MQSeries Clustering on Windows NT

Prepared by:
Scott Meridew – Senior Technical Consultant,
CGI Group Inc.

CGI Proprietary
The information contained herein is proprietary to one or more CGI Group of companies or other parties and shall not be used, reproduced or disclosed to others except as specifically permitted in writing by the proprietor. The recipient of this information, by its retention and use, agrees to protect the same from loss, theft or unauthorized use.

TABLE OF CONTENTS

1 ABOUT THESE ARTICLES 3

1.1 CONVENTIONS USED IN THE ARTICLES 3
1.2 AUDIENCE 3

2 INTRODUCTION TO MQSERIES CLUSTERING 4

2.1 WHAT IS A CLUSTER? 4
2.2 CLUSTERING VERSUS DISTRIBUTED QUEUING 5
2.2.1 Load Balancing and Performance 6
2.2.2 High Availability 9
2.2.3 Simplified Management 10
2.2.4 Dynamic Object Creation 11
2.3 MQSERIES CLUSTERING COMPONENTS 13
2.3.1 Repository Queue Managers 13
2.3.2 Cluster Channels (Sender-Receiver) 14
2.3.3 Cluster Queues (Local and Transmission) 15
2.3.4 NameList 16
2.3.5 The default cluster objects summary 16

1 About These Articles

With the introduction of MQSeries V5.1, (and V2.1 on OS/390) IBM has included the ability to logically group or "cluster" queue managers together. This series of technical articles is broken down as follows:

- An introduction to MQSeries Clustering. *This will define clustering, and how it applies to MQSeries. Clustering versus distributed queuing, and the components of an MQSeries cluster are discussed.*
- Defining and managing an MQSeries Cluster. *A step-by-step approach to building an MQSeries cluster on Windows NT. Screen shots and instructions on how-to add/remove MQSeries objects within the cluster. Trouble-shooting clustering problems.*
- Advanced features in MQSeries Clustering. *Customizing workload exits. Overlapping and cluster to non-cluster communications. Securing your MQSeries cluster.*
- Practical business applications for MQSeries Clustering. *The provisioning model. The Web storefront model.*

1.1 Conventions used in the articles

Throughout this document formatting has been used to indicate the following to the reader:

- Important notes are in ***bold italics***
- Definitions and specific new terms are in *italics*
- MQSeries MQSC commands, objects, and API calls are in **BOLD UPPERCASE**.
- URLs, FTP and email addresses are in blue when displayed from a monitor. For print, http:// always proceeds a URL, ftp:// always proceeds an FTP address, and mailto: always proceeds an email address.

1.2 Audience

This document will be useful for both MQSeries developers who are implementing an MQSeries cluster, as well as system architects and designers.

2 Introduction to MQSeries Clustering

2.1 What Is a Cluster?

Clustering is not a new concept. Clustering in its simplest form means logically grouping two or more components and making them appear as one to a consumer of the components. Two forms of clustering exist: shared and shared-nothing.

A shared cluster generally involves creating redundant, shareable components for the purpose of high availability, fault tolerance and load balancing. Typically, system resources that can be shared include disk storage and CPU/applications. When disk storage is clustered, the operating system must manage locking conditions across the CPUs to maintain the integrity of the data being accessed. This is no simple task, and is the reason many vendors provide only shared-nothing clustering solutions.

A shared-nothing cluster (such as Microsoft's Cluster Server) does not share any components, but usually involves replication of data (in a timely manner) that can be used by a hot standby system/application. In this scenario, clustering provides fail-over capabilities, but not load-balancing.

Digital Equipment Corp. (DEC), now a division within Compaq, has produced shared cluster systems since the mid-1980s. These systems typically provide redundant CPU, disk subsystem, operating system and applications. The idea is that if any one of these components fail, an alternate path survives to provide a relatively transparent fail over to the users. The two key components to this architecture are a high-speed computer-interconnect bus that allows the processors to communicate very quickly with one another, and hierarchical storage controllers with dual-ported disks to provide the concurrent access to shared storage.

Clustered applications in a shared-nothing environment are a little different than their shared cluster counterparts. Since no common data is shared among the applications, each instance of the application has access to a replicated copy of configuration and/or production data. MQSeries clustering works on this concept of replicated data. MQSeries object definitions are stored in a repository queue and replicated, either in full or partially, to replication partners known as repository queue managers. We'll discuss the components later.

2.2 Clustering Versus Distributed Queuing

Clustering offers several benefits over distributed queuing. By clustering and sharing data, the MQSeries network becomes more dynamic and more manageable. As your MQSeries network grows, the risk of errors increases, and the flexibility decreases. With clustering however, the risk of errors can be greatly reduced, while maintaining the flexibility to modify your configuration on the fly.

Clustering also provides a simple way to provide fault-tolerance and high availability. The product provides built-in, customizable logic for routing messages to destination queues. You can use the default method for message distribution, or create customized '*workload exits*' for advanced routing algorithms.

In a distributed queuing architecture, this logic has to be coded into the applications that are sending the messages. In addition, error-handling routines need to be developed to manage the condition where one or more queue managers become unavailable. Furthermore, additional code is needed to recognize when these unavailable queue managers become available once again.

MQSeries clustering will dynamically create and destroy certain objects as needed, further increasing the ease of management. Specifically, the channel definitions to/from cluster queue managers are created automatically--however, the first channel pair must be defined manually. These are known as Cluster Sender (**CLUSSDR**) and Cluster Receiver (**CLUSRCVR**) channels. As with all MQSeries implementations, all channels are uni-directional and therefore require this pairing.

Perhaps the best feature of MQSeries clustering is that existing applications do not need to be modified in order to benefit from the features of clustering. The entire cluster configuration is handled at the administration layer – entirely transparent from users and applications.

2.2.1 Load Balancing and Performance

MQSeries clustering provides essential load-balancing features by providing a default method for identifying target queues. Consider the following diagram:

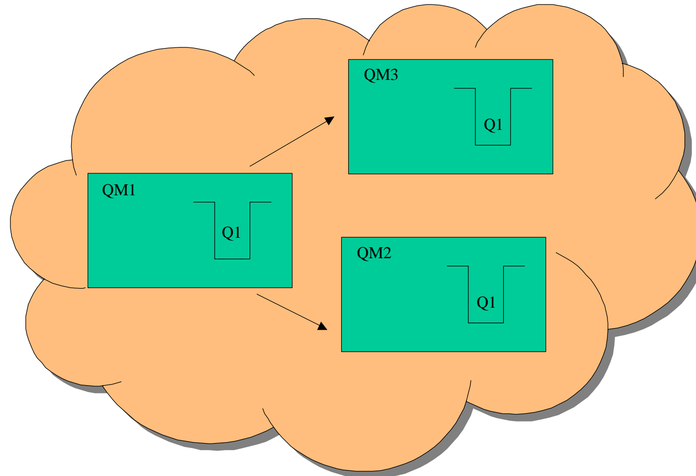

Diagram 1

Cluster Workload Management Algorithm

An application running on QM1 needs to put a message to Q1. But which queue will be the target? In this case, Q1 on QM1 will <u>always</u> be used.

MQSeries clustering uses a *workload management algorithm*, which operates according to the following rules:

- If a local queue exists, the message will always be sent to the local queue. That is, a queue managed by the same queue manager that is putting the message
- If multiple instances of the queue are available, but only on non-local queue managers, then a round robin approach is used. The state of each sender channel, as well as the priority associated with the channel (**NETPRTY** channel attribute) is used in determining which queue is selected first
- If an attempt fails to put the message to the selected target queue, each alternate target is attempted in succession
- If all target queues fail, then the message is sent to the deal letter queue

Cluster Workload Exits

In addition to the *workload management algorithm,* you can customize how workload is distributed using *cluster workload exits.* There are many reasons you may need to customize. For example:

- If you need to determine which channels use a high-speed connection in order to favor that channel
- Perhaps application performance is your primary concern and you want to send messages to the least busy processor
- What if your data is partitioned? You may have the same application running on QM1 and QM2, but each instance is responsible for a certain range of data, like customer identifier. Your workload exit could examine the message data first, and then route to the application that handles that specific customer
- Perhaps the previous applications are designated by a quality of service (QOS) and high-value customer requests need to be routed to the system running the more reliable or faster instance of the application

Cluster workload exits are called at open or put time. That is, when the sending application calls **MQPUT**, **MQPUT1** or **MQOPEN**. ***Note: Only one cluster workload exit can be defined for a given queue manager.***

To define a workload exit, go to the properties sheet of the queue manager from MQSeries Explorer as shown in Figure 1. Enter the *cluster workload exit* name, the *cluster workload data* (user data contained in a maximum 32-character field) and the *cluster workload length* (this is the maximum length of the message data passed to the cluster workload exit. The actual length of data passed to the exit is the minimum of: the length of the message, the queue-manager's *MaxMsgLength* attribute, and the *cluster workload length*).

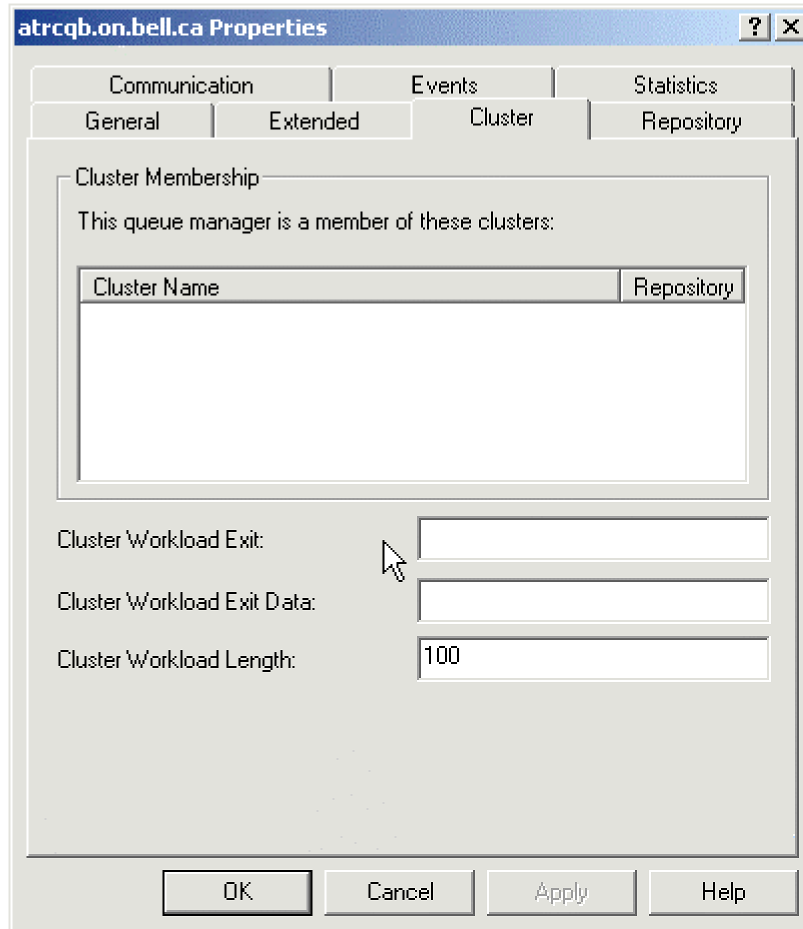

Figure 1

2.2.2 High Availability

By implementing an architecture with multiple instances of a cluster queue, higher availability can be realized. As previously mentioned, the *cluster workload algorithm* contains built-in recovery mechanisms to accommodate situations where messages cannot be routed to a specific target cluster queue. The default mechanism will successively attempt to route the message to any available target queue based on channel priority (NETPRTY channel attribute) and status. Of course, you can write your own cluster workload exits to handle this scenario as well.

In a distributed queuing environment, these types of failures would result in the message being routed to the dead letter queue, where some cleanup application would need to take some form of corrective action. The cluster workload algorithm thus provides high availability by having messages processed by alternate means. It is recommended that the redundant target cluster queues be placed on separate nodes, in order to increase the availability. This will ensure that messages will continue to be processed, even in the event that a destination queue manager fails. Placing all of your target queues on a single node would not provide high availability in the event of a queue manager or system failure, but only that of a queue failure.

Duplication of the cluster repository also provides for high availability. IBM recommends that at least two queue managers in the cluster are designated as full cluster repository queue managers. As we'll see in later articles, the management of the cluster repositories is critical in ensuring high availability of your cluster.

2.2.3 Simplified Management

Any MQSeries administrator will agree that managing MQ objects in a complex network can be very cumbersome, especially without a third-party management tool such as Candle Command Center or Tivoli's MQ. The fewer the objects, the easier it is to manage both a static MQ network and a growing one.

Including remote queue definitions, a simple MQSeries network with four Queue Managers with two local queues would require a minimum of 68 objects to be defined, broken down as:

- 3 sender channels per queue manager (12 total)
- 3 receiver channels per queue manager (12 total)
- 3 transmission queues per queue manager (12 total)
- 2 local queue definitions per queue manager (8 total)
- 6 remote queue definitions per queue manager (24 total)

In a clustered environment, the number of objects to be defined is reduced to 16. This is because only one cluster sender and cluster receiver channel is needed, and no transmission queues or remote queue definitions are necessary. Therefore the only objects required are:

- 1 cluster sender channel per queue manager (4 total)
- 1 cluster receiver channel per queue manager (4 total)
- 2 local queue definitions per queue manager (8 total)

With this type of configuration, the risk of making errors in defining transmission queues and remote queue definitions is eliminated.

2.2.4 Dynamic Object Creation

MQSeries clustering will automatically create and destroy certain objects, as they are needed for clustering. The two types of objects created are cluster channels and cluster queues.

Channels

MQSeries clustering relieves some of the burden of managing channels. In studies performed by IBM, channels were the number-one cause of administrative frustration. To reduce the level of aggravation and the risk of errors through incorrect channel declaration, clustering requires only two channels to be defined on each queue manager:

- 1 cluster receiver channel (CLUSRCVR);
- 1 cluster sender channel (CLUSSDR)

Once these channels are defined, MQSeries will automatically create any subsequent channels to other queue managers in the cluster. For instance, in our diagram, if QM1 needed to send a message to QM3/Q1, but had only been defined with sender/receiver channels to QM2, the sender/receiver channels to QM3 would automatically be created. The attributes of these channels are taken from the cluster sender/receiver channels on the destination queue manager (QM3 in this case).

Remote Queues

Queues defined as cluster queues are automatically made available to all queue managers in the cluster. What happens locally is that MQSeries dynamically creates a remote queue definition on the local queue manager. These queues can be displayed using MQSeries Explorer and will appear as "local cluster queues." The queue clq_test in Figure 2 is a local cluster queue on QM_cgilab72.on.bell.ca that is hosted by the QM_cgilab71.on.bell.ca queue manager.

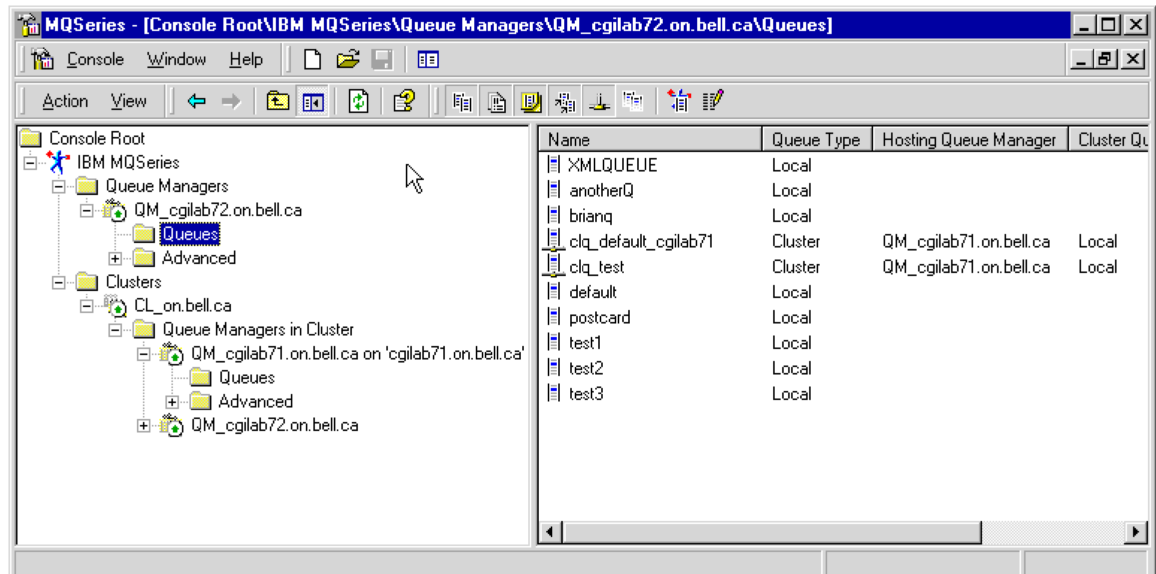

Figure 2

Removal of Dynamic Objects

These objects will automatically be destroyed after a period of inactivity. However, you can refresh the local queue manager repository information using MQSeries Explorer as shown in Figure 3. This will destroy and recreate the automatically defined cluster channels and cluster queue definitions on the local machine. You can also see the local shared cluster queues in Figure 2 (clq_test and clq_default_cgilab71) that have been manually created on QM_cgilab71.on.bell.ca.

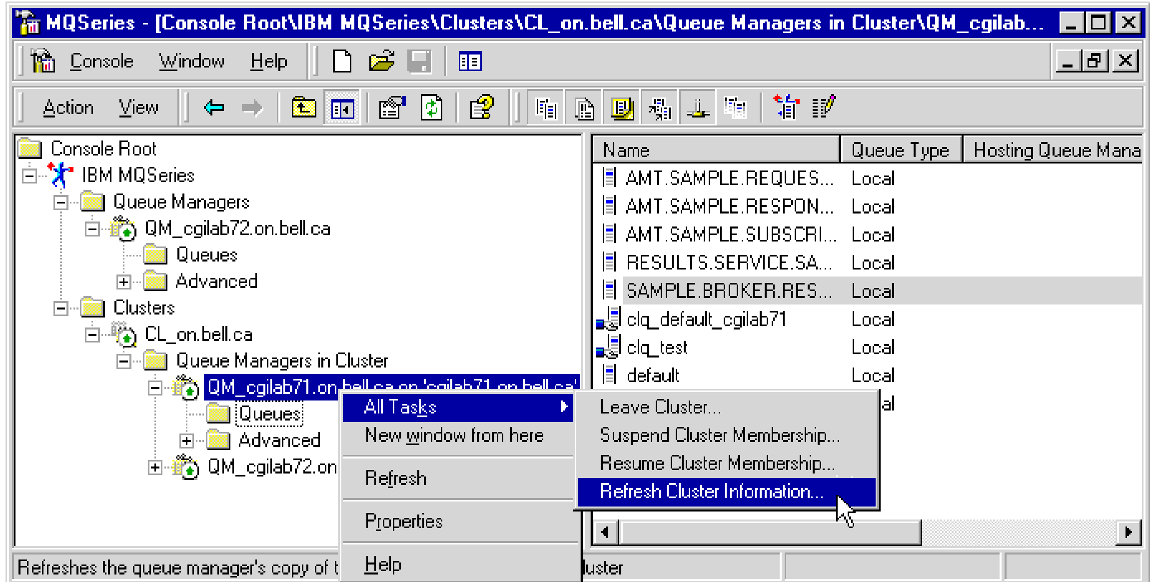

Figure 3

2.3 MQSeries Clustering Components

2.3.1 Repository Queue Managers

Repository queue managers are the keepers of cluster information. At least two repository queue managers should be defined in a cluster for availability purposes. A cluster queue manager can be designated as either a FULL or PARTIAL repository. A FULL repository contains information about all the queue managers in a cluster, including the queue manager names, locations, their channels, what queues they host, etc. A PARTIAL repository is held by all non-FULL repository queue managers in a cluster, and contains only information about queue managers that the local system needs to exchange messages with.

Cluster repository information is retained and managed within a local system queue known as the SYSTEM.CLUSTER.REPOSITORY.QUEUE. The cluster repository information is replicated as it changes. The SYSTEM.CLUSTER.COMMAND.QUEUE is used to send/receive repository updates from repository queue managers. This information can be refreshed, as in Figure 3.

To define a queue manager as a repository queue manager, open MQSeries Explorer and right click on the Queue Manager. Then select Properties, and choose the Repository tab. Select the radio button "Repository for a cluster" and enter the name of the cluster, as shown in Figure 4.

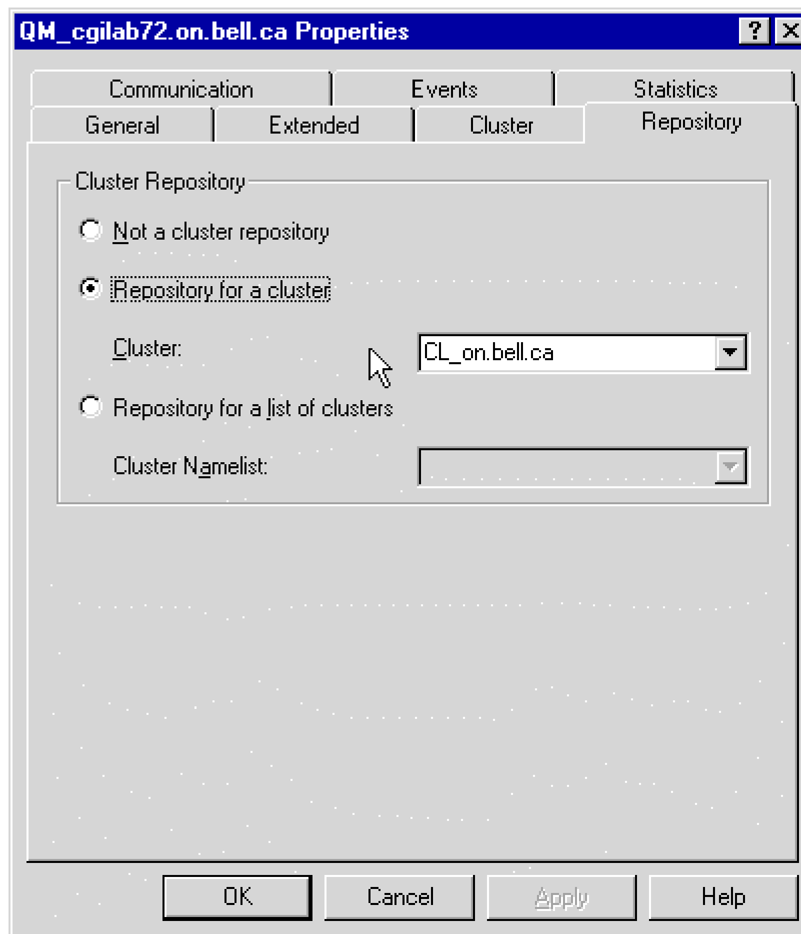

Figure 4

2.3.2 Cluster Channels (Sender-Receiver)

Cluster channels are used by cluster queue managers to exchange messages between one another. A minimum of two channels are needed, a cluster sender (CLUSSDR) and a cluster receiver (CLUSRCVR). It is important to note that the channel pair names must match. That is, the CLUSSDR channel name on QM1 must match the corresponding CLUSRCVR channel name on QM2.

To define a cluster channel, use the MQSeries Explorer and right click on Channels/New as shown in Figure 5.

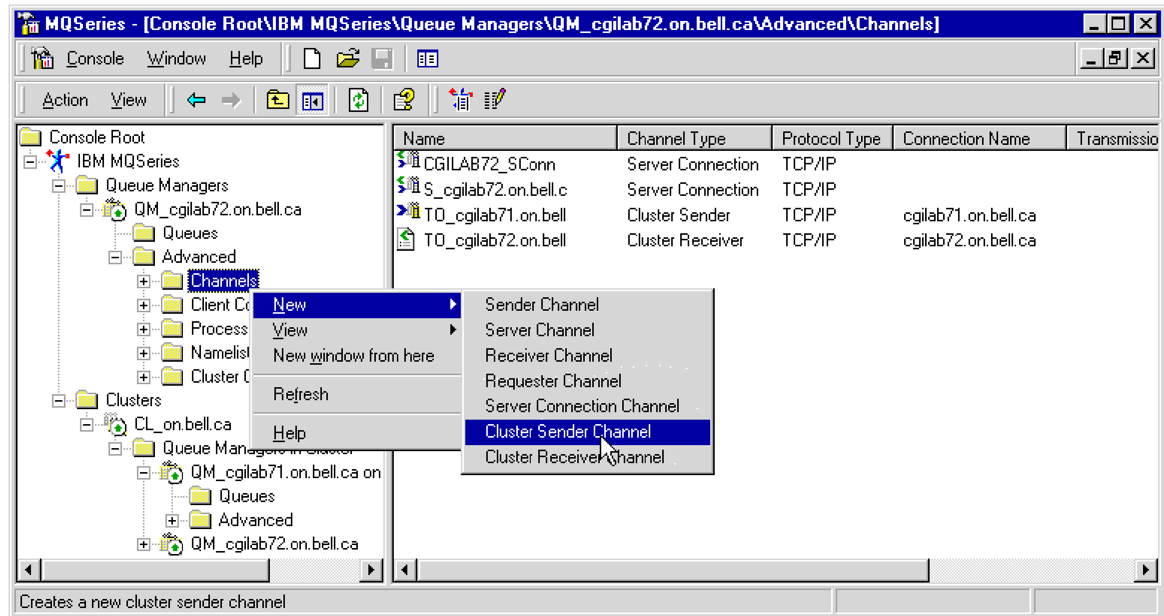

Figure 5

At a minimum, you must supply the channel name, the connection name and the transport type. Valid transport types for Windows NT are TCP/IP, NetBIOS, IPX or LU6.2. On the “New cluster sender channel” Properties sheet, select the “Cluster” tab to share the channel in the cluster, and to set the priority (NETPRTY).

MQSeries will automatically define any additional cluster channels needed. This is the default, unlike other channels that MQSeries can create automatically (i.e. Server Connection channels).

2.3.3 Cluster Queues (Local and Transmission)

Cluster queues are either defined by users, as in the case of local cluster queues, or automatically in the case of cluster transmission queues.

To create a cluster queue, you must first create a local queue on the queue manager that will host the queue. By selecting "Share in Cluster" or "Share in a list of clusters" (and selecting the cluster name or namelist) as shown in Figure 6, you can share this queue in the cluster(s).

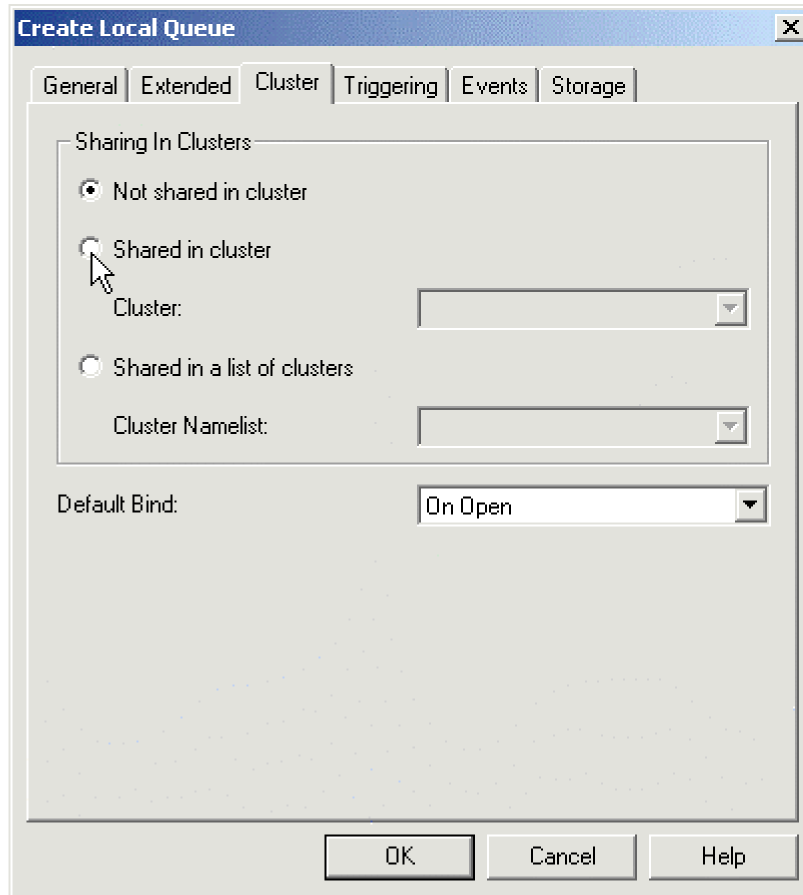

Figure 6

To view cluster queues within MQSeries Explorer, you must first click the Show/Hide Cluster Queues toggle button on the toolbar, shown below as the fifth icon from the left.

Figure 6a

In addition, a system default "cluster transmission queue" called **SYSTEM.CLUSTER.TRANSMIT.QUEUE** is automatically created when you create a new queue

manager. This queue is similar to the user-defined default transmission queue, but takes precedence over it in cases where the target queue is resolved via the repository.

2.3.4 NameList

A namelist can be used to define a group of clusters. This would typically be used when defining a queue manager that will host repositories for multiple clusters. In order to specify that the queue manager will be a repository queue manager for a list of clusters, use the properties sheet for the queue manager and select the "Repository for a list of clusters" radio button as shown in Figure 4. Additionally, queues can be shared in a list of clusters also, as shown in Figure 6.

To create a namelist, right click the Namelist folder in MQSeries Explorer as shown in Figure 7.

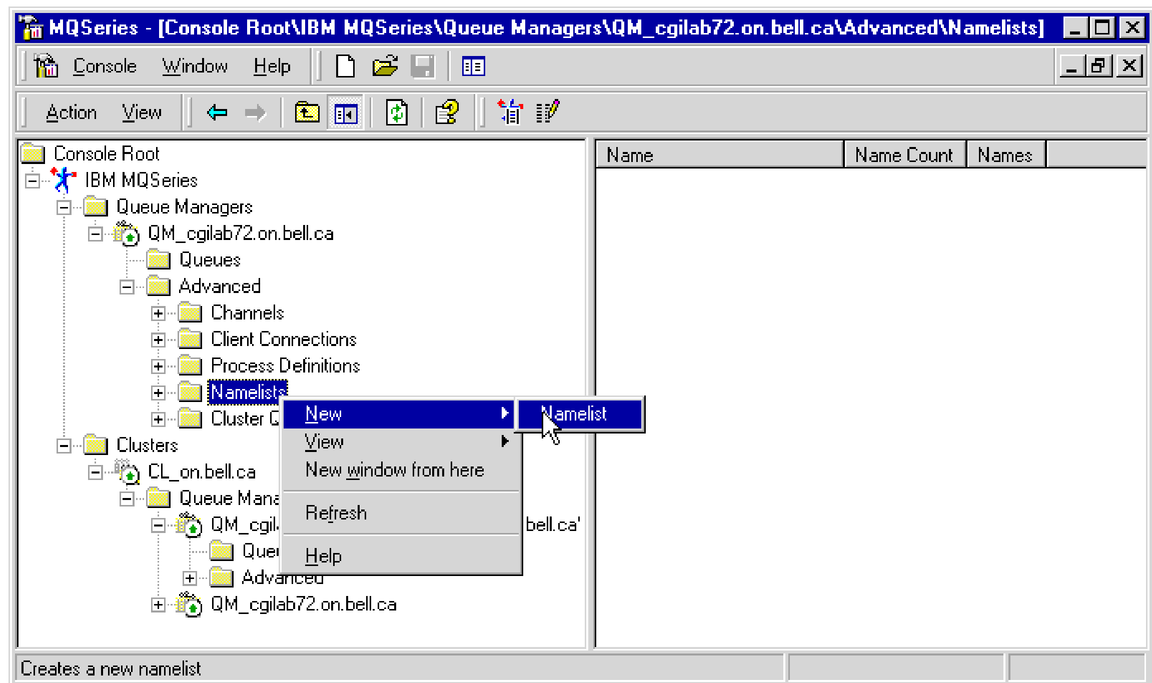

Figure 7

2.3.5 The default cluster objects summary

As with any MQSeries installation, several default objects are created when you create the queue manager. Most of these have previously been discussed. Here is a synopsis of the system cluster objects:

SYSTEM.CLUSTER.REPOSITORY.QUEUE – the queue that holds repository information;
SYSTEM.CLUSTER.COMMAND.QUEUE – the queue used to send repository update messages;
SYSTEM.CLUSTER.TRANSMIT.QUEUE – the default cluster transmission queue;
SYSTEM.DEF.CLUSSDR – the default definition for a cluster sender channel;
SYSTEM.DEF.CLUSRCVR – the default definition for a cluster receiver channel

In order to display the system objects within MQSeries Explorer, you must click on the Display System Objects Icon on the toolbar, shown below as the second icon from the right.

Figure 7a