp` causes the collection of data from the installation, plus all queue managers in that installation (whether running or not).

To reduce the volume of data and the time taken to collect the data, you can run `sdcp` to collect data for a specific queue manager:

```
sdcp –m qmgr
```

In the event that one of the queue managers you are collecting information on appears to be unresponsive, specify the –e parameter to exclude the use of IBM MQ utilities to collect data for running queue managers:

```
sdcp –m qmgr -e
```

The following example illustrates collecting data with typical parameters specified:

```
sdcp -w /home/fred -m QM1 -p 99999,888,777
```

Result of running sdcp

Running sdcp creates a compressed tar archive ("tarball") in the working directory (or current directory, if –w was not specified) containing all of the data that sdcp collected from the installation, queue managers, and the system.

The tarball can then be transferred electronically to IBM Support (using binary mode ftp) for analysis.

The sdcp working directory is used as a location for temporary files created as data is gathered about the installation. By default, all files created in the working directory have a common prefix of MQSDCP, followed by the date and time when sdcp was started, for example: MQSDCP-170420-123729.

If the -p flag is specified, the prefix consists of the PMR number appended with SDCP, followed by the date and time when sdcp was started, for example: 03825,122,000-SDCP-170420-123729

The final tarball is named using the same prefix that was used for the temporary files, followed by -archive.tar.Z, for example: MQSDCP-170420-123729-archive.tar.Z.

Information collected by sdcp

sdcp uses standard HPE NonStop Server OSS and Guardian utilities to collect data.

In detail, sdcp collects the following information:

- The OSS and Guardian filesystem objects for the IBM MQ installation and queue managers.
- Version information for all sdcp binaries
- The status and state of all sdcp processes running for queue managers
- Information about the HPE NonStop Server operating system level, versions of critical HPE system software subsystems
- Basic information about the OSS shell environment that sdcp is running in
- TMF subsystem status
- Disk space summary for the system
- Physical and virtual memory status of each CPU
- OSS fileset status
- Basic configuration of the EMS subsystem
- Status of all TCP/IP subsystems
- The contents of the IBM MQ errors and trace directories
- The IBM MQ installation wide configuration files mqs.ini
- The results of running dspmq (which queue managers are defined and their state)
- For each running queue manager, unless excluded using the –e parameter, configuration and status data for all IBM MQ objects, channels, authority data, and clustering objects

sdcp typically takes a few minutes to run, depending on system load, but does not interrupt any operations. With default settings and two running queue managers, sdcp takes approximately 5 minutes on a NB5400C, depending on system load.

IBM Support directs customers to use this tool in situations where a queue manager is experiencing problems, to quickly and accurately capture data, and enable faster progress into problem recovery mode (where necessary).

Chapter 18. NonStop specific log messages

The following NonStop specific message might be issued by IBM MQ for HPE NonStop V8.

AMQ5401: The volume is not audited

context:

`crtmqm`

meaning:

`crtmqm` was called with option `–ng` (setting Guardian subvolume) but the subvolume specified does not reside on an audited disk.

consequence:

The `crtmqm` command fails, no queue manager is created.

Recommended action:

Specify a subvolume of an audited disk when creating a queue manager.

AMQ5402: Cache Mgr ... from QM ... had failed

context:

Running queue manager.

meaning:

A cache manager process in the queue manager given in the message has failed.

consequence:

All non persistent messages in queues of this cache manager are lost. Applications will receive Reason 2208 (MQRC_FILE_SYSTEM_ERROR) when trying to MQGET these messages. This error will be reported once for each lost message. When MQGET has been called for each lost message, normal operation will continue. The failed process will automatically be restarted by the system.

Recommended action:

When using non-persistent messages code your application to be able to deal with that error code.

AMQ5408: .. called PROCESS_STOP_ on ... with rc=....

context:

`endmqm` phase.

meaning:

In the `endmqm` phase some IBM MQ processes were hard stopped by the IBM MQ engine.

consequence:

This is an informational message only.

Recommended action:

None.

AMQ5408: Configuration for queue manager '&3' not fault tolerant.

context:

`strmqm` phase.

meaning:

The queue manager was configured to run in non-fault tolerant mode, that is, without an EC backup process.

consequence:

This is an informational message only. A failure of the queue manager CPU brings down the queue manager, the queue manager has to be restarted manually.

Recommended action:

None.

AMQ5404: No CPU available for backup of EC for queue manager

context:

`strmqm` phase or running queue manager.

meaning:

The queue manager is configured to run in fault tolerant mode, but there is no CPU available to start the backup process in.

consequence:

If IBM MQ is running on a two CPU system, this message is normal if a CPU goes down. If running on a system with four or more CPUs, some CPUs might not be in the allowed list for IBM MQ (`runnscnf` configuration). If no allowed CPU is available, IBM MQ runs in non-fault-tolerant mode until an allowed CPU becomes available again.

Recommended action:

Consider configuring more allowed CPUs.

AMQ5405: Takeover of queue manager '&3' occurred...

context:

Running queue manager.

meaning:

A CPU failure has caused the queue manager to move to a different CPU. A typical cause for this is the failure of the queue manager CPU. It can also happen if, for any reason, the primary EC process fails.

consequence:

IBM MQ recovers by itself. Applications coded to do retries after a delay for failed connections are able to reconnect to the queue manager again within a few seconds.

Recommended action:

Analyze the root cause of the failing CPU.

AMQ5407: Backup of queue manager ... could not be started

context:

`strmqm` or running queue manager.

meaning:

IBM MQ has tried to start a backup EC process to run in fault tolerant mode but this `ahs` failed.

consequence:

Check EMS and FFSTs for the reason why the EC backup process could not be started.

Recommended action:

Clear the root cause of the failure.

AMQ7024: Arguments supplied to a command are not valid. Usage: altmqusr [-m QMgrName] -p Principal (-u UserName | -r)

context:

A required option is missing, or an invalid option was used.

meaning:

A required option is missing or, an invalid option was used.

consequence:

The command is not executed.

Recommended action:

Correct the input and repeat the command.

The HP NonStop Server User name was specified incorrectly

context:

Execution of `altmqusr` command. This message is shown at the terminal, it does not appear in the log.

meaning:

A value to the `-u` option was invalid; the NonStop OS user ID does not exist.

consequence:

The command is not executed.

Recommended action:

Correct the input and repeat the command.

AMQ7028: The queue manager is not available for use

context:

Execution of `altmqusr` command. This message is shown at the terminal, it does not appear in the log.

meaning:

A value to the -m option is invalid; the queue manager does not exist.

consequence:

The command is not executed.

Recommended action:

Correct the input and repeat the command.

AMQ7024: Arguments supplied to a command are not valid. Usage: dspmqusr [-m QMgrName] [-p Principal]

context:

Execution of dspmqusr command. This message is shown at the terminal, it does not appear in the log.

meaning:

A required option is missing or an invalid option was supplied.

consequence:

The command is not executed.

Recommended action:

Correct the input and repeat the command.

Username mapping for Queue Manager 'i517x' Principal Userid Username Alias GroupName GroupType The Principal name was specified incorrectly.

context:

Execution of dspmqusr command. This message is shown at the terminal, it does not appear in the log.

meaning:

A value for the option -p is invalid: the principal does not exist.

consequence:

The command is not executed.

Recommended action:

Correct the input and repeat the command; use dspmqusr to verify existing principals.

Notices

This information was developed for products and services offered in the U.S.A.

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant 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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

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

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of IBM MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Sending your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

- Send an email to ibmkc@us.ibm.com
- Use the form on the Web at: www.ibm.com/software/data/rcf/

Include the following information:

Your name and address
Your email address
Your telephone or fax number
The publication title and order number
The topic and page number related to your comment
The text of your comment

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

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

Thank you for your participation.