High Availability in Message Queue 4.1

Message Queue 4.1 introduces a new type of broker cluster, a *high-availability (HA) cluster.* (The old-style *conventional clusters* continue to be supported as well.) HA clusters provide an even greater degree of service availability than conventional ones: if one of the brokers within the cluster should fail, another can take over ownership of its pending messages and see that they are delivered to their destinations without interruption of service. The purpose of this paper is to describe this new capability and the new and modified Message Queue features that support it.

High-Availability Clusters

High-availability clusters are distinguished from the conventional type by a new configuration property, `imq.cluster.ha` (see Table 6 on page 13), which is set to `true` for brokers belonging to an HA cluster and `false` otherwise. The two types differ in that all brokers in an HA cluster share the same JDBC-based persistent store, rather than each having its own store. This eliminates the need for a master broker to track the cluster's persistent state. The shared store holds pending messages, destination definitions, information on durable subscriptions, and the dynamic state of acknowledgments and transactions. In the event of broker failure, this enables another broker to assume ownership of the failed broker's persistent state and provide uninterrupted service to its clients.

Cluster Configuration

HA clusters are self-configuring: any broker configured to use the cluster's shared store is automatically registered as part of the cluster at startup, without further action on the client's or administrator's part. This eliminates the need to explicitly specify a broker list (`imq.cluster.brokerlist`), as is necessary for conventional clusters. Similarly, the concept of a master broker is inapplicable to HA clusters. Consequently the `imq.cluster.brokerlist` and `imq.cluster.masterbroker` configuration properties are not supported for HA clusters.

Conversely, two new configuration properties are added for configuring HA clusters:

- `imq.cluster.clusterid` (Table 6 on page 13) gives the *cluster identifier,* which will be appended to the names of all database tables in the cluster's shared persistent store. The value of this property must be the same for all brokers in a given cluster, but must be unique for each cluster: no two running clusters may have the same cluster identifier.
- `imq.brokerid` (Table 4 on page 11) specifies a unique *broker identifier* for each broker in the cluster; this value must be different for each broker.

The existing cluster configuration properties `imq.cluster.hostname` and `imq.cluster.port` can be set independently for each individual broker, but `imq.cluster.transport` and `imq.cluster.url` must have the same values for all brokers in the cluster.

Persistent Store Configuration

The persistent data store for an HA cluster is maintained on a high-availability database server, using the Java Database Connectivity (JDBC) API. All brokers belonging to such a cluster must therefore have their `imq.persist.store` property set to `jdbc`.

The database server may be Sun's own High Availability Database (HADB) server, or it may be a third-party product such as Oracle's Real Application Clusters (RAC) or DataMirror Corporation's PointBase. A new configuration property, `imq.persist.jdbc.dbVendor` (see Table 5 on page 12), specifies the name of the database vendor. As shown in Table 5, all other JDBC-related property names have been modified to include this vendor name: for instance, when using Sun's HADB for the HA server, the Java class name of the JDBC driver is now specified by the property `imq.persist.jdbc.hadb.driver` rather than `imq.persist.jdbc.driver` as

previously. (This change in the names of the JDBC properties now applies to all brokers, conventional as well as HA; the former names without the vendor name are deprecated.) All brokers in a cluster must have the same values for all such properties.

Another new property, `imq.persist.jdbc.`*`vendorName`*`.property.`*`propName`*, has been added for specifying any additional, vendor-specific properties that a particular database server may require.

NOTE If the integration between Message Queue and Application Server is local (that is, there is a one-to-one relationship between Application Server instances and Message Queue message brokers), the Application Server will automatically propagate these properties to each broker in the HA cluster. However, if the integration is remote (a single Application Server instance using an externally configured broker cluster), then it is the Message Queue administrator's responsibility to configure the needed properties manually for each broker.

After setting all of the needed JDBC configuration properties for the brokers in an HA cluster, you must also install your JDBC driver's `.jar` file in the appropriate directory location, depending on your operating-system platform (as listed in Appendix A of the *Message Queue Administration Guide*) and then execute the Database Manager command

```
imqdbmgr create tbl
```

to create the database schema for the HA persistent data store.

Failure Detection and Takeover Properties

Brokers within an HA cluster periodically inform each other that they are still in operation by exchanging *heartbeat packets* and updating their state information in the cluster's shared persistent store. Several new configuration properties (summarized in Table 6 on page 13) govern the exchange of heartbeat packets.

The `imq.cluster.heartbeat.hostname` and `imq.cluster.heartbeat.port` properties give the host name (or IP address) and port number for the heartbeat connection service. `imq.cluster.heartbeat.interval` defines the interval, in seconds, at which heartbeat packets are transmitted; `imq.cluster.heartbeat.threshold` specifies the number of missed heartbeat intervals after which a broker is considered suspect of failure.

When no heartbeat has been detected from a given broker for the specified number of heartbeat intervals, the other brokers in the cluster begin to monitor the suspect broker's state information in the persistent store to confirm whether the broker has indeed failed. The broker's state is then monitored at intervals defined by the `imq.cluster.monitor.interval` property. If the broker fails to update its state within the number of monitor intervals specified by `imq.cluster.monitor.threshold`, the broker is considered to have failed.

On detecting that another broker within the cluster has failed, the broker detecting the failure attempts to take over the failed broker's persistent state (pending messages, destination definitions, durable subscriptions, pending acknowledgments, and open transactions) in order to provide uninterrupted service to the failed broker's clients. If two or more brokers attempt such takeover, only the first will succeed; that broker acquires a lock on the failed broker's data in the persistent store, preventing subsequent takeover attempts by other brokers from succeeding. After an initial waiting period, the takeover broker will then clean up any transient resources (such as transactions and temporary destinations) belonging to the failed broker; these resources will be unavailable if the client later reconnects.

Managing High-Availability Clusters

This section presents step-by-step procedures for performing a variety of administrative tasks for a high-availability cluster:

Clustering High-Availability Brokers

Because high-availability brokers are self-configuring, there is no need to explicitly specify the list of brokers to be included in the cluster, as is done with conventional clusters. Instead, all that is needed is to set each broker's configuration properties appropriately and then start the broker; as long as its properties are set properly, it

will automatically be incorporated into the cluster. Table 1 shows the required settings. In addition, there may be vendor-specific settings required for a particular vendor's database; Tables 2 and 3 show these vendor-specific settings for Sun's own HADB and for MySQL from MySQLAB, respectively.

Table 1 Required Configuration Properties for HA Clusters

Property	Required Value	Description
`imq.cluster.ha`	`true`	Broker is part of an HA cluster
`imq.cluster.clusterid`		Cluster identifier Must be the same for all brokers in the cluster.
`imq.brokerid`		Broker identifier Must be different for each broker in the cluster.
`imq.persist.store`	`jdbc`	Model for persistent data storage Only JDBC-based persistence is supported for HA data stores.
`imq.persist.jdbc.dbVendor`		Name of database vendor for persistent data store: `hadb` HADB (Sun Microsystems, Inc.) `derby` Derby (Apache Software Foundation) `pointbase` PointBase (DataMirror Corporation) `oracle` Oracle Real Application Cluster (Oracle Corporation) `mysql` MySQL (MySQLAB)

Table 2 Vendor-Specific Configuration Properties for HADB Database

Property	Description
`imq.persist.jdbc.hadb.user`	User name for opening database connection
`imq.persist.jdbc.hadb.password`	Password for opening database connection

`imq.persist.jdbc.hadb.property.serverList`	JDBC URL of database
	Use the command
	`hadbm get JdbcURL`
	to get the URL; remove the prefix
	`jdbc:sun:hadb`
	and use
	host`:`*port*`,`*host*`:`*port*`...`
	for the property value.

Table 3 Vendor-Specific Configuration Properties for MySQL Database

Property	Description
`imq.persist.jdbc.mysql.user`	User name for opening database connection
`imq.persist.jdbc.mysql.password`	Password for opening database connection
`imq.persist.jdbc.mysql.property.url`	JDBC URL for opening database

The property values can be set separately in each broker's instance configuration file, or they can be specified in a cluster configuration file that all the brokers share. The procedures are as follows:

➤ To Cluster HA Brokers Using Instance Configuration Files

1. **For each broker in the cluster:**

   a. **Start the broker with the `imqbrokerd` command.**

      The first time a broker instance is run, an instance configuration file (`config.properties`) is automatically created.

   b. **Shut down the broker.**

      Use the `imqcmd shutdown bkr` command.

   c. **Edit the instance configuration file to specify the broker's HA-related configuration properties.**

      Table 1 shows the required property values.

   d. **Specify any additional, vendor-specific properties that may be needed.**

      Tables 2 and 3 show the required properties for HADB and MySQL databases, respectively.

2. **Place a copy of, or a symbolic link to, your JDBC driver's `.jar` file in the appropriate location, depending on your platform:**

   Solaris: `/usr/share/lib/imq/ext/`
   Linux: `/opt/sun/mq/share/lib/`
   Windows: `IMQ_VARHOME\\lib\\ext`

3. **Create the database schema needed for Message Queue persistence.**

   Use the `imqdbmgr create tbl` command; see Table 5 on page 12.

4. **Restart each broker with the `imqbrokerd` command.**

   The brokers will automatically register themselves into the cluster on startup.

➤ To Cluster HA Brokers Using a Cluster Configuration File

An alternative method, better suited for production systems, is to use a cluster configuration file to specify the composition of the cluster:

1. **Create a cluster configuration file specifying the cluster's HA-related configuration properties.**

   Table 1 shows the required property values. However, do *not* include the `imq.brokerid` property in the cluster configuration file; this must be specified separately for each individual broker in the cluster.

2. **Specify any additional, vendor-specific properties that may be needed.**

   Tables 2 and 3 show the required properties for HADB and MySQL databases, respectively.

3. **For each broker in the cluster:**

   a. **Start the broker with the `imqbrokerd` command.**

      The first time a broker instance is run, an instance configuration file (`config.properties`) is automatically created.

   b. **Shut down the broker.**

      Use the `imqcmd shutdown bkr` command.

   c. **Edit the instance configuration file to specify the location of the cluster configuration file.**

      In the broker's instance configuration file, set the `imq.cluster.url` property to point to the location of the cluster configuration file you created in step 1.

   d. **Specify the broker identifier.**

      Set the `imq.brokerid` property in the instance configuration file to the broker's unique broker identifier. This value must be different for each broker.

4. **Place a copy of, or a symbolic link to, your JDBC driver's `.jar` file in the appropriate location, depending on your platform:**

   Solaris: `/usr/share/lib/imq/ext/`
   Linux: `/opt/sun/mq/share/lib/`
   Windows: `IMQ_VARHOME\\lib\\ext`

5. **Create the database schema needed for Message Queue persistence.**

   Use the `imqdbmgr create tbl` command; see Table 5 on page 12.

6. **Restart each broker with the `imqbrokerd` command.**

Adding and Removing Brokers

Because HA clusters are self-configuring, the procedures for adding and removing brokers are simpler than for a conventional cluster:

➤ To Add a New Broker to an HA Cluster

1. **Set the new broker's HA-related properties, as described in the preceding section.**

   You can do this either by specifying the individual properties in the broker's instance configuration file (`config.properties`) or, if there is a cluster configuration file, by setting the broker's `imq.cluster.url` property to point to it.

2. **Start the new broker with the `imqbrokerd` command.**

   The broker will automatically register itself into the cluster on startup.

➤ To Remove a Broker from an HA Cluster

1. **Make sure the broker is not running.**

   If necessary, use the command

   ```
   imqcmd shutdown bkr
   ```

   to shud down the broker.

2. **Remove the broker from the cluster with the command**

   ```
   imqdbmgr remove bkr
   ```

Displaying the Cluster Configuration

A new Command utility subcommand

```
imqcmd list bkr
```

lists the current status of all brokers included in the cluster to which a given broker belongs. This subcommand can be used for both conventional and HA clusters; for HA clusters, it displays the information shown in Example 1.

Example 1 Configuration Listing for an HA Cluster

```
Listing all the brokers in the cluster that the following broker is a member of:

-----------------------------------------------
Host         Primary Port    Cluster Broker ID
-----------------------------------------------
localHost    7676            brokerA

Cluster ID                           myClusterID
Cluster Is Highly Available          True

-------------------------------------------------------------------------------------------------------------
                                                                          ID of broker       Time since last
Broker ID       Address         State                 Msgs in store   performing takeover   status timestamp
-------------------------------------------------------------------------------------------------------------
brokerA         localhost:7676  OPERATING             121                                   30 sec
brokerB         greyhound:7676  TAKEOVER_STARTED      52              brokerA               3 hrs
brokerC         jpgserv:7676    SHUTDOWN_STARTED      12346                                 10 sec
brokerD         icdev:7676      TAKEOVER_COMPLETE     0               brokerA               2 min
brokerE         mrperf:7676     *unknown              12                                    0 sec
brokerG         iclab1:7676     QUIESCING             4                                     2 sec
brokerH         iclab2:7676     QUIESCE_COMPLETE      8                                     5 sec
```

Forcing or Preventing Takeover of a Broker

Although the takeover of a failed broker's persistent data by another broker in an HA cluster is automatic, there may be times when you need to force such a takeover to occur manually. (This might be necessary, for instance, if the takeover broker were to fail before completing the takeover process.) In such cases, you can initiate a takeover manually from the command line with the new command

```
imqcmd takeover bkr -n brokerID
```

where *brokerID* is the broker identifier of the broker to be taken over. If the specified broker appears to be running, the Command utility will display a confirmation message:

```
The broker associated with brokerID last accessed the database # seconds ago.
Do you want to take over for this broker?
```

You can suppress this message, and force the takeover to occur unconditionally, by using the `-f` option to the `imqcmd takeover bkr` command:

```
imqcmd takeover bkr -f -n brokerID
```

Conversely, to *prevent* automatic takeover from occurring when shutting down a broker, use the `-nofailover` option:

```
imqcmd shutdown bkr -nofailover
```

You can also *quiesce* a broker with the command

```
imqcmd quiesce bkr
```

This causes the broker to refuse any new client connections while continuing to service old ones, allowing the broker's operation to wind down normally without triggering a takeover by another broker. This is useful, for instance, to prepare the broker to be shut down administratively for upgrade or similar purposes. You can reverse the process and return the broker to normal operation with the command

```
imqcmd unquiesce bkr
```

Managing the HA Data Store

When converting to high-availability operation, you can use the Message Queue Database Manager utility (`imqdbmgr`) to convert an existing standalone JDBC-based persistent data store to a shared HA store:

```
imqdbmgr upgrade hastore
```

NOTE The conversion is supported only for Message Queue versions beginning with 4.1. To convert a standalone data store from an earlier version of Message Queue, the data store must first be upgraded to version 4.1. The broker will perform this upgrade automatically on detecting such an older data store during startup. You can then use the `imqdbmgr` command shown above to convert the upgraded data store for high-availability use.

For durability and reliability, it is a good idea to back up a high-availability cluster's shared persistent data store periodically to backup files. This creates a snapshot of the data store that you can then use to restore the data in case of catastrophic failure. The command for backing up the data store is

```
imqdbmgr backup -dir backupDir
```

where *backupDir* is the path to the directory in which to place the backup files. To restore the data store from these files, use the command

```
imqdbmgr restore -restore backupDir
```

Broker Properties Reference

The tables in this section provide reference information about new broker properties that have been added to Message Queue 4.1 to support high-availability clusters, as well as existing properties that have been modified or enhanced for high-availability operation.

Connection Property

Table 4 shows a new broker connection property that has been added for HA operation.

Table 4 Broker Connection Property for HA

Property	Type	Default Value	Description
`imq.brokerid`	String	None	Broker identifier Must be unique; no two running brokers may have the same broker identifier For brokers using a JDBC-based persistent data store, this string is appended to the names of all database tables to make them unique in the case where more than one broker instance is using the same database. Must be an alphanumeric string of no more than *n* - 13 characters, where *n* is the maximum table name length allowed by the database. This property is usually unnecessary for an embedded database, which stores data for only one broker instance. **Note:** This property replaces the former `imq.persist.jdbc.brokerid` property, which is now deprecated. For high-availability brokers (`imq.cluster.ha` = `true`), database table names use the `imq.cluster.clusterid` property (see Table 6) instead.

JDBC-Based Persistence Properties

Table 5 lists JDBC-based persistence properties that have been added or modified for HA operation.

Table 5 Broker JDBC Persistence Properties for HA

Property	Type	Default Value	Description
imq.persist.jdbc.dbVendor	String	derby	Name of database vendor for persistent data store: hadb HADB (Sun Microsystems, Inc.) derby Derby (Apache Software Foundation) pointbase PointBase (DataMirror Corporation) oracle Oracle Real Application Cluster (Oracle Corporation) mysql MySQL (MySQLAB)
imq.persist.jdbc.*vendorName*.driver	String	None	Java class name of JDBC driver for connecting to database from vendor *vendorName*
imq.persist.jdbc.*vendorName*.opendburl	String	None	URL for connecting to existing database from vendor *vendorName*
imq.persist.jdbc.*vendorName*.createdburl[1]	String	None	URL for creating new database from vendor *vendorName* Needed only if the database will be created using the Message Queue Database Manager utility (imqdbmgr).
imq.persist.jdbc.*vendorName*.closedburl[1]	String	None	URL for closing connection to database from vendor *vendorName*
imq.persist.jdbc.*vendorName*.user[1]	String	None	User name, if required, for connecting to database from vendor *vendorName* For security reasons, the value can instead be specified using command line options imqbrokerd -dbuser and imqdbmgr -u.
imq.persist.jdbc.*vendorName*.needpassword[1]	Boolean	false	Does database from vendor *vendorName* require a password for broker access? If true, the imqbrokerd and imqdbmgr commands will prompt for a password, unless you use the -passfile option to specify a password file containing it.
imq.persist.jdbc.*vendorName*.password[1,2]	String	None	Password, if required, for connecting to database from vendor *vendorName*
imq.persist.jdbc.*vendorName*.property.*propName*[1]	String	None	Vendor-specific property *propName* for database from vendor *vendorName*

1. Optional
2. Should be used only in password files

Cluster Configuration Properties

Table 6 lists cluster configuration properties that have been added or modified for HA operation.

Table 6 Broker Cluster Properties for HA

Property	Type	Default Value	Description
`imq.cluster.ha`	Boolean	`false`	Is broker part of an HA cluster?
`imq.cluster.clusterid`[1,2]	String	None	Cluster identifier Must be unique; no two running clusters may have the same cluster identifier. This identifier is appended to the names of all database tables in the cluster's shared persistent store. Must be an alphanumeric string of no more than *n* - 13 characters, where *n* is the maximum table name length allowed by the database. **Note:** For brokers belonging to an HA cluster, this property is used in database table names in place of `imq.brokerid` (see Table 4).
`imq.cluster.heartbeat.hostname`[1,2]	String	None	Host name or IP address for heartbeat service If specified, overrides `imq.hostname` for the heartbeat service.
`imq.cluster.heartbeat.port`[1,2]	Integer	`7676`	Port number for heartbeat service A value of `0` specifies that the port number should be allocated dynamically by the Port Mapper.
`imq.cluster.heartbeat.interval`[1,2]	Integer	`2`	Interval between heartbeats, in seconds
`imq.cluster.heartbeat.threshold`[1,2]	Integer	`3`	Number of missed heartbeat intervals after which to invoke monitor service
`imq.cluster.monitor.interval`[1,2]	Integer	`30`	Interval, in seconds, at which to update monitor time stamp **Note:** Larger values for this property will reduce the frequency of database access and thus improve overall system performance, but at the cost of slower detection and takeover in the event of broker failure.
`imq.cluster.monitor.threshold`[1,2]	Integer	`2`	Number of missed monitor intervals after which to initiate broker takeover

1. HA clusters only
2. Must have the same value for all brokers in a cluster

Command Line Reference

The tables in this section provide reference information about new utility commands that have been added to Message Queue 4.1 to support high-availability clusters, as well as existing commands that have been modified or enhanced for high-availability operation.

Command Utility

Table 7 lists Command utility (`imqcmd`) subcommands that have been added or modified for HA operation.

Table 7 New and Modified Command Utility Subcommands for HA

Property	Description
`shutdown bkr` [`-b` *hostName*`:`*portNumber*] [`-time` *nSeconds*] [`-nofailover`]	Shut down broker The new `-time` option specifies the interval, in seconds, to wait before shutting down the broker. The new `-nofailover` option indicates that no other broker is to take over the persistent data of the one being shut down.
`quiesce bkr` [`-b` *hostName*`:`*portNumber*]	Quiesce broker The broker will stop accepting new connections; existing connections will continue to operate.
`unquiesce bkr` [`-b` *hostName*`:`*portNumber*]	Unquiesce broker The broker will resume accepting new connections, returning to normal operation.
`takeover bkr -n` *brokerID* [`-f`]	Initiate broker takeover If the specified broker appears to be running, a confirmation message (`Do you want to take over for this broker?`) will be displayed. The `-f` option suppresses this message and initiates the takeover unconditionally.
`query bkr -b` *hostName*`:`*portNumber*	List broker property values For brokers belonging to a cluster, now also lists cluster properties such as broker list, master broker (for conventional clusters), and cluster identifier (for HA clusters).
`list bkr`	List brokers in cluster

Database Manager Utility

Table 8 lists Database Manager utility (`imqdbmgr`) subcommands that have been added or modified for HA operation.

Table 8 New and Modified Database Manager Subcommands for HA

Property	Description
`create tbl`	Create persistent store schema for existing database Used on external database systems. For brokers belonging to a high-availability cluster (`imq.cluster.ha` = `true`), the schema created is for the cluster's shared persistent data store, according to the HA database vendor identified by the broker's `imq.persist.jdbc.dbVendor` property. If `imq.cluster.ha` = `false`, the schema is for the individual broker's standalone data store. Since the two types of store can coexist in the same database, they are distinguished by appending a suffix to all table names: *C**clusterID* Shared store *S**brokerID* Standalone store
`upgrade hastore`	Upgrade standalone store to high-availability (HA) shared store
`backup`	Back up JDBC-based store to backup files
`restore`	Restore JDBC-based store from backup files
`remove bkr`	Remove broker from HA shared store The broker must not be running.

Table 9 lists new command-line options that have been added to the Database Manager utility (`imqdbmgr`) for HA operation.

Table 9 New Database Manager Options for HA

Property	Description
`-dir` *dirPath*	Backup directory for backing up or restoring JDBC-based data store
`-n` *brokerID*	Broker identifier of broker to be removed from HA shared store

JMX Enhancements

The tables in this section provide reference information about new attributes, operations, and notifications that have been added to Message Queue 4.1's Java Management Extensions (JMX) client interface to support high-availability clusters, as well as existing ones that have been modified or enhanced for high-availability operation.

Broker Configuration MBean

Table 10 lists broker configuration attributes that have been added or modified for HA operation. The information in this table supplements Table 3-2 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 10 New and Modified Broker Configuration Operations for HA

Name	Parameters	Result Type	Description
`shutdown`	`nofailover` (`Boolean`) `time` (`Long`)	None	Shut down broker If `nofailover` is `false` or null, another broker will attempt to take over from this broker when it shuts down; this applies only to brokers in a high-availability (HA) cluster. If `nofailover` is `true`, no such takeover attempt will occur. The `time` parameter specifies the interval, in seconds, before the broker actually shuts down; for immediate shutdown, specify `0` or `null`.
`takeover`	`brokerID` (`String`)	None	Take over operation for specified broker The desired broker is designated by its broker identifier (`brokerID`).

Broker Monitor MBean

Table 11 lists broker monitor notifications that have been added for HA operation. The information in this table supplements Table 3-5 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 11 New Broker Monitor Notifications for HA

Name	Utility Constant	Description
mq.broker.takeover.start[1]	BrokerNotification.BROKER_TAKEOVER_START	Broker has begun taking over for another broker
mq.broker.takeover.complete[1]	BrokerNotification.BROKER_TAKEOVER_COMPLETE	Broker has finished taking over for another broker
mq.broker.takeover.fail[1]	BrokerNotification.BROKER_TAKEOVER_FAIL	Attempted broker takeover has failed

1. HA brokers only

Table 12 shows a new data retrieval method that has been added to class BrokerNotification for HA operation. The information in this table supplements Table 3-6 in the *Message Queue 4.0 Developer's Guide for JMX Clients*.

Table 12 New Data Retrieval Method for Broker Monitor Notifications

Method	Result Type	Description
getFailedBrokerID	String	Broker identifier of broker being taken over

Cluster Configuration MBean

Table 13 lists cluster configuration attributes that have been added for HA operation. The information in this table supplements Table 3-74 in the *Message Queue 4.0 Developer's Guide for JMX Clients*.

Table 13 New Cluster Configuration Attributes for HA

Name	Type	Settable?	Description
HighlyAvailable	Boolean	No	High-availability cluster?
ClusterID[1]	String	No	Cluster identifier Used to identify which table in the HA database contains a broker's cluster information.

1. HA clusters only

Table 14 lists cluster configuration operations that have been added or modified for HA operation. The information in this table supplements Table 3-75 in the *Message Queue 4.0 Developer's Guide for JMX Clients*.

Table 14 New and Modified Cluster Configuration Operations for HA

Name	Parameters	Result Type	Description
`getBrokerAddresses`	None	`String[]`	Addresses of brokers in cluster Each address specifies the host name and Port Mapper port number of a broker in the cluster, in the form *hostName*:*portNumber*. **Example:** `host1:3000` For conventional clusters, the list includes all brokers specified by the broker property `imq.cluster.brokerlist`. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.
`getBrokerIDs`[1]	None	`String[]`	Broker identifiers of brokers in cluster The list includes all active and inactive brokers in the cluster table stored in the HA database.
`getBrokerInfoByID`[1]	`brokerID` (`String`)	`CompositeData`	Descriptive information about broker The desired broker is designated by its broker identifier (`brokerID`). The value returned is a JMX `CompositeData` object describing the broker. For conventional clusters, the operation returns `null`.
`getBrokerInfo`	None	`CompositeData[]`	Descriptive information about all brokers in cluster The value returned is an array of JMX `CompositeData` objects describing the brokers. For conventional clusters, the array includes all brokers specified by the broker property `imq.cluster.brokerlist`. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.

1. HA clusters only

Table 15 shows a new lookup key that has been added for use with the `CompositeData` objects returned by cluster configuration attributes and operations. The information in this table supplements Table 3-76 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 15 New Lookup Key for Cluster Configuration Information

Key	Value Type	Description
`ID`[1]	`String`	Broker identifier

1. HA clusters only

Cluster Monitor MBean

Table 16 lists cluster monitor attributes that have been added for HA operation. The information in this table supplements Table 3-78 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 16 New Cluster Monitor Attributes for HA

Name	Type	Settable?	Description
`HighlyAvailable`	`Boolean`	No	High-availability cluster?
`ClusterID`[1]	`String`	No	Cluster identifier Used to identify which table in the HA database contains a broker's cluster information.

1. HA clusters only

Table 17 lists cluster monitor operations that have been added or modified for HA operation. The information in this table supplements Table 3-79 in the *Message Queue 4.0 Developer's Guide for JMX Clients*.

Table 17 New and Modified Cluster Monitor Operations for HA

Name	Parameters	Result Type	Description
`getBrokerAddresses`	None	`String[]`	Addresses of brokers in cluster Each address specifies the host name and Port Mapper port number of a broker in the cluster, in the form *hostName*:*portNumber*. **Example:** `host1:3000` For conventional clusters, the list includes all brokers specified by the broker property `imq.cluster.brokerlist`. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.
`getBrokerIDs`[1]	None	`String[]`	Broker identifiers of brokers in cluster The list includes all active and inactive brokers in the cluster table stored in the HA database.
`getBrokerInfoByID`[1]	`brokerID` (`String`)	`CompositeData`	Descriptive information about broker The desired broker is designated by its broker identifier (`brokerID`). The value returned is a JMX `CompositeData` object describing the broker. For conventional clusters, the operation returns `null`.
`getBrokerInfo`	None	`CompositeData[]`	Descriptive information about all brokers in cluster The value returned is an array of JMX `CompositeData` objects describing the brokers. For conventional clusters, the array includes all brokers specified by the broker property `imq.cluster.brokerlist`. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.

1. HA clusters only

Table 18 lists new lookup keys that have been added for use with the `CompositeData` objects returned by cluster monitor attributes and operations. The information in this table supplements Table 3-80 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 18 New Lookup Keys for Cluster Monitor Information

Key	Value Type	Description
`ID`[1]	`String`	Broker identifier
`TakeoverBrokerID`[1]	`String`	Broker identifier of broker that has taken over for this broker
`NumMsgs`[1]	`Long`	Current number of messages stored in memory and persistent store
`StatusTimestamp`	`Long`	Time of last status update, in standard Java format (milliseconds since January 1, 1970, 00:00:00 UTC) Used to determine whether a broker is running. The interval at which a broker updates its status can be configured with the broker property `imq.cluster.monitor.interval` (see Table 6 on page 13).

1. HA clusters only

Table 19 lists new broker state values for the lookup keys `State` and `StateLabel` in the `CompositeData` objects returned by cluster monitor attributes and operations. The information in this table supplements Table 3-81 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 19 New Broker State Values for HA

Value	Utility Constant	String Representation	Meaning
1	`BrokerState.TAKEOVER_STARTED`	`TAKEOVER_STARTED`	Broker has begun taking over for another broker
2	`BrokerState.TAKEOVER_COMPLETE`	`TAKEOVER_COMPLETE`	Broker has finished taking over for another broker
3	`BrokerState.TAKEOVER_FAILED`	`TAKEOVER_FAILED`	Attempted broker takeover has failed

Table 20 lists cluster monitor notifications that have been added for HA operation. The information in this table supplements Table 3-82 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 20 New Cluster Monitor Notifications for HA

Name	Utility Constant	Description
`mq.broker.takeover.start`[1]	`BrokerNotification.BROKER_TAKEOVER_START`	Broker has begun taking over for another broker
`mq.broker.takeover.complete`[1]	`BrokerNotification.BROKER_TAKEOVER_COMPLETE`	Broker has finished taking over for another broker
`mq.broker.takeover.fail`[1]	`BrokerNotification.BROKER_TAKEOVER_FAIL`	Attempted broker takeover has failed

1. HA brokers only

Table 21 shows a new data retrieval method that has been added to class `ClusterNotification` for HA operation. The information in this table supplements Table 3-83 in the *Message Queue 4.0 Developer's Guide for JMX Clients.*

Table 21 New Data Retrieval Method for Cluster Monitor Notifications

Method	Result Type	Description
`isHighlyAvailable`	`Boolean`	High-availability cluster?