WebSphere MQ Integrator for HP-UX - Implementing with MC/ServiceGuard

Version 1.0

10th March 2003

Andrew Ford
WebSphere MQ Integrator Development
IBM UK

Take Note!

Before using this report be sure to read the general information under "Notices".

First Edition, March 2003
This edition applies to Version 1.0 of *"Configuring WebSphere MQ Integrator for HPUX with MCSG"* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

Trademarks

Acknowledgments

This support pack is based upon the high availability SupportPacs written by Graham Wallis for AIX and Solaris.

Summary of Changes

Date	Changes
March 2003	Version 1.0 - Initial release as a SupportPac

Bibliography

The expected audience for this document is trained professionals with experience of:

- MCSG
- WebSphere MQSeries for HPUX 11
- WebSphere MQ Integrator for HPUX 11
- DB2 for HPUX 11

It is expected that the reader is familiar with the concepts and activities involved in setting up and running MCSG clusters, MQSeries queue managers, and MQ Integrator brokers and User Name Servers. The following sources of information may be useful in this regard:

- WebSphere MQSeries product manuals, especially
  - *WebSphere MQSeries Command Reference*, SC33-1369-11
- WebSphere MQSeries Integrator product manuals, especially
  - *WebSphere MQ Integrator for HPUX Installation Guide*,
  - *WebSphere MQ Integrator for HPUX Administration Guide*,
- DB2 product manuals, especially
  - *DB2 for HPUX Administration Guide: Design and Implementation*

All of these books are available online.

For WebSphere MQSeries and WebSphere MQ Integrator:

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

For DB2:

http://www-3.ibm.com/software/data/hp/dmproducts.html

Chapter 1. Introduction

Concepts

HP-UX is Hewlett Packard's UNIX operating system which runs on its 9000 series systems.

Multi-Computer/ServiceGuard (MC/ServiceGuard) is a control application that can link HP-UX servers into highly available clusters. Clustering servers enables parallel access to data, which can help provide the redundancy and fault resilience required for business-critical applications.

WebSphere MQSeries for HP-UX provides asynchronous message and queuing capabilities with assured, once-only delivery of messages.

By using WebSphere MQ Integrator for HP-UX, and MC/ServiceGuard for HP-UX together, it possible to further enhance the availability of WebSphere MQ Integrator Brokers. With a suitably configured MC/ServiceGuard cluster, it is possible for failures of power supplies, nodes, disks, disk controllers, networks, network adapters or queue manager processes to be detected and automatically trigger recovery procedures to bring an affected Queue Manager back online as quickly as possible.

This SupportPac provides notes and sample scripts to assist with the installation and configuration of WebSphere MQ Integrator for HP-UX in an MC/ServiceGuard environment. Used in conjunction with the documentation for HP-UX, MC/ServiceGuard and MQSeries, it shows how to create and configure MQSeries queue managers, and WebSphere MQ Integrator brokers such that they are amenable to operation within an MC/ServiceGuard cluster. It also shows how to configure MC/ServiceGuard to take control of such.

Certification

The certification process for WebSphere MQ Integrator V2.1 with MC/ServiceGuard was carried out by the Hewlett Packard Partner Technology Access Center. Included with this service pack is a copy of the test report they generated, it contains a number of MC/ServiceGuard set up scripts that could be used as templates for creating your own implementation. The document also expands on Hewlett Packard specific areas that are not covered in this document.

Definition of the word *"cluster"*

The word "cluster" has a number of different meanings within the computing industry. Throughout this document, unless explicitly noted otherwise, the word "cluster" is used to describe an "MC/ServiceGuard cluster", which is a collection of nodes and resources (such as disks and networks) which cooperate to provide high availability of services running within the cluster. It is worth making a clear distinction between such an "MC/ServiceGuard cluster" and the use of the phrase "MQSeries Cluster", which refers to a collection of queue managers which can allow access to their queues by other queue managers in the cluster. The relationship between these two types of cluster is described in "Relationship to MQSeries Clusters" later in this chapter.

Functional Capabilities

Cluster Configurations

This SupportPac can be used to help set up either **standby** or **takeover** configurations, including mutual takeover where all cluster nodes are running WebSphere MQ Integrator workload.

A **standby** configuration is the most basic cluster configuration in which one node performs work whilst the other node acts only as standby. The standby node is referred to as the "adoptive node" and does not perform work; this configuration is sometimes called "cold standby". Such a configuration requires a high degree of hardware redundancy. To economise on hardware, it is possible to extend this configuration to have multiple worker nodes with a single standby node, the idea being that the standby node can take over the work of any worker node. This is still referred to as a standby configuration and sometimes as an "N+1" configuration.

A **takeover** configuration is a more advanced configuration in which all nodes perform some kind of work and critical work can be taken over in the event of a node failure. This type of cluster configuration is sometimes referred to as "Active/Active" to indicate that all nodes are actively processing critical workload.

In an “Active/Active” configuration all nodes perform highly available (movable) work, and consideration must be given to the peak load which may be placed on any node which can take over the work of other nodes. Such a node must possess sufficient capacity to maintain an acceptable level of performance.

Figure 1. A shared nothing cluster

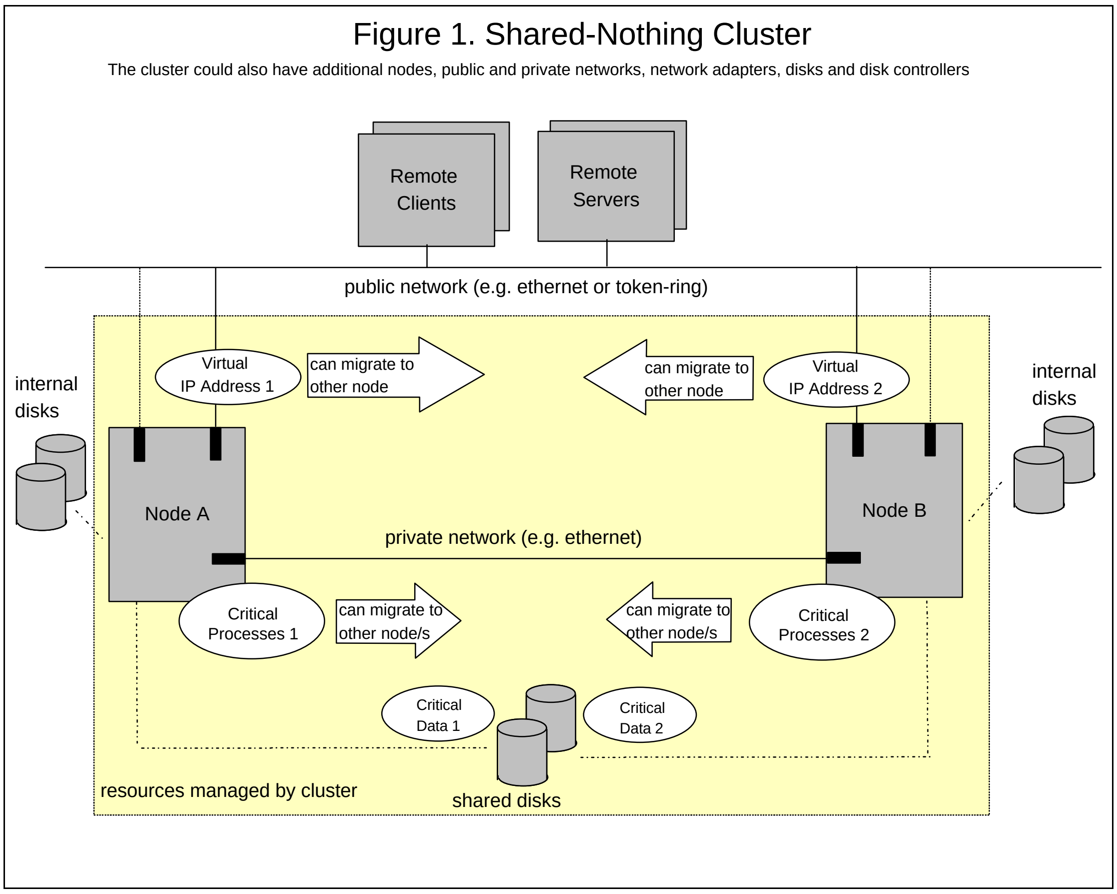


Figure 1. Shared-Nothing Cluster
The cluster could also have additional nodes, public and private networks, network adapters, disks and disk controllers
Remote Clients
Remote Servers
public network (e.g. ethernet or token-ring)
internal disks
Virtual IP Address 1
can migrate to other node
can migrate to other node
Virtual IP Address 2
internal disks
Node A
private network (e.g. ethernet)
Node B
Critical Processes 1
can migrate to other node/s
can migrate to other node/s
Critical Processes 2
Critical Data 1
Critical Data 2
resources managed by cluster
shared disks

This SupportPac contains scripts that can be used to configure one or more application servers that each contain an WMQI broker or UNS. The SupportPac also contains scripts that enable each application server to be monitored.

The scripts can be used as shipped, or as a basis from which to develop your own scripts. Whether the scripts are being used to manage brokers or a UNS they rely on the scripts from MQSeries SupportPac MC6B to manage queue managers and when managing a broker they also rely on the use of MC/ServiceGuard scripts to manage the database instance containing the broker database. MC/ServiceGuard scripts to manage database instances are available separately.

The example scripts supplied with this SupportPac contain the logic necessary to allow one or more brokers and one UNS to be run within a cluster. This is described fully in Chapter 4. It is possible to configure multiple resource groups. This, combined with the ability to move resource groups between nodes, allows you to simultaneously run highly available brokers on the cluster nodes, subject only to the constraint that each broker can only run on a node that can access the necessary shared disks. This constraint is enforced by the package, which has a list of which nodes it can run on and the volume groups it owns. In the case of the UNS, you can only configure one UNS per cluster. This is because the name "UserNameServer" is fixed, which would cause different UNSs to interfere if you tried to configure them simultaneously on one node[1]. The script used for creating an HA compliant UNS enforces this constraint.

The ability to move resource groups between nodes, makes it important to consider the peak load which may be placed on any node which can run multiple resource groups. Such a node must possess sufficient capacity to maintain an acceptable level of performance.

Network connections

Clustered services, such as MQSeries queue managers, are configured to use virtual IP addresses which are under cluster control. When a clustered service moves from one cluster node to the other, it takes its virtual IP address with it. The virtual IP address is different to the stationary physical IP address that is assigned to a cluster node. Remote clients and servers which need to communicate with clustered services must be configured to connect to the virtual IP address and must be written such that they can tolerate a broken connection by repeatedly trying to reconnect.

Specified Operating Environment

Hardware Requirements

The minimum hardware requirements for a cluster are at least two HP-UX servers, a shared disk enclosure, a private (non-IP) network (e.g. serial) and a public (IP) network. Refer to the MC/ServiceGuard Planning Guide for details of how to select appropriate disk technologies and networks. As described in the Planning Guide, it is likely that you will want to include additional networks and network adapters to eliminate these single points of failure. The hardware requirements for a cluster which will support WebSphere MQ Integrator are the same as the above, but you should plan carefully the layout of filesystems and disks. Refer to "Chapter 3. Configuration" of this document for assistance in planning the filesystem layout.

The example scripts included in this SupportPac were tested by Hewlett Packard on a pair of HP9000 `L' class servers connected to a shared disk subsystem. Communication between the servers was via Ethernet and serial networks.

Please note that the certification carried out by Hewlett Packard is ONLY for the software listed in the requirements below, this does not mean that other combinations will work only that they have not been certified to work.

Software Requirements

Versions of software tested on are:

- HPUX Version 11i
- MC/ServiceGuard Version 11.14
- WebSphere MQ Integrator for HPUX Version 2.1, including
  - WebSphere MQSeries for HPUX 11 Version 5.3
  - DB2 for HP-UX Version 7.1 Enterprise Edition with fixpac1
- WebSphere MQSeries SupportPac MC6B: WebSphere MQ for HP-UX - Implementing with Multi-Computer/ServiceGuard (Version 2.0).

Networks

The SupportPac has been tested with TCP/IP public networks, and serial private networks. TCP/IP networks could be used for both public and private networks.

It would also be possible to configure MCSG to handle SNA networks.

[1] Technically, you could partition the cluster into non-overlapping sets of nodes and configure multiple resource groups each confined to one of these sets. Each package could then support a UNS. However, this is not recommended.

Chapter 2. Installation

HPUX, MCSG, WebSphere MQSeries, WebSphere MQSeries SupportPac MC6B, WebSphere MQ Integrator and DB2 should already be installed, using the normal procedures onto internal disks on each of the nodes. It is important that under normal operating conditions you are running identical versions of all the software on all cluster nodes.

When installing WebSphere MQSeries, note the advice in SupportPac MC6B about filesystems and the requirement that the "mqm" username and "mqm" groupname have been created and each have the same numeric value on all of the cluster nodes.

Installing the SupportPac

As part of installing SupportPac MC6B, you will have created a working directory which, by default, is called /MQHA/bin. You can either install this SupportPac into the same directory or use a different one. If you use a different location you'll need to change the example scripts. The following description assumes that you are using the /MQHA/bin directory.

Download the SupportPac onto each of the cluster nodes into the /MQHA/bin directory and uncompress and untar it. Then ensure that all the scripts are executable, for example by issuing:

```
chmod +x ha* wmqi*
```

The SupportPac contains the following files:

Broker-related commands and methods	**UNS-related commands and methods**
• hamqsicreatebroker • hamqsideletebroker • hamqsiaddbrokerstandby • hamqsiremovebrokerstandby • hamqsi_start_broker_as • hamqsi_start_broker • hamqsi_stop_broker_as • hamqsi_stop_broker • hamqsi_monitor_broker_as	• hamqsicreateusernameserver • hamqsideleteusernameserver • hamqsiaddunsstandby • hamqsiremoveunsstandby • hamqsi_start_uns_as • hamqsi_start_uns • hamqsi_stop_uns_as • hamqsi_stop_uns • hamqsi_monitor_uns_as
General files and utilities	**Files supplied by Hewlett Packard**
• Hamqsi_qm_datapath • README • license2.txt • IC64_V10.pdf	• WMQI2.1cert.pdf • control.sh • wmqi.sh • wmqi.ascii

The hamqsicreatebroker and hamqsicreateusernameserver scripts used during the creation and configuration of a broker or UNS also create an Application Monitor for that broker or UNS. These application monitors are saved in /MQHA/bin and are called hamqsi_applmon.<broker>, where <broker> is the name of the broker to be monitored and hamqsi_applmon.UNS.

Chapter 3. Planning your configuration

WMQI contains a number of components which perform different functions within a broker domain. The message broker runs message flows, and therefore is critical to the continued provision of the messaging service and should be run under MC/ServiceGuard control. The broker depends on a queue manager, which must be run on the same node as the broker and is another critical function which needs to be under MC/ServiceGuard control. The broker also depends on access to a DB2 database known as the broker database, which may either be run locally (on the same node as the broker) or remotely on a node outside the cluster. The database is another critical component of the solution architecture and should be placed under MC/ServiceGuard control if it is run locally, inside the cluster.

The broker domain may include a User Name Server (UNS), to provide topic-based security. If this is the case, then the UNS is also a critical service. Like the broker database, the UNS can be placed either inside or outside the cluster. If it is inside the cluster (i.e. it runs on one of the cluster nodes) then it needs to be placed under MC/ServiceGuard control. The UNS depends on a queue manager which must be run on the same node as the UNS, also under MC/ServiceGuard control.

The broker domain is controlled via the Configuration Manager, which is responsible for storing the definitions of brokers, execution groups and message flows and deploying them. Since availability of the Configuration Manager only affects the ability to change the configuration, and does not affect the delivery of messages, it is considered to be less critical than the previously described components. This document provides assistance in making the message broker (and its queue manager and database) and the UNS (and its queue manager) highly available using MC/ServiceGuard. It does not include consideration of the Configuration Manager, which does not run on HP-UX. Throughout the remainder of this document, the Configuration Manager and its corresponding queue manager are assumed to be located on a separate machine which can connect to the cluster.

Figure 2. Inclusion of WMQI components in the cluster

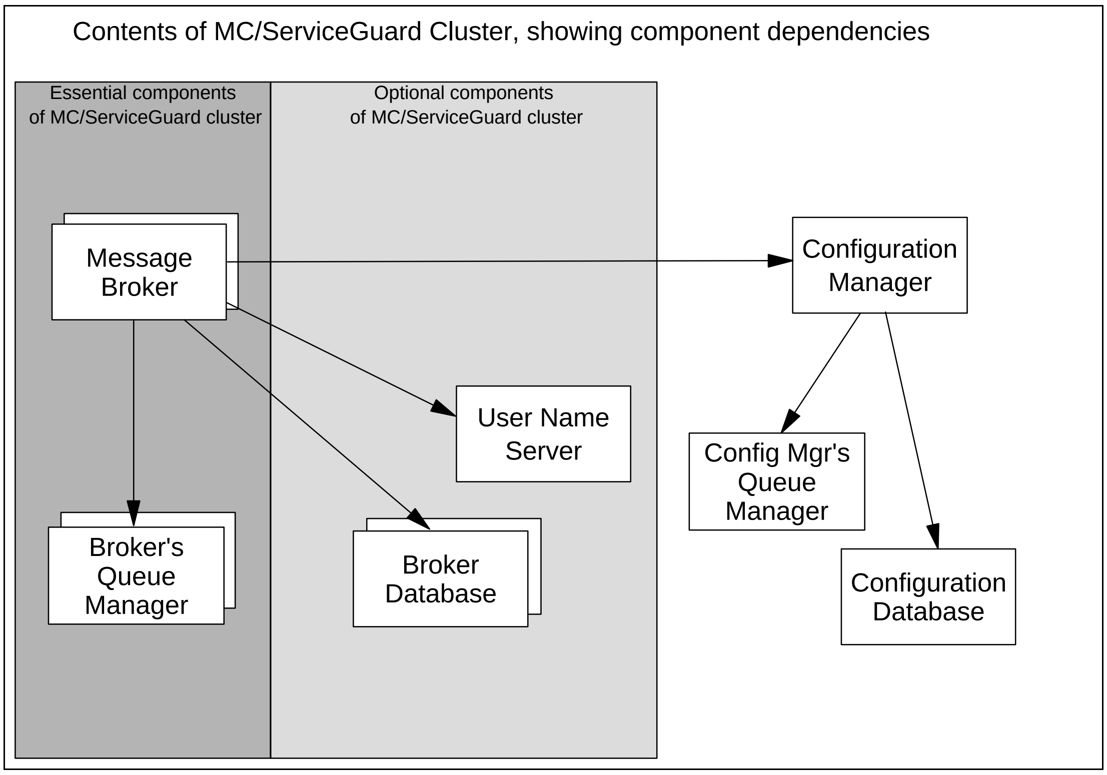

Contents of MC/ServiceGuard Cluster, showing component dependencies
Essential components of MC/ServiceGuard cluster
Optional components of MC/ServiceGuard cluster
Message Broker
Configuration Manager
User Name Server
Config Mgr's Queue Manager
Broker's Queue Manager
Broker Database
Configuration Database

Figure 2 shows the relationships between these components and their recommended placement relative to the MC/ServiceGuard Cluster. The double boxes show which components have an extra instance if an additional broker is configured. Before proceeding, the reader is advised to study the diagram and consider which components of their solution architecture they wish to place on the cluster nodes and which components will be run remotely. The components which you decide to run on the cluster nodes need to be put under MC/ServiceGuard control. Components which are run outside the cluster may also need to made highly available. They could, for example, be put into a separate cluster. The contents of this document could be used in conjunction with the product documentation as a basis for how to do that. Alternatively the remote components could be made highly available in other ways, but it is outside the scope of this document to describe this.

The remainder of this Chapter describes the message broker and UNS components in more detail and then describes the steps needed to configure the cluster.

Message broker

A WMQI message broker relies on an MQSeries queue manager and a DB2 database referred to as the broker database, which in turn depend on other lower level resources, such as disks and IP addresses.

The broker database might be run in a database instance on the same node as the message broker, in which case the database instance and its lower level dependencies must be failed over with the message broker. Alternatively the broker database might be run in a remote instance accessed using a remote ODBC connection, in which case it is necessary is to ensure that the database is accessible from either cluster node, so that the message broker can operate correctly on either node.

The unit of failover in MC/ServiceGuard is a package. A package is a collection of resources (e.g. disks, IP addresses, processes) needed to deliver a highly available service. Ideally a package should contain only those processes and resources needed for a particular instance of a service. This approach maximises the independence of each package, providing flexibility and minimising disruption during a failure or planned maintenance.

The smallest unit of failover of an WMQI message broker is the broker service together with the MQSeries queue manager upon which it depends and the broker database, if run locally. The optimal configuration of brokers and groups is to place each broker in a separate group, with the resources upon which it depends. Additional brokers should be configured to use separate database instances and be placed in separate groups. The UNS should also be placed in a separate group. A broker group must contain the message broker, the broker's queue manager and the queue manager's shared disks and IP address. If the broker database is run locally then the broker group also needs to contain the database instance and its shared disks.

The resource groups that might therefore be constructed, depending on your placement of broker databases are as shown in the Figure 3 and Figure 4.

You *could* put multiple message brokers (and associated resources) into a group, but if you did so they would all have to failover to another node together, even if the problem causing the failover were confined to one message broker or its dependencies. This would cause unnecessary disruption to any other message brokers in the same group. Users who wish to use application monitoring should also note the restriction that only one application server in a package can be monitored. If you wanted to monitor multiple brokers in the same group, you would need to edit the example scripts so that the brokers were in the same application server.

In HPUX, a message broker makes use of the /var/wmqi/brokers/<broker> directory and the /var/wmqi/registry/<broker> directory, so these directories need to be on shared disks in addition to the directories used by the queue manager and broker database instance.

A broker runs as a pair of processes, called bipservice and bipbroker. The latter in turn creates the execution groups that run message flows. It is this collection of processes which are managed by MC/ServiceGuard. The configuration steps described in Chapter 4 describe how to use the example broker scripts to place a broker under MC/ServiceGuard control.

Figure 3. Broker group including local database

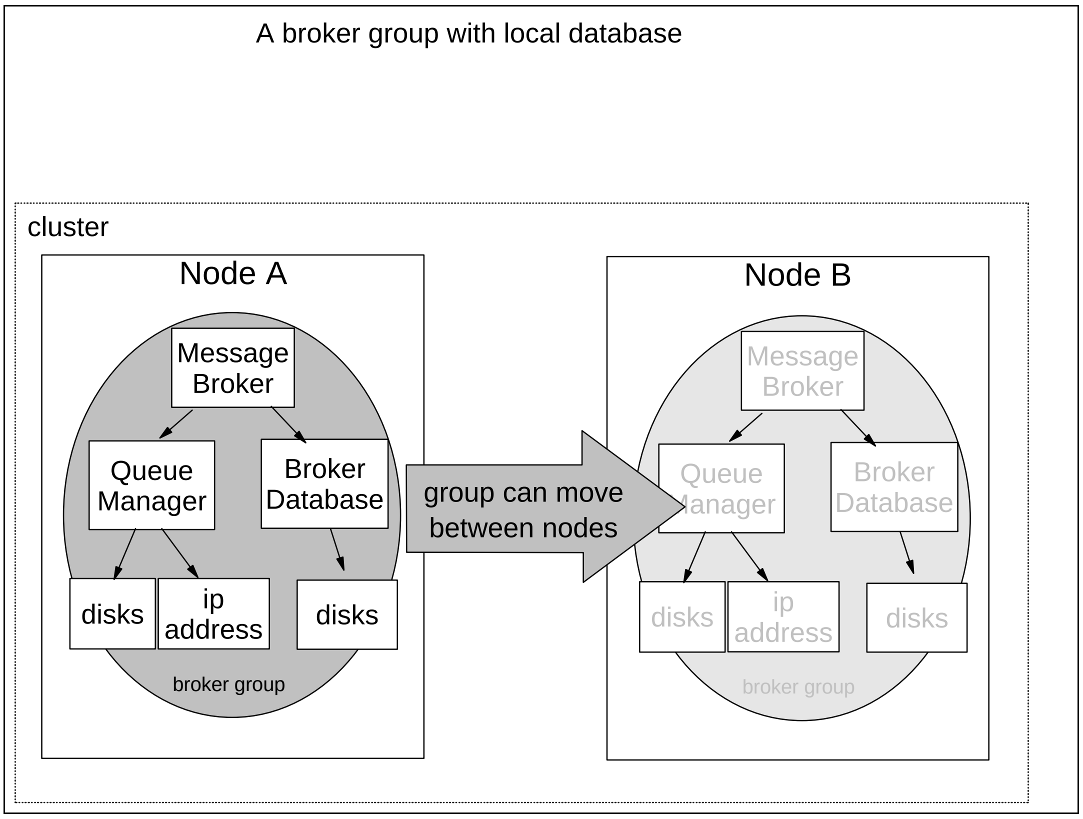
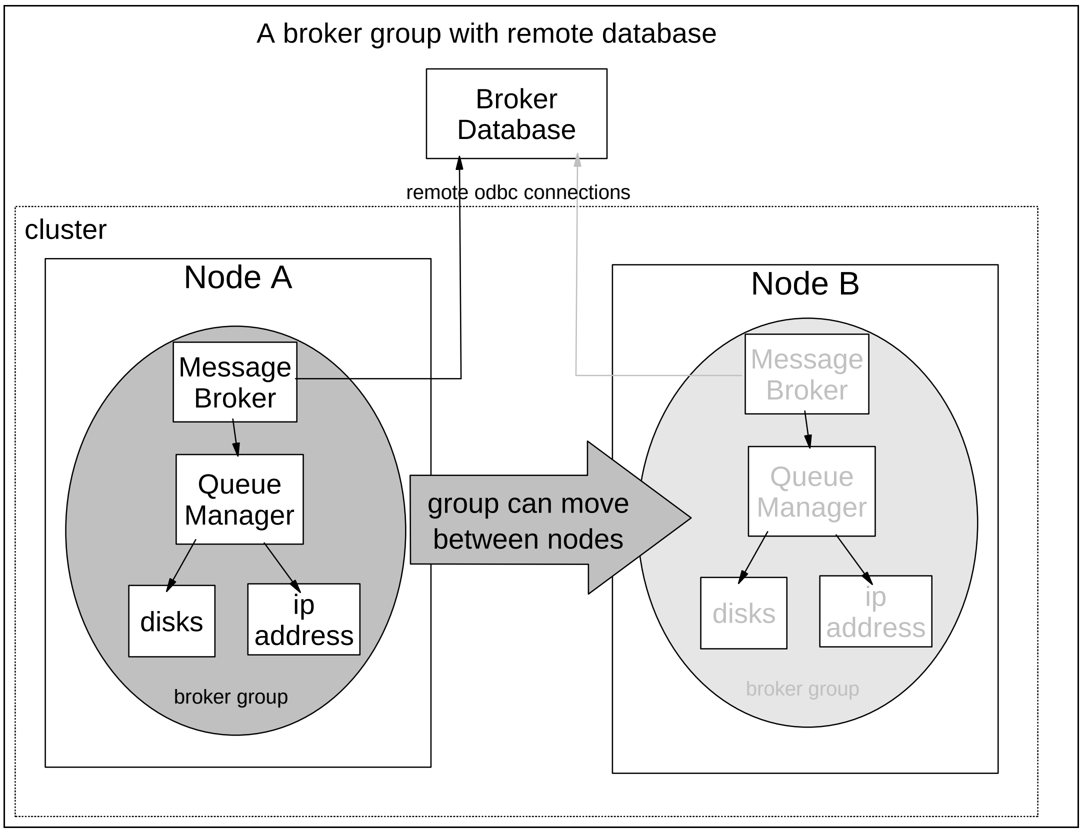


Figure 4. Broker group relying on remote database

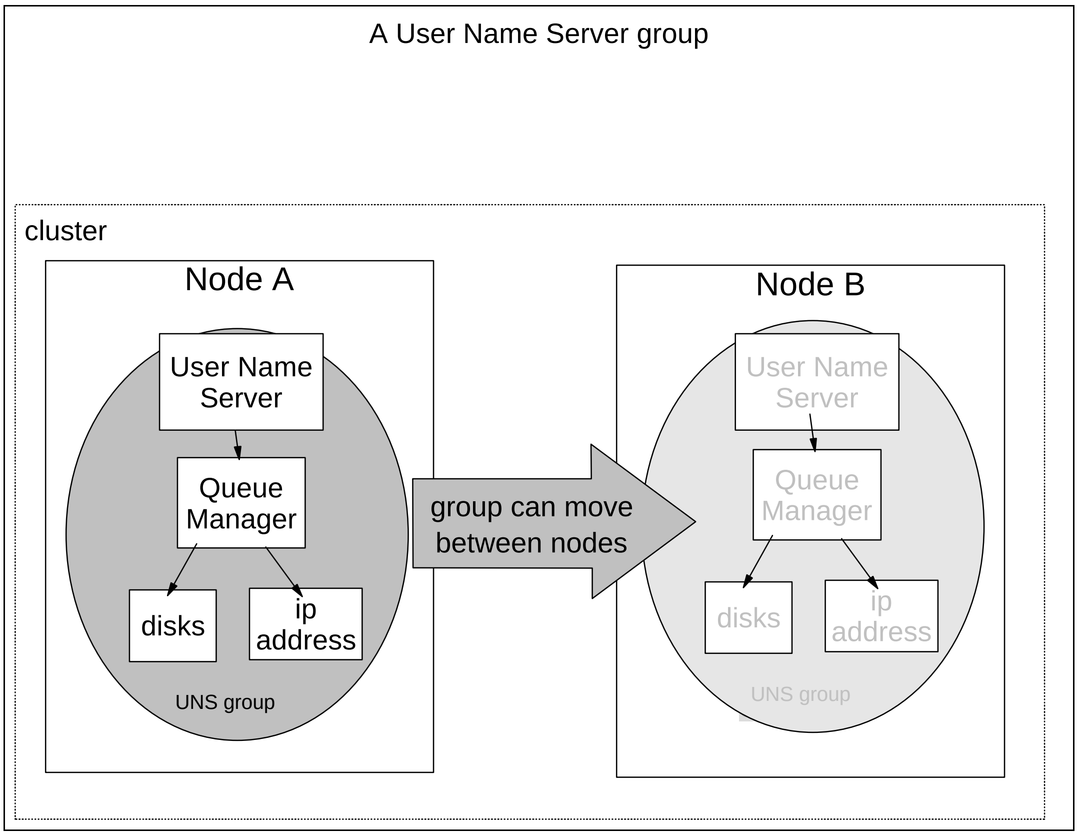


User Name Server

The User Name Server (UNS) provides a topic security service and is accessed via MQSeries. It must have a queue manager which is run on the same node as the UNS, and the queue manager will require an IP address and shared disk resources. The IP address and shared disk are separate from those described earlier for the broker.

Similar to the rationale for placing each message broker in a separate package, the most flexible configuration for the UNS is for it to be placed in a package of its own. This package must contain the UNS, it's queue manager and the queue manager's shared disks and IP address. Figure 5 shows the contents of a package containing the UNS.

Figure 5. UNS group



You *could* put the UNS in the same group as one or more brokers and it could share its queue manager with a broker, but this would bind the UNS permanently to that broker. For flexibility it is better to separate them. Users who wish to use application monitoring should also note the restriction that only one application server in a package can be monitored. If you were to place the UNS in the same group as one or more brokers and wanted to monitor all of them, you would need to edit the example scripts to put multiple components in an application server.

On HPUX, the UNS makes use of the /var/wmqi/brokers/UserNameServer directory and the /var/wmqi/registry/UserNameServer directory, so these directories need to be stored on shared disks in addition to the directories used by the queue manager.

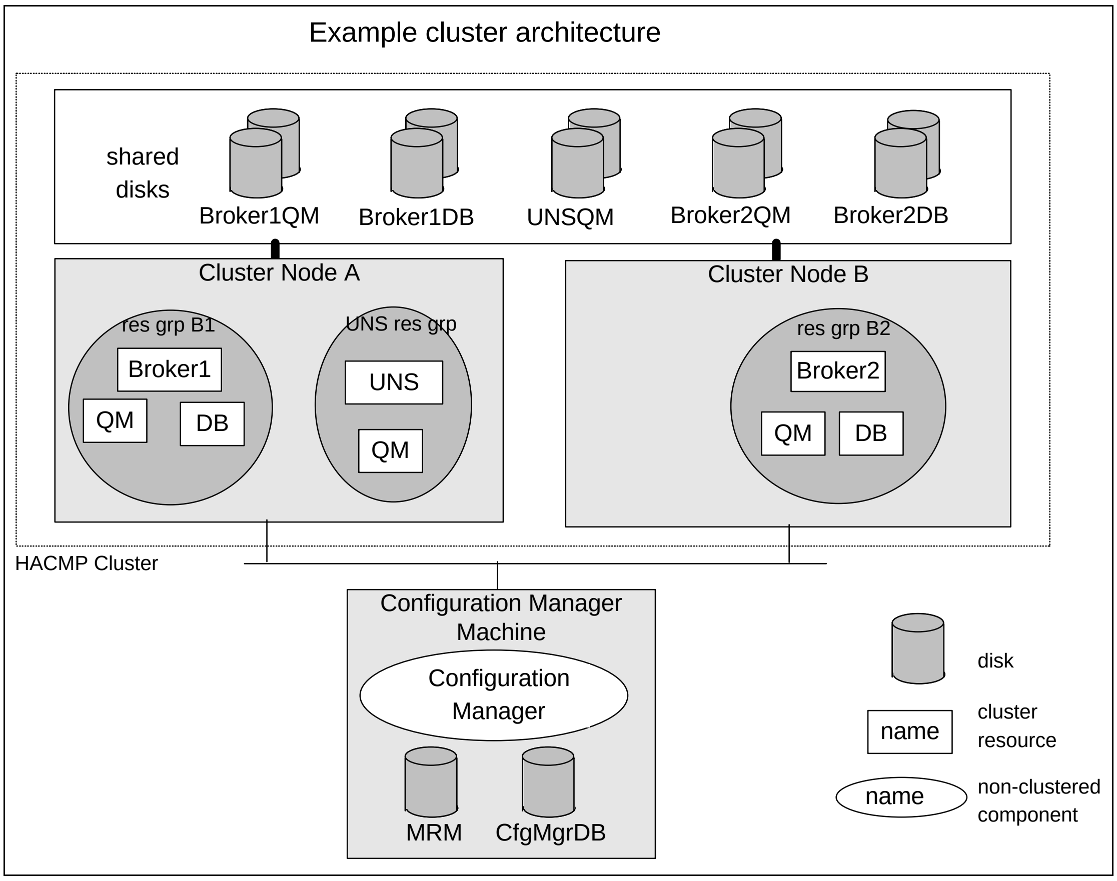
The UNS runs as a pair of processes, bipservice and bipuns. It is these processes that are managed by MC/ServiceGuard. The configuration steps described in Chapter 4 describe how to use the example UNS scripts to place a UNS under MC/ServiceGuard control.

Architectural guidelines

From the preceding discussion it is apparent that there are a number of choices as to which components your architecture will contain and which of them will be clustered. It is advisable to decide and record at this stage what the architecture will be. The following list provides a few suggested guidelines which may help in this. You don't have to adhere to them, but they may help. Figure 6 shows one possible architecture which implements the guidelines.

- Each broker should be in a separate package dedicated to that broker.
- Each broker must have its own queue manager, not used by any other brokers. The queue manager must be in the same package as the broker.
- Each broker should use a separate database instance.
- The broker database instance can be remote, in which case it should be run on a machine outside the cluster. Alternatively, the broker database instance can be run on the same machine as the broker, in which case it should be managed by MC/ServiceGuard and must be in the same package as the broker it serves.
- The UNS, if you have one, should be run in the MC/ServiceGuard Cluster and should be in a separate package.
- The UNS should have its own queue manager, which must be in the same package as the UNS.
- The Configuration Manager does not run on HP-UX so must be outside the MC/ServiceGuard Cluster. You could cluster it using other means.

Figure 6. An example cluster architecture



Chapter 4. Configuration steps

This chapter describes how to create and configure the components of the overall architecture that you designed in Chapter 3.

Configuration steps 1 to 3 create fundamental resources and must be done first.

Step 4 is optional, depending on whether you will use a UNS or not.

Step 5 describes how to create and cluster one message broker. For additional message brokers, repeat this step.

On completion of this chapter you should have an operational broker cluster.

Removal from the cluster and deletion of resources is covered in Chapter 5.

For all of these configuration steps you need to ensure that you are running as a user with sufficient privilege for each operation. On the cluster machines you will need root access.

Step 1. Create the Configuration Manager

If you already have a Configuration Manager installed, for control of existing brokers, then you could use that Configuration Manager to configure and deploy the highly available broker(s) as well. If you do not already have a Configuration Manager, or you wish to create an additional one for controlling the highly available broker(s), then perform this step.

As discussed in Chapter 3, the Configuration Manager is run on a machine outside the cluster. This step should be performed on that machine and creates the Configuration Manager in the normal manner.

Actions

1. MQSeries, DB2 and the WMQI Configuration Manager component must be installed on the machine.
2. On the machine where the Configuration Manager will be run, create the Configuration Manager's queue manager.
3. Create the Configuration Manager using the "mqsicreateconfigmgr" command, or the command assistant, as described in the WMQI Administration Guide.
4. Start the Configuration Manager and check whether it is available by using the Control Center GUI.

Step 2. Configure the cluster

It is assumed that you will create a single MC/ServiceGuard Cluster in which brokers and (optionally) the UNS will be run. You could put the brokers and UNS into separate clusters, in which case you should repeat this step for each cluster you wish to create.

Configuration of MC/ServiceGuard should be performed as described in the MC/ServiceGuard documentation.

Actions:

1. Configure TCP/IP on the cluster nodes for MC/ServiceGuard. Remember to configure ˜root/.rhosts, /etc/rc.net, etc.
2. Configure the cluster, cluster nodes and adapters to MC/ServiceGuard as usual.
3. Synchronise the Cluster Topology.
4. Now would be a good time to create and configure the user accounts that will be used to run the database instances, brokers and UNS. Home directories, (numeric) user ids, passwords, profiles and group memberships should be the same on all cluster nodes.

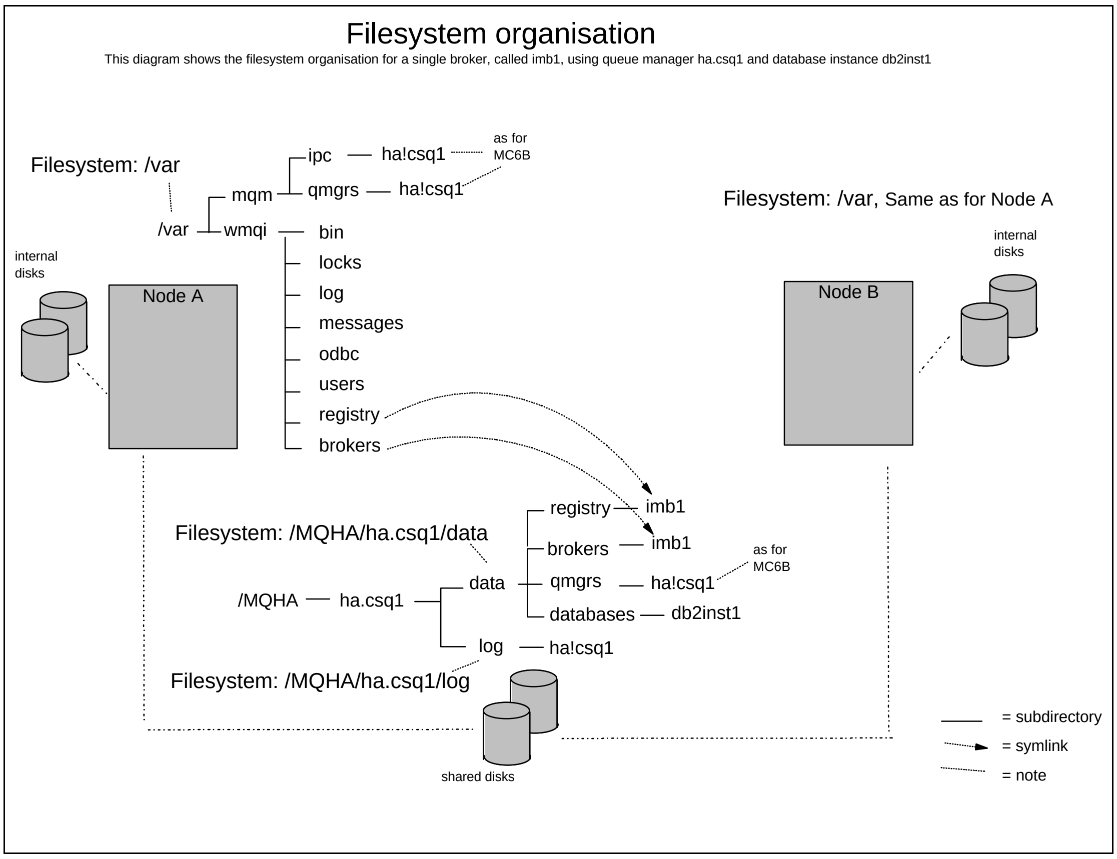
Step 3. Configure the shared disks

This step creates the volume group and filesystems required by a package containing an WMQI broker and its dependencies or the UNS and its dependencies. The suggested layout is based on the advice earlier that each broker or UNS should be put into a separate package. If you chose a different architecture in Chapter 3, then you may need to vary these instructions.

For each broker that you wish to run in the cluster, you also need to specify where the broker queue manager, the broker database instance and the broker itself should store their files. The UNS is similar to a broker except that it doesn't use a database. You need to create a volume group for each package. Within each volume group, you can decide where to place the files for the queue manager, database instance and broker, or queue manager and UNS, but the following is recommended:

- Queue manager - refer to SupportPac MC6B for details of how the queue manager's filesystems should be organised.
- Database instance - you can locate the database instance home directory for each broker's database instance under the data directory for the corresponding queue manager, in the same filesystem.
- Broker - a broker stores some information on disks. When you install WMQI on a node, it creates the /var/wmqi directory within the existing /var filesystem on internal disks. There is only one such directory per node, regardless of the number of brokers that the node may host. Within the /var/wmqi directory, there are broker specific directories which need to be on shared disk so that they can be moved to a surviving node in the event of a failure. The remainder of /var/wmqi, those parts which are not specific to one particular broker, should be on internal disks. The broker specific data stored on shared disk can be placed in the filesystem used for the queue manager data files, below the directory specified by the MQHAFSDATA environment variable used in SupportPac MC6B. The split between which broker directories are local and which are shared is shown in Figure 7, which should be read in conjunction with the similar diagram from SupportPac MC6B. The broker-related commands described later in this chapter set this up for you.
- UNS - the UNS directories should be organised in the same way as the broker directories with the exception that a UNS has no database. The UNS-related commands described later in this chapter set up the UNS directories for you.

Figure 7. Division of broker directories between local disks and shared disks



Actions:

1. For each broker, create a volume group that will be used for the queue manager's data and log files, the database instance and the broker-specific directories under /var/wmqi.
2. For the UNS, create a volume group that will be used for the queue manager's data and log files and the UNS directories under /var/wmqi.
3. For each volume group, create the data and log filesystems as described in SupportPac MC6B, Step 2. When choosing the path names for the filesystems you may prefer to use the name of the broker or the name "UNS" - instead of using the name of the queue manager.
4. For each node in turn, import each volume group and ensure that the filesystems can be mounted, then unmount the filesystems.

Step 4. Optional - Create a clustered UNS

If you plan to use a UNS within the broker domain then you have two choices, as discussed in Chapter 3. The UNS can either be run in an MC/ServiceGuard cluster, for example the one that contains the broker(s), or the UNS can be run elsewhere on a standalone machine.

Perform this step if you plan to cluster the UNS. If you plan to use a non-clustered UNS then create and configure it in the normal manner (i.e. as described in the WMQI Administration Guide) and proceed to Step 5. If you do not plan to use a UNS within the broker domain then proceed to Step 5.

A clustered UNS can be run in the same cluster as the broker or it can be run in a separate cluster. The same principles apply in both cases, regarding the relationship between the UNS, its queue manager and MC/ServiceGuard. If run in a separate cluster, then the UNS will be completely independent from the broker cluster and will require its own queue manager. If the UNS is put in the same cluster as the broker(s), then you have a choice as to how independent you make it. For maximum flexibility it is recommended that the UNS is in a separate package from any of the brokers in the cluster, allowing you to run the UNS and brokers on separate nodes. This requires that the UNS have a separate queue manager from any queue managers used by the brokers. The following instructions are written assuming that the UNS is being put in a separate package. If you were to decide to configure the UNS to use a queue manager that will also be used by a broker you would have to configure the UNS and broker to be in the same package.

Step 4a. Create the UNS queue manager

The UNS relies on a queue manager to receive requests from brokers. This step creates the queue manager.

During the creation of the UNS queue manager, you will create the package as described in SupportPac MC6B.

The UNS queue manager needs to be configured so that the UNS can communicate with the Configuration Manager. Broker to UNS communication is described in Step 5. Remember that because the UNS queue manager is clustered you need to use the service address for channels sending to the UNS queue manager rather than the machine IP address.

Actions:

1. On one node, create a clustered queue manager as described in SupportPac MC6B, using the hacrtmqm command. Use the volume group that you created for the UNS in Step 3 and place the volume group and queue manager into a package to which the UNS will be added in Step 4c. Don't configure the application server or application monitor described in SupportPac MC6B - you will create an application server that covers both the UNS and the queue manager, in Step 4c.
2. Set up queues and channels between the UNS queue manager and the Configuration Manager queue manager:
    a. On the Configuration Manager queue manager create a transmission queue for communication to the UNS queue manager. Ensure that the queue is given the same name and case as the UNS queue manager. The transmission queue should be set to trigger the sender channel.
    b. On the Configuration Manager queue manager create a sender and receiver channel for communication with the UNS queue manager. The sender channel should use the <u>service</u> address of the UNS package and the UNS queue manager's port number.
    c. On the UNS queue manager create a transmission queue for communication to the Configuration Manager queue manager. Ensure that the queue is given the same name and case as the Configuration Manager queue manager. The transmission queue should be set to trigger the sender channel.
    d. On the UNS queue manager create sender and receiver channels to match those just created on the Configuration Manager queue manager. The sender channel should use the IP address of the machine where the Configuration Manager queue manager runs, and the corresponding listener port number.
3. Test that the above queue managers can communicate regardless of which node owns the package for the UNS.

Step 4b. Create the UNS

The UNS stores data to disk in the /var/wmqi/brokers/UserNameServer directory and the /var/wmqi/registry/UserNameServer directory. These directories need to be on shared disk. This is handled for you if you create the UNS using the hamqsicreateusernameserver command, which accepts the same arguments as the mqsicreateusernameserver command. It creates the UNS and then moves the above directories to shared disk and creates symlinks to them from their usual paths. The hamqsicreateusernameserver command creates the UNS directories under the data path of the queue manager used by the UNS. This directory is the parent of the "qmgrs" directory on shared disk in which the queue manager exists.

Actions:

1. Create the UNS on the node hosting the package using the hamqsicreateusernameserver command.

hamqsicreateusernameserver command

The hamqsicreateusernameserver command will create the UNS and will ensure that its directories are arranged to allow for HA operation.

The hamqsicreateusernameserver command puts the UNS directories under the same path used for the data associated with the queue manager which the UNS uses. It parses the /var/mqm/mqs.ini file to locate this path information.

The invocation of the hamqsicreateusernameserver command uses exactly the same parameters that you would normally use for mqsicreateusernameserver.

You must be root to run this command.

Syntax

```
hamqsicreateusernameserver <creation parameters>
```

Parameters

- creation parameters - are exactly the same as for the regular WMQI mqsicreateusernameserver command

Example:

```
hamqsicreateusernameserver -i wmqi -a wmqi -q ha.csq1
```

2. Ensure that you can start and stop the UNS manually using the mqsistart and mqsistop commands. You will need to login as the user id under which the UNS runs to test this.

3. On any *other* nodes in the resource group's nodelist (i.e. excluding the one on which you just created the UNS), run the hamqsiaddunsstandby command to create the information needed by these nodes to enable them to host the UNS.

hamqsiaddunsstandby

The hamqsiaddunsstandby command will create the information required for a cluster node to act as a standby for the UNS. This command does not create the UNS - which is created as described earlier. This command defines symbolic links within subdirectories under /var/wmqi on the standby node which allow the UNS to move to that node.

The hamqsiaddunsstandby command expects the UNS directories to have been created by the hamqsicreateusernameserver command under the same path used for the data associated with the queue manager which the UNS uses. It parses the /var/mqm/mqs.ini file to locate this path information.

You must be root to run this command.

Syntax

```
hamqsiaddunsstandby <qm name> <userid>
```

Parameters

- qm name - the name of the queue manager used by the UNS
- userid - the account under which the UNS service runs

Example:

```
hamqsiaddunsstandby ha.csq1 wmqi
```

Step 4c. Place the UNS under cluster control

The UNS and its queue manager are managed by a single MC/ServiceGuard application server, which can be configured using the example UNS scripts supplied by this SupportPac. The UNS depends on the queue manager and the start and stop sequence is coded in the example scripts.

The UNS scripts are:

- **hamqsi_start_uns_as**
- **hamqsi_stop_uns_as**
- **hamqsi_start_uns**
- **hamqsi_stop_uns**

The hamqsi_start_uns_as and hamqsi_stop_uns_as scripts are the ones you configure as the start and stop methods for the application server. They invoke methods supplied by MC6B to control the queue manager and the hamqsi_start_uns and hamqsi_stop_uns methods to control the UNS.

If you are using MC/ServiceGuard then you can configure an application monitor which will monitor the health of the UNS and its queue manager. This will monitor the UNS and its queue manager and trigger recovery actions as a result of failures. Recovery actions include the ability to perform local restarts of the UNS and queue manager (see below) or to cause a failover of the package to another node. In MC/ServiceGuard you can only configure one application monitor per package.

When you ran hamqsicreateusernameserver an application monitor was created. This application monitor is specifically for monitoring an application server containing a UNS and queue manager and is called:

- **hamqsi_applmon.UNS**

If you configure MC/ServiceGuard to use the application monitor, it will call it periodically to check that the UNS processes and queue manager are running. The example application monitor checks that the bipservice process is running. The bipservice process monitors and restarts the bipuns process.

The fault monitoring interval is configured in MC/ServiceGuard, which also includes the parameters that control whether a failure of either component of the application server will trigger a restart. It is recommended that the restart count is set to 1 so that one restart is attempted, and that the time period is set to a small multiple of the expected start time for the components of the UNS group. With these settings, if successive restarts fail without a significant period of stability between, then the package will failover to a different node. Attempting more restarts on a node on which a restart has just failed is unlikely to succeed.

Actions:

1. Create an application server which will run the UNS and its queue manager using the example scripts provided in this SupportPac. The example scripts are called hamqsi_start_uns_as and hamqsi_stop_uns_as and are described in the following frames.

hamqsi_start_uns_as

The example start script is called hamqsi_start_uns_as. This script is robust in that it does not assume anything about the state of the UNS or queue manager on entry.

It accepts two command line parameters which are the queue manager name and the userid under which the UNS runs, so when you define the start command in MC/ServiceGuard, include the parameters.

Example

```
"/MQHA/bin/hamqsi_start_uns_as ha.csq1 wmqi"
```

hamqsi_stop_uns_as

The example stop script is called hamqsi_stop_uns_as.

The stop script accepts three command line parameters, the first is the queue manager name, the second parameter is the UNS userid and the third is the timeout (in seconds) to use on each of the levels of severity of stop. When you define the stop command in MC/ServiceGuard you should include the parameters.

Example

```
"/MQHA/bin/hamqsi_stop_uns_as ha.csq1 wmqi 10"
```

The stop command will use the timeout parameter as the time to allow either the queue manager or UNS to respond to an attempt to stop it. If a stop attempt times out, then a more severe stop is performed. The stop command has to ensure that the UNS and queue manager are both fully stopped by the time the command completes.

2. If you are using MC/ServiceGuard, then you can also specify an application monitor using the hamqsi_applmon.UNS script created by hamqsicreateusernameserver. An application monitor script cannot be passed parameters, so just specify the name of the monitor script. Also configure the other application monitor parameters, including the monitoring interval and the

restart parameters you require. The example application monitor script provided in this SupportPac is described in the following frame:

hamqsi_applmon.UNS

The hamqsi_applmon.UNS created for you by hamqsicreateusernameserver will be called at the polling frequency you specify to MC/ServiceGuard. It is a parameter-less wrapper script that calls hamqsi_monitor_uns_as which checks the state of the UNS and queue manager.

Success of both tests causes the application monitor to return a zero exit code to MC/ServiceGuard, indicating that the UNS and queue manager are working properly. Failure of either test will result in the application monitor returning a non-zero exit code to MC/ServiceGuard. MC/ServiceGuard will then take whatever action has been configured. If you wish to use the example application monitor then supply its name to MC/ServiceGuard for the UNS application server. The monitoring script accepts no parameters.

Example

```
"/MQHA/bin/hamqsi_applmon.UNS"
```

The example application monitor is tolerant if it finds that the queue manager is starting because this may be due to the stabilisation interval being too short.

3. Synchronise the cluster resources.
4. Ensure that the UNS and its queue manager are stopped, and start the application server.
5. Check that the UNS and queue manager started and test that the package can be moved from one node to the other and that the UNS runs correctly on each node.
6. Ensure that stopping the application server stops the UNS and its queue manager.
7. With the application server started, verify that the MC/ServiceGuard local restart capability is working as configured. During this testing a convenient way to cause failures is to identify the bipservice for the UserNameServer and kill it.

Step 5. Create and configure a broker

Perform this step for each broker that you plan to cluster.

A broker relies on a queue manager and a broker database. Each broker must have its own queue manager which is not shared with any other brokers, although a broker can share a queue manager with the UNS. For maximum flexibility it is recommended that each broker is placed in a separate package from any other brokers or the UNS. This requires that the broker does not share its queue manager with the UNS and that it has exclusive use of the database instance that contains the broker database. If you were to allow brokers to share a database instance, they would have to be in the same package. If you were to allow a broker and the UNS to share a queue manager, they would have to be in the same package. The following instructions are written assuming that the broker is being put in a separate package.

This step creates the queue manager and broker database instance components within a package. It then creates the broker and adds that to the package. Finally, an application server and application monitor are configured.

Step 5a. Create and configure the broker queue manager

A broker relies on a queue manager to receive and send messages and to communicate with other WMQI components. This step creates the queue manager.

During the creation of the broker queue manager, you will create the package as described in

The broker queue manager needs to be configured so that the broker can communicate with the Configuration Manager and UNS. The following actions are written assuming that the UNS is not sharing the broker queue manager. If the broker is sharing its queue manager with the UNS, then you can omit the creation of the relevant transmission queues and channels. If the broker is running in a collective then it will also need to communicate with other brokers and you should configure additional queues and channels for broker to broker communication. Remember that because the broker queue manager is clustered you need to use the service address for channels sending to the broker queue manager rather than the machine IP address.

Actions:

1. On one node, create a clustered queue manager as described in SupportPac MC6B, using the hacrtmqm command. Use the volume group that you created for the broker in Step 3 and place the volume group and queue manager into a package to which the broker will be added in Step 5d. Don't configure the application server or application monitor described in SupportPac MC6B - you will create an application server that covers the broker, queue manager and broker database instance, in Step 5d.
2. Set up queues and channels between the broker queue manager and the Configuration Manager queue manager:
    a. On the Configuration Manager queue manager create a transmission queue for communication to the broker queue manager. Ensure that the queue is given the same name and case as the broker queue manager. The transmission queue should be set to trigger the sender channel.

    b. On the Configuration Manager queue manager create a sender and receiver channel for communication with the broker queue manager. The sender channel should use the service address of the broker package and the broker queue manager's port number.

    c. On the broker queue manager create a transmission queue for communication to the Configuration Manager queue manager. Ensure that the queue is given the same name and case as the Configuration Manager queue manager. The transmission queue should be set to trigger the sender channel.

    d. On the broker queue manager create sender and receiver channels to match those just created on the Configuration Manager queue manager. The sender channel should use the IP address of the machine where the Configuration Manager queue manager runs, and the corresponding listener port number.
3. If you are using a UNS, set up queues and channels between the broker queue manager and the UNS queue manager:
    a. On the broker queue manager create a transmission queue for communication to the UNS queue manager. Ensure that the queue is given the same name and case as the UNS queue manager. The transmission queue should be set to trigger the sender channel.

    b. On the broker queue manager create a sender and receiver channel for communication with the UNS queue manager. If the UNS is clustered, the sender channel should use the service address of the UNS package and the UNS queue manager's port number.

    c. On the UNS queue manager create a transmission queue for communication to the broker queue manager. Ensure that the queue is given the same name and case as the broker queue manager. The transmission queue should be set to trigger the sender channel.

    d. On the UNS queue manager create a sender and receiver channel for communication with the broker queue manager, with the same names as the receiver and sender channel just created on the broker queue manager. The sender channel should use the service address of the broker package and the broker queue manager's port number.
4. Test that the above queue managers can communicate regardless of which node owns the resource groups they belong to.

Step 5b. Create and configure the broker database

The message broker relies on a broker database and in this description it is assumed that this is running under DB2. This step creates the database instance which the broker will use. The instance runs the broker database in which the broker tables are created when the broker is created. As discussed in the start of Chapter 3, there are two options regarding where the broker database is run, either inside or outside the cluster. If you choose to run the database outside the cluster then simply follow the instructions in the WMQI documentation for creating the broker database but ensure that you consider whether the database is a single point of failure and make appropriate provision for the availability of the database. If you are configuring the database inside the cluster (recommended) then follow the instructions in the remainder of this step.

The database instance is the unit of failover of the database manager.

The database instance is made highly available by invoking its HA scripts from within the scripts you configure for the broker resource group's application server. If you are using DB2 , then scripts can be found on the DB2 web site in appendix 5 of the “IBM DB2 EE v.7.1 Implementation and Certification with MC/ServiceGuard High Availability Software” document. If you are using a different database manager then follow the instructions provided with that database manager. The example scripts supplied in this SupportPac for the broker application server (described fully in Step 5d) include calls to the DB2 Version 7.1 scripts. If you are using a different database manager then edit the scripts accordingly.

Actions:

1. Create a database instance home directory in the volume group owned by the package. As portrayed in Figure 7, this can be in the queue manager's data path in the "databases" directory.
2. Create a database instance owner user, specifying the home directory just created.
3. Create the database instance.
4. Start the instance and create and configure the broker database as described in the WMQI documentation, including creation of an ODBC data source for it, defined on all cluster nodes that may host the package.
5. Don't create an application server for the database instance - it will be included in the application server which you will create in Step 5d.
6. Ensure that the database instance runs correctly on each node the package can move to. You will need to manually start and stop the database instance to test this.

Step 5c. Create the message broker

A broker stores data to disk in the /var/wmqi/brokers/<broker> directory and the /var/wmqi/registry/<broker> directory. These directories need to be on shared disk. This is handled for you if you create the broker using the hamqsicreatebroker command, which accepts the same arguments as the mqsicreatebroker command. It creates the broker and then moves the above directories to shared disk and creates symlinks to them from their usual paths. The hamqsicreatebroker command creates the broker directories under the data path of the queue manager used by the broker. This directory is the parent of the "qmgrs" directory on shared disk in which the queue manager exists.

Actions:

1. Create the broker on the node hosting the logical host using the hamqsicreatebroker command.

hamqsicreatebroker command

The hamqsicreatebroker command will create the broker and will ensure that its directories are arranged to allow for HA operation.

The hamqsicreatebroker command puts the broker directories under the same path used for the data associated with the queue manager which the broker uses. It parses the /var/mqm/mqs.ini file to locate this path information.

The invocation of the hamqsicreatebroker command uses exactly the same parameters that you would normally use for mqsicreatebroker.

You must be root to run this command.

Syntax

```
hamqsicreatebroker <creation parameters>
```

Parameters

- creation parameters - are exactly the same as for the regular WMQI mqsicreatebroker command

Example:

```
hamqsicreatebroker imb1 -i wmqi -a wmqi -q ha.csq1 -n
IMB1DB
```

2. Ensure that you can start and stop the broker manually using the mqsistart and mqsistop commands. You will need to login as the user id under which the UNS runs to test this.

3. On any *other* nodes in the resource group's nodelist (i.e. excluding the one on which you just created the broker), run the hamqsiaddbrokerstandby command to create the information needed by these nodes to enable them to host the broker.

hamqsiaddbrokerstandby

The hamqsiaddbrokerstandby command will create the information required for a cluster node to act as a standby for the broker. This command does not create the broker - which is created as described earlier. This command defines symbolic links within subdirectories under /var/wmqi on the standby node which allow the broker to move to that node.

The hamqsiaddbrokerstandby command expects the broker directories to have been created by the hamqsicreatebroker command under the same path used for the data associated with the queue manager which the broker uses. It parses the /var/mqm/mqs.ini file to locate this path information.

You must be root to run this command.

Syntax

```
hamqsiaddbrokerstandby <broker> <qm> <userid>
```

Parameters

- broker - the name of the broker
- qm - the name of the queue manager used by the broker
- userid - the account under which the broker service runs

Example:

```
hamqsiaddbrokerstandby imb1 ha.csq1 wmqi
```

Step 5d. Place the broker under cluster control

The broker, queue manager and database instance are managed by a single MC/ServiceGuard application server, which can be configured using the example broker scripts supplied by this SupportPac. The broker depends on the queue manager and the database instance and the start and stop sequence is coded in the example scripts.

The broker scripts are:

- **hamqsi_start_broker_as**
- **hamqsi_stop_broker_as**
- **hamqsi_start_broker**
- **hamqsi_stop_broker**

The hamqsi_start_broker_as and hamqsi_stop_broker_as scripts are the ones you configure as the start and stop methods for the application server. They invoke methods supplied by SupportPac MC6B to control the queue manager and invoke the database scripts to control the database instance. In addition they invoke the hamqsi_start_broker and hamqsi_stop_broker methods to control the broker.

If you are using MC/ServiceGuard then you can configure an application monitor which will monitor the health of the broker, queue manager and database instance. The application monitor will assess the state of these components and trigger recovery actions as a result of failures. Recovery actions include the ability to perform local restarts of the broker and its dependencies (see below) or to cause a failover of the package to another node. In MC/ServiceGuard you can only configure one application monitor per package.

When you ran hamqsicreatebroker an application monitor was created. This application monitor is specifically for monitoring an application server containing a broker, a queue manager and a database instance and is called:

- **hamqsi_applmon.<broker>**

where <broker> is the name of the broker. If you configure MC/ServiceGuard to use the application monitor, it will call it periodically to check that the broker, queue manager and database instance are running. The example application monitor checks that the bipservice process is running. The bipservice process monitors and restarts the bipbroker process. The bipbroker process monitors and restarts DataFlowEngines. The application monitor does not check for DataFlowEngines because you may have none deployed. If you wish to monitor for these as well, then customise the example hamqsi_monitor_broker_as script, but remember that depending on how you customise it, you may have to suspend monitoring if you wish to deploy a removal of a DataFlowEngine, otherwise it would be classed as a failure.

The fault monitoring interval is configured in MC/ServiceGuard, which also includes the parameters that control whether a failure of either component of the application server will trigger a restart. It is recommended that the restart count is set to 1 so that one restart is attempted, and that the time period is set to a small multiple of the expected start time for the components of the broker group. With these settings, if successive restarts fail without a significant period of stability between, then the package will failover to a different node. Attempting more restarts on a node on which a restart has just failed is unlikely to succeed.

Actions:

1. Create an application server which will run the broker, its queue manager and the database instance, using the example scripts provided in this SupportPac. The example scripts are called hamqsi_start_broker_as and hamqsi_stop_broker_as and are described in the following frames.

hamqsi_start_broker_as

The example start script is called hamqsi_start_broker_as. This script is robust in that it does not assume anything about the state of the broker, queue manager or database instance on entry.

It accepts command line parameters which provide the name of the broker, the queue manager, the userid under which the broker runs and the names of the database instance and database. When you define the start command in MC/ServiceGuard, include the parameters.

Example

```
"/MQHA/bin/hamqsi_start_broker_as imb1 ha.csq1 wmqi
db2inst1 IMB1DB"
```

hamqsi_stop_broker_as

The example stop script is called hamqsi_stop_broker_as.

The stop script accepts command line parameters that provide the name of the broker, the queue manager name, the broker userid, the database instance name and a timeout (in seconds) to use on each of the levels of severity of stop. When you define the stop command in MC/ServiceGuard you should include the parameters.

Example

```
"/MQHA/bin/hamqsi_stop_broker_as imb1 ha.csq1 wmqi
db2inst1 10"
```

The stop command will use the timeout parameter as the time to allow either the queue manager or broker to respond to an attempt to stop it. If a stop attempt times out, then a more severe stop is performed. The stop command has to ensure that the broker and queue manager are both fully stopped by the time the command completes.

2. If you are using MC/ServiceGuard, then you can also specify an application monitor using the hamqsi_applmon.<broker> script created by hamqsicreatebroker. An application monitor script cannot be passed parameters, so just specify the name of the monitor script. Also configure the other application monitor parameters, including the monitoring interval and the restart parameters you require. The example application monitor script provided in this SupportPac is described in the following frame:

hamqsi_applmon.<broker>

The hamqsi_applmon.<broker> created for you by hamqsicreatebroker will be called at the polling frequency you specify to MC/ServiceGuard. It is a parameter-less wrapper script that calls hamqsi_monitor_broker_as which checks the state of the broker, the queue manager and the database instance.

A successful test of all three components causes the application monitor to return a zero exit code to MC/ServiceGuard, indicating that the components are working properly. Failure of any component test will result in the application monitor returning a non-zero exit code to MC/ServiceGuard. MC/ServiceGuard will then take whatever action has been configured. If you wish to use the example application monitor then supply its name to MC/ServiceGuard for the broker application server. The monitoring script accepts no parameters.

Example

```
"/MQHA/bin/hamqsi_applmon.<broker>"
```

The example application monitor is tolerant if it finds that the queue manager is starting because this may be due to the stabilisation interval being too short.

3. Synchronise the cluster resources.
4. Ensure that the broker, queue manager and database instance are stopped, and start the application server.
5. Check that the components started and test that the package can be moved from one node to the other and that they run correctly on each node.
6. Ensure that stopping the application server stops the components.

7. With the application server started, verify that the MC/ServiceGuard local restart capability is working as configured. During this testing a convenient way to cause failures is to identify the bipservice for the broker and kill it.

Chapter 5. Removal and deletion steps

Removal and deletion are defined as follows:

Removal of a component from the cluster configuration returns the component to manual control, rather than it being controlled and monitored by the cluster. This does not destroy the components, which should continue to function normally, but under manual control. Once removed from the cluster configuration the component will not be monitored for failures and will remain on one node.

When a component is being removed from the cluster, the application server which manages it should be stopped. The configuration can then be changed and synchronised.

Deletion of a component destroys the component. It is recommended that you remove a component from cluster control before deleting it. A component can only be deleted on the node which is hosting it.

The following activities are related to removal and deletion of components.

- Removal of the UNS from the cluster configuration
- Removal of UNS standby information from a node
- Deletion of the UNS component
- Removal of a broker from the cluster configuration
- Removal of a broker standby information from a node
- Deletion of a broker component

Step 6. Remove the UNS from the cluster configuration

As described previously, removal of a component from the cluster configuration returns the component to manual control. If the UNS is removed from the cluster configuration, it can continue to operate, but must be started and stopped manually.

When removing the UNS from the cluster configuration remember that it relies on a queue manager. If you intend to continue to use the UNS, it would be inadvisable to remove it from cluster control and leave the queue manager under cluster control. For details of how to remove the queue manager from cluster control, refer to SupportPac MC6B, Step 7.

Actions:

1. Stop the application server which runs the UNS.
2. Delete the application monitor, if configured.
3. Delete the application server. If you wish to retain the queue manager under cluster control then you may decide to replace the application server with one that uses the scripts from SupportPac MC6B.
4. If you have no further use for them, remove the filesystem, service label and volume group resources from the package and delete the group.
5. Synchronise the cluster resources configuration.

Step 7. Remove UNS standby information from a node

With the UNS removed from cluster control, as described in Step 6, it is now safe to remove the UNS information from any standby nodes.

This standby information was created when you ran the hamqsiaddunsstandby command - see Step 4b. There is a companion command called hamqsiremoveunsstandby, which removes the standby information.

Actions:

1. Ensure that the UNS has been removed from cluster control and identify which node it is running on.
2. Identify which *other* nodes have standby information on them. If you are not sure whether a node has standby information then look for a symbolic link called /var/wmqi/brokers/UserNameServer. If a node has such a link, then it is a standby node.
3. On the standby nodes, run the hamqsiremoveunsstandby command.

hamqsiremoveunsstandby command

The hamqsiremoveunsstandby command will remove the standby information from standby nodes for the UNS. This will remove the symlinks for the UNS from the subdirectories under the /var/wmqi directory.

You must be root to run this command.

Syntax

```
hamqsiremoveunsstandby
```

Parameters

- none

Step 8. Delete the UNS

When the UNS has been removed from cluster control, it is possible to delete it. The UNS was created by the hamqsicreateusernameserver command, in such a way that it is amenable to HA operation. There is a companion command called hamqsideleteusernameserver which is aware of the changes made for HA operation and which should always be used to delete the UNS. This companion command reverses the HA changes and then deletes the UNS.

This destroys the UNS.

Actions:

1. Delete the UNS from the Configuration Manager and deploy the changes as described in the WMQI Administration Guide.
2. Identify the node on which the UNS is defined and on that node ensure that the UNS is stopped. If it is not, then issue the mqsistop UserNameServer command to stop it.
3. On the same node, as root, run the hamqsideleteusernameserver command.

hamqsideleteusernameserver command

The hamqsideleteusernameserver command will delete the UNS. This will destroy its control files and remove the definition of the broker from the /var/wmqi directory. This is similar to the behaviour of the mqsideleteusernameserver command, which the command uses internally.

You must be root to run this command.

Syntax

```
hamqsideleteusernameserver <userid>
```

Parameters

- userid - the userid under which the UNS service runs

Step 9. Remove a broker from the cluster configuration

As described previously, removal of a component from the cluster configuration returns the component to manual control. If the broker is removed from the cluster configuration, it can continue to operate, but must be started and stopped manually.

When removing the broker from the cluster configuration remember that it relies on a queue manager and database instance. If you intend to continue to use the broker, it would be inadvisable to remove it from cluster control and leave the queue manager or database instance under cluster control. For details of how to remove the queue manager from cluster control, refer to SupportPac MC6B, Step 7. For details of how to remove the database instance, refer to the database manager documentation.

Actions:

1. Stop the application server which runs the broker.
2. Delete the application monitor, if configured.
3. Delete the application server. If you wish to retain the queue manager or database instance under cluster control then you may decide to replace the application server with one that uses the scripts from SupportPac MC6B or the database scripts.
4. If you have no further use for them, remove the filesystem, service label and volume group resources from the package and delete the group.
5. Synchronise the cluster resources configuration.

Step 10. Remove broker standby information from a node

With the broker removed from cluster control, as described in Step 9, it is now safe to remove the broker information from any standby nodes.

This standby information was created when you ran the hamqsiaddbrokerstandby command - see Step 5c. There is a companion command called hamqsiremovebrokerstandby, which removes the standby information.

Actions:

1. Ensure that the broker has been removed from cluster control and identify which node it is running on.
2. Identify which *other* nodes have standby information on them. If you are not sure whether a node has standby information then look for a symbolic link called /var/wmqi/brokers/<broker>, where <broker> is the name of the broker. If a node has such a link, then it is a standby node.
3. On the standby nodes, as root, run the hamqsiremovebrokerstandby command.

hamqsiremovebrokerstandby command

The hamqsiremovebrokerstandby command will remove the standby information from standby nodes for a broker. This will remove the symlinks for the broker from the subdirectories under the /var/wmqi directory.

You must be root to run this command.

Syntax

```
hamqsiremovebrokerstandby <broker name>
```

Parameters

- broker name - the name of the broker to be removed

Step 11. Delete a broker

When a broker has been removed from cluster control, it is possible to delete it. The broker was created by the hamqsicreatebroker command, in such a way that it is amenable to HA operation. There is a companion command called hamqsideletebroker which is aware of the changes made for HA operation and which should always be used to delete the broker. This companion command reverses the HA changes and then deletes the broker.

This destroys the broker.

Actions:

1. Delete the broker from the Configuration Manager and deploy the changes as described in the WMQI Administration Guide.
2. Identify the node on which the broker is defined and on that node ensure that the broker is stopped. If it is not, then issue the mqsistop <broker> command to stop it, where <broker> is the name of the broker.
3. On the same node, as root, run the hamqsideletebroker command.

hamqsideletebroker command

The hamqsideletebroker command will delete a broker. This will destroy its control files and remove the definition of the broker from the /var/wmqi directory. This is similar to the behaviour of the mqsideletebroker command, which the hamqsideletebroker command uses internally.

You must be root to run this command.

Syntax

```
hamqsideletebroker <broker name> <userid> <qm>
```

Parameters

- broker name - the name of the broker to be deleted
- userid - the userid under which the broker service runs
- qm - the name of the queue manager used by the broker

Comments

If you have any comments on this SupportPac, please send them to:

Email:

aford@uk.ibm.com
wallisgd@hursley.ibm.com

Post:

Andrew Ford
MailPoint 211
IBM UK Laboratories Ltd
Hursley Park
Winchester
Great Britain
SO21 2JN

Graham Wallis
MailPoint 211
IBM UK Laboratories Ltd
Hursley Park
Winchester
Great Britain
SO21 2JN