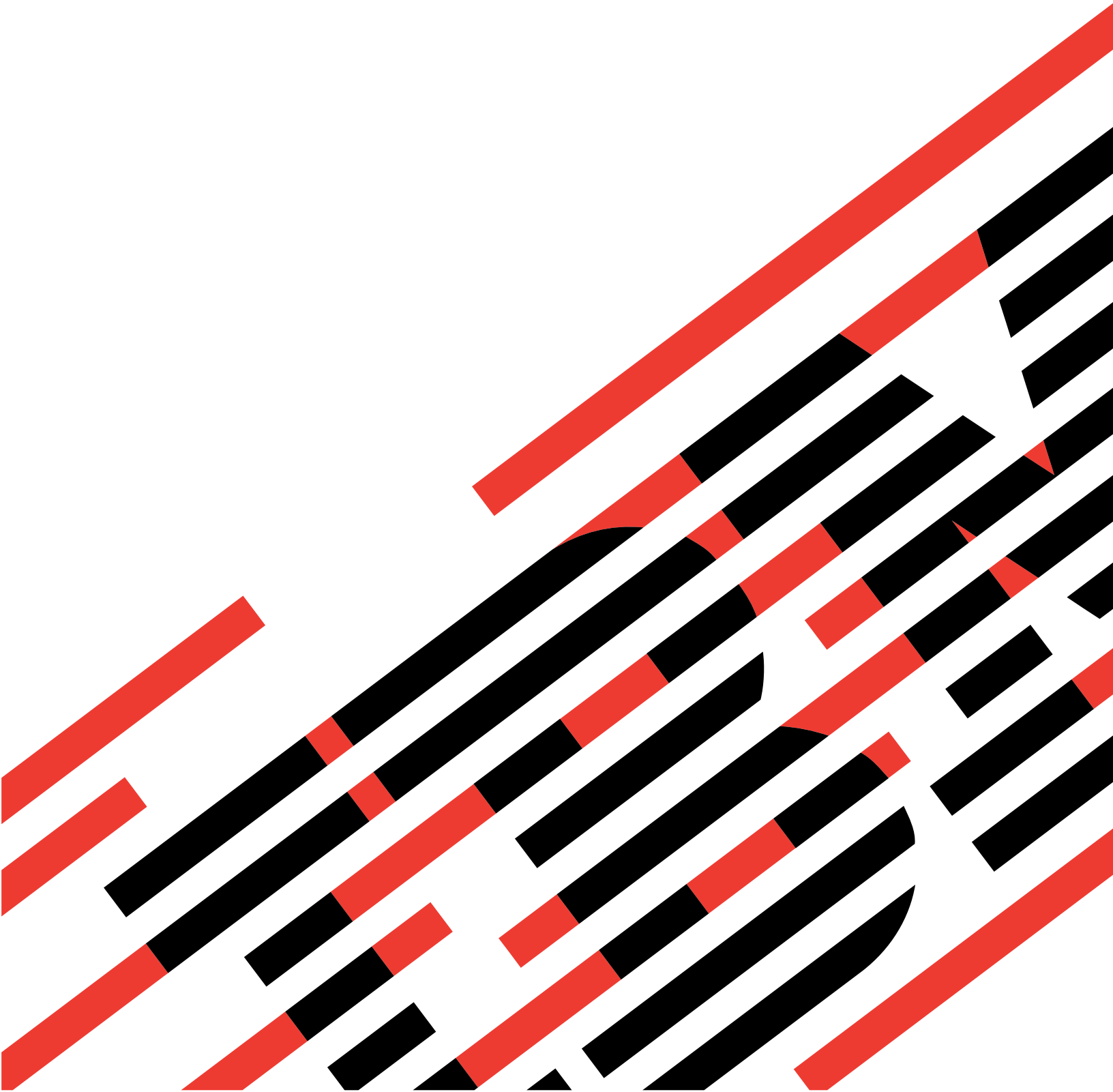
IBM

eserver

iSeries

Clusters

Version 5 Release 3



IBM

@server

iSeries

Clusters

Version 5 Release 3

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 107.

Sixth edition (August 2005)

This edition applies to Version 5, Release 3, Modification 0 of IBM Operating System/400 (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Clusters. 1
What's new for V5R3 1
Print this topic. 2
Cluster concepts 3
Benefits of clusters 3
Cluster basics 3
The elements of a cluster 6
Cluster node 7
Cluster resource group 7
Cluster resource group exit programs 8
Recovery domain 8
Cluster version 10
Device domains 12
Resilient resources 13
Resilient applications 14
Resilient data 14
Resilient devices 14
Failover 15
Example: Failure. 15
Switchover 17
Rejoin 17
Example: Rejoin 18
Merge 19
Replication 19
Heartbeat monitoring 19
Reliable message function 21
Change cluster resource services settings . . 22
Cluster partition 22
How a cluster works 22
Plan for clusters 23
Solutions for configuring and managing clusters 23
iSeries Navigator cluster management . . . 24
Cluster commands and APIs. 25
Cluster middleware business partners and available clustering products 25
Cluster requirements 26
Hardware requirements for clusters 26
Software and licensing requirements for clusters 26
Communications requirements for clusters . . 27
Design your cluster. 28
Design your network for clusters 28
Set up IP addresses. 28
Set TCP/IP configuration attributes . . . 28
Tips: Cluster communications 29
Avoid a cluster partition 29
Dedicate a network for clusters. 29
Multiple-release clusters 30
Identify servers to include in a cluster . . . 30
Identify applications to include in a cluster . . 30
Plan for data resilience 30
Determine which data should be made resilient. 31
Comparison of replication, switched disks, and cross-site mirroring 31
Plan for replication 33
Plan for switchable independent disk pools and geographic mirroring 33
Cluster security 34
Enable a node to be added to a cluster . . . 34
Distribute cluster-wide information 34
Maintain user profiles on all nodes 35
Cluster configuration checklist 35
Cluster applications 37
OS/400 architecture for cluster-enabled applications 38
Writing a highly available cluster application . . 38
Make application programs resilient 39
Restart highly available cluster applications . 40
Calling a cluster resource group exit program 40
Application CRG considerations 41
Manage application CRG IP addresses . . . 41
Example: Application cluster resource group failover actions 42
Example: Application exit program 42
Configure clusters 78
Create a cluster 78
Manage clusters 79
Add a node to a cluster 80
Start a cluster node 80
Delete a cluster 81
Remove a cluster node 82
Adjust the cluster version of a cluster 82
Change the recovery domain for a cluster resource group 83
Perform a switchover 83
Add a node to a device domain 84
Remove a node from a device domain 85
Monitor cluster status 85
Cluster performance 86
Balance network load for clusters 86
Tune cluster performance. 86
End cluster jobs 87
Job structure and user queues 87
Backup and recovery of clusters 88
Save cluster configurations 89
Examples: Cluster configurations 89
Troubleshoot clusters 89
Determine if a cluster problem exists 90
Common cluster problems 91
Partition errors 93
Determine primary and secondary cluster partitions 93
Change partitioned nodes to failed 94
Tips: Cluster partitions 95
Cluster recovery 96
Recover from cluster job failures 96
Recover a damaged cluster object 96
Recover a cluster after a complete system loss 97
Recover a cluster after a disaster 98
Restore a cluster from backup tapes 98

Frequently asked questions about iSeries Navigator cluster management 98
General 99
iSeries Navigator cluster management . . 99
Communications 101
Security 102
Troubleshooting 103
Who to call for cluster suppport 104
Related information 105

Appendix. Notices 107
Trademarks 109
Terms and conditions for downloading and printing information 109
Code disclaimer information 110

Clusters

Clusters let you efficiently group your iSeries(TM) servers together to set up an environment that provides availability that approaches 100 percent for your critical applications, devices, and data. Clusters also provide simplified systems management and increased scalability to seamlessly add new components as your business grows.

"What's new for V5R3"
Take a look at what is new for this release."Cluster concepts" on page 3

"Print this topic" on page 2
View or download a PDF version of this Clusters topic for viewing or printing.

"Cluster concepts" on page 3
Get a complete understanding of how clusters work. Read about the benefits of clusters and how they can be important to you, as well as information on important clustering concepts and how they fit together.

"Plan for clusters" on page 23
Find what you need to do before you can set up clusters on your iSeries servers. Find out the prerequisites for clusters as well as hints on designing your cluster. Finally, read tips for setting up your network and some performance hints for clusters.

"Cluster applications" on page 37
Read about considerations for writing and implementing highly available applications within your cluster.

"Configure clusters" on page 78
Understand how to go about creating a cluster.

"Manage clusters" on page 79
Read the cluster management procedures to help you maintain your cluster.

"Examples: Cluster configurations" on page 89
Use these examples of typical cluster implementations to understand when, why, and how implementing clusters can be beneficial.

"Troubleshoot clusters" on page 89
Find error recovery solutions for problems that are specific to clusters.

"Related information" on page 105
IBM(R) related information contains technical, know-how, and "how to" information.

Note: Read the "Code disclaimer information" on page 110 for important legal information.

What's new for V5R3

Clustering has been enhanced in V5R3 to provide support for:

Geographic mirroring

Geographic mirroring is a subfunction of cross-site mirroring (XSM), which is part of OS/400(R) Option 41, High Availability Switchable Resources. Geographic mirroring allows you to maintain a replica of an independent disk pool at a remote physical location.

- Geographic mirroring

What's new as of 27 February 2006

Additional information was added to help users perform upgrades in a clustered environment. See "Cluster version" on page 10 for details.

How to see what's new or changed

To help you see where technical changes have been made, this information uses:

- The

  image to mark where new or changed information begins.
- The

  image to mark where new or changed information ends.

To find other information about what's new or changed this release, see the Memo to Users.

Print this topic

To view or download the PDF version of this topic, select Clusters (about 938 KB).

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the link above).
2. 

   Click **Save Target As...** if you are using Internet Explorer. Click **Save Link As...** if you are using Netscape Communicator.

   
3. Navigate to the directory in which you would like to save the PDF.
4. Click **Save**.

Downloading Adobe Acrobat Reader

You need Adobe Acrobat Reader to view or print these PDFs. You can download a copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html)

.

Cluster concepts

An iSeries(TM) cluster is a collection or group of one or more servers or logical partitions that work together as a single server. Servers in a cluster, called "Cluster node" on page 7, work cooperatively to provide a single computing solution. iSeries clustering supports up to 128 nodes in a cluster. This allows you to efficiently group your iSeries servers together to set up an environment that provides availability that approaches 100 percent for your critical applications and your critical data. This helps ensure that your critical servers and applications are available 24 hours a day, seven days a week. Clusters also provide simplified systems management and increased scalability to seamlessly add new components as your business grows.

For more cluster concepts, see the following:

- "Benefits of clusters"
- "Cluster basics"
- "The elements of a cluster" on page 6
- "How a cluster works" on page 22

Benefits of clusters

Clustering offers a continuous availability solution if your business demands operational systems 24 hours a day, seven days a week. By implementing clusters, you can greatly reduce the number and duration of unplanned outages, ensuring that your servers, data, and applications are continuously available.

The major benefits that clusters can offer your business are:

Continuous availability
: The "Switchover" on page 17 and "Failover" on page 15 mechanisms provided by clusters ensure that your servers, data and applications remain continuously available.

Simplified administration
: You can manage a group of systems as a single server or single database, without having to sign on to individual servers.

Increased scalability
: Seamlessly add new components as your business growth requires.

Cluster basics

Before you begin to design and customize a cluster that will satisfy your needs, you need to understand the basic clustering concepts. The example below illustrates the basic constructs of a cluster: its **"Cluster node" on page 7** and **"Cluster resource group" on page 7**.

In this cluster, there are five cluster nodes. Nodes are the iSeries(TM) servers or logical partitions that are members of the cluster. When you create a cluster, you specify the servers that you want to include in the cluster as nodes.

There are three **cluster resource groups** (CRGs) present in this example. A cluster resource group serves as the control object for a collection of resilient resources. The CRG defines actions to be taken during a switchover or failover. Each CRG accomplishes this by defining the following:

- "Recovery domain" on page 8 - specifies the role of each node in the CRG:
  - The **primary** node is the cluster node that is the primary point of access for the resilient cluster resource.
  - A **backup** node is a cluster node that will take over the role of primary access if the present primary node fails or a manual switchover is initiated.
  - A **replicate** node is a cluster node that has copies of cluster resources, but is unable to assume the role of primary or backup.
- "Cluster resource group exit programs" on page 8 - manages cluster-related events for that group; one such event would be moving an access point from one node to another node

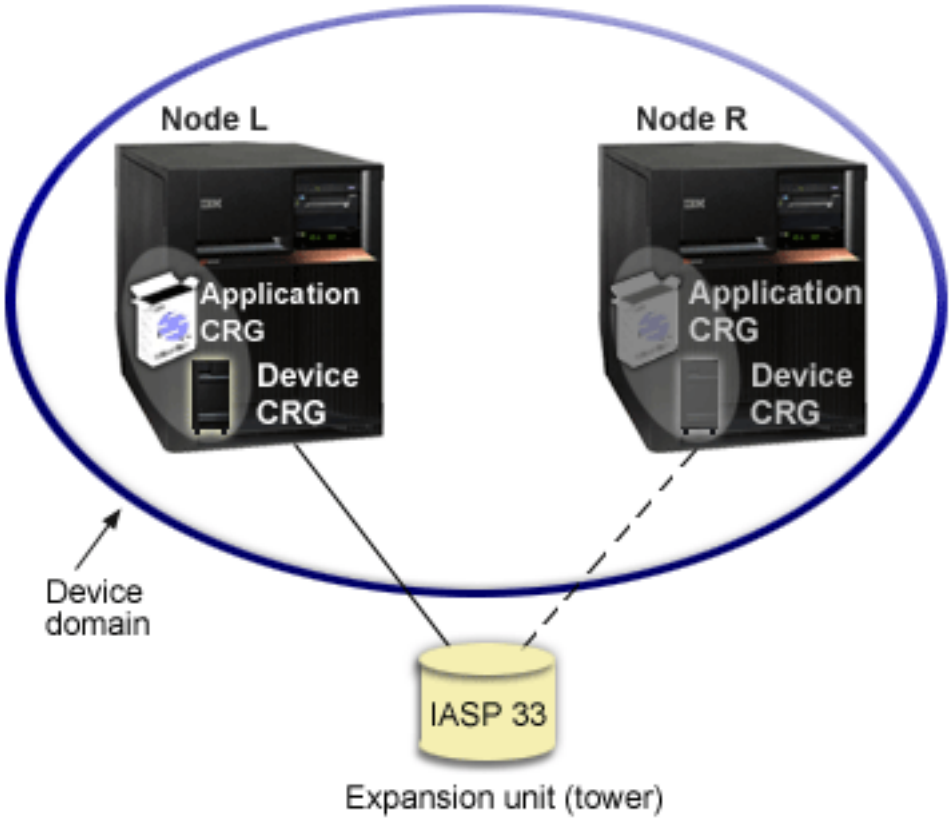
When you create a CRG in a cluster, the CRG object is created on all nodes specified to be included in the recovery domain. However, a single system image of the CRG object, which you can access from any active node in the CRG's recovery domain, is provided. That is, any changes made to the CRG will be made on all nodes in the recovery domain.

An iSeries cluster supports three types of CRGs: application, data and device. In the example above, one CRG of each type is present:

- **Data CRG**: The data CRG is present on Node 1, Node 2 and Node 3. This means that the recovery domain for the data CRG has specified a role for Node1 (primary), Node 2 (first backup) and Node 3 (second backup). In the example, Node 1 is currently serving as the primary point of access. Node 2 is defined as the first backup in the recovery domain. This means that Node 2 contains a copy of the resource which is kept current through replication. Should a failover or switchover occur, Node 2 would become the primary point of access.
- **Application CRG**: The application CRG is present on Node 4 and Node 5. This means that the recovery domain for the application CRG has specified Node 4 and Node 5. In the example, Node 4 is currently serving as the primary point of access. Should a failover or switchover occur, Node 5 would become the primary point of access for the application. Requires a takeover IP address.
- **Device CRG**: The device CRG is present on Node 2 and Node 3. This means that the recovery domain for the device CRG has specified Node 2 and Node 3. In the example, Node 2 is currently serving as the primary point of access. This means that the resilient device owned by the device CRG can currently be accessed from Node 2. Should a failover or switchover occur, Node 3 would become the primary point of access for the device.

  A device CRG requires a resilient device called an independent disk pool (also called an independent auxiliary storage pool or independent ASP) to be configured on an external device, an expansion unit (tower) or IOP in a logical partition. See Independent disk pools for more complete documentation on switchable independent disk pools.

  The nodes in the recovery domain of a device CRG must also be members of the same device domain. The example below illustrates a device CRG with Node L and Node R in its recovery domain. Both nodes are also members of the same device domain. See "Device domains" on page 12 for more details.



The elements of a cluster

The following are constructs, events, actions, and terms associated with iSeries(TM) clustering:

Cluster: An iSeries cluster is a collection of one or more servers that work together as a single server. The following are elements of a cluster:

- "Cluster node" on page 7: A cluster node is an iSeries server or a logical partition that is a member of a cluster.
- "Cluster resource group" on page 7: A cluster resource group (CRG) is an OS/400(R) system object that is a set or grouping of cluster resources that define actions to be taken during a switchover or failover. The cluster resource group identifies two important elements:
  - "Cluster resource group exit programs" on page 8: Cluster resource group exit programs manage the movement of the access point of a resilient resource.
  - "Recovery domain" on page 8: A recovery domain is a subset of nodes in a cluster grouped together to provide availability for one or more resources. A domain represents the nodes of the cluster where a cluster resource exists.
- **cluster resource services**: Cluster resource services is the set of OS/400 system service functions that support iSeries cluster implementations.
- "Cluster version" on page 10: The cluster version identifies the communication level of the nodes in the cluster.
- "Device domains" on page 12: A device domain is a subset of nodes in a cluster grouped together to share device resources.
- **"Resilient resources" on page 13**: A resilient resource is a device, data, or an application that can be recovered if a node in a cluster fails. The types of resilient resources include:

– “Resilient applications” on page 14: Resilient applications are applications that can be restarted on a different cluster node without requiring you to reconfigure the clients.
– “Resilient data” on page 14: Resilient data is data that is replicated, or copied, on more than one node in a cluster.
– “Resilient devices” on page 14: Resilient devices are physical resources, represented by a configuration object such as a device description, that are accessible from more than one node in a cluster through the use of switched disk technology and independent disk pools.

Cluster events
The following are events, actions, and services that occur within a cluster:

- “Failover” on page 15: A failover is a cluster event where the primary database server, application server, and device server automatically switches over to a backup system due to the failure of the primary server, without any manual intervention.
- “Switchover” on page 17: A switchover is a cluster event where the primary database server, application server, or device server switches over to a backup system due to the manual intervention from the cluster management interface.
- **join**: Join means to become a new member of a cluster.
- **“Rejoin” on page 17**: Rejoin means to become an active member of a cluster after having been a nonparticipating member.
- **“Merge” on page 19**: A merge occurs when a node or nodes rejoin the cluster after a cluster partition has occured.
- “Replication” on page 19: Replication is the process of copying objects from one node in a cluster to one or more other nodes in the cluster, which makes the objects on all the systems identical.
- “Heartbeat monitoring” on page 19: Heartbeat monitoring ensures that each node is active by sending a signal around the cluster to detect activity.
- “Reliable message function” on page 21: The reliable message function of cluster resource services keeps track of each node in a cluster and ensures that all nodes have consistent information about the state of cluster resources.
- **“Cluster partition” on page 22**: A cluster partition is a subset of the active cluster nodes that result from a network failure. Members of a partition maintain connectivity with each other.

Cluster node
A **cluster node** is an iSeries(TM) server or logical partition that is a member of a cluster.

Each cluster node is identified by an 8-character cluster node name that is associated with one or more IP addresses that represent an iSeries server. When configuring a cluster, you can use any name that you want for a node in the cluster. However, it is recommended that the node name be the same as the host name or the system name.

Cluster communications makes use of the TCP/IP protocol suite to provide the communications paths between cluster services on each node in the cluster. The set of cluster nodes that are configured as part of the cluster is referred to as the **cluster membership list**.

Cluster resource group
A **cluster resource group (CRG)** is an OS/400(R) system object that is a set or grouping of cluster resources that define actions to be taken during a switchover or failover. The group identifies two important elements:

- “Recovery domain” on page 8
- “Cluster resource group exit programs” on page 8 that manages cluster-related events for that group - one such event would be moving an access point from one node to another node

A collection of related cluster resources that defines actions to be taken during a switchover operation of the access point of resilient resources. The group describes a recovery domain and supplies the name of the cluster resource group exit program that manages the movement of an access point.

Cluster resource group objects are defined as data resilient, application resilient, or device resilient. Data resiliency enables multiple copies of data to be maintained on more than one node in a cluster and enables the point of access to be changed to a backup node. Application resiliency enables an application program to be restarted on either the same node or a different node in the cluster. Device resiliency enables a device resource to be moved (switched) to a backup node.

Each data and application cluster resource group has a cluster resource group exit program associated with it. The exit program is optional for resilient device cluster resource groups.

In iSeries(TM) Navigator, cluster resource groups are referred to differently.

- A device CRG is referred to as a **switchable hardware group**.
- An application CRG is referred to as a **switchable software product**.
- A data CRG is referred to as a **switchable data group**.

See Managing the processing of cluster resource groups for more details.

Cluster resource group exit programs

Cluster resource group exit programs manage the movement of the access point of a resilient resource. Cluster resource group exit programs are called during different phases of a cluster environment. These programs establish and manage the environment necessary for data, device, or application resiliency within a cluster. They are called when a cluster event that impacts a cluster resource group occurs to gracefully handle the processing of the cluster event, such as a switchover or a failover. Exit programs are written or provided by cluster middleware business partners and by cluster-aware application program providers.

For detailed information on the cluster resource group exit programs, including what information is passed to them for each action code, see Cluster Resource Group Exit Program in the cluster API documentation.

Recovery domain

A **recovery domain** is a subset of cluster nodes that are grouped together in a cluster resource group (CRG) for a common purpose such as performing a recovery action. A domain represents those nodes of the cluster from which cluster resource can be accessed. This subset of cluster nodes that is assigned to a particular cluster resource group either supports the primary point of access, secondary (backup) point of access, or replicate.

The three types of roles a node can have in a recovery domain are:

Primary

The cluster node that is the primary point of access for the resilient cluster resource.
- For a data CRG, the primary node contains the principle copy of a resource.
- For an application CRG, the primary node is the system on which the application is currently running.
- For a device CRG, the primary node is the current owner of the device resource.

If the primary node for a CRG fails, or a manual switchover is initiated, all CRG objects fail or switch over to a backup node.

Backup
The cluster node that will take over the role of primary access if the present primary node fails or a manual switchover is initiated. For a data CRG, this cluster node contains a copy of that resource which is kept current with replication.

Replicate
A cluster node that has copies of cluster resources, but is unable to assume the role of primary or backup. Failover or switchover to a replicate node is not allowed. If you ever want a replicate node to become a primary, you must first change the role of the replicate node to that of a backup node. This can be accomplished by "Change the recovery domain for a cluster resource group" on page 83.

The switchover and failover order is the relationship (or order) that you have defined among the primary node and backup nodes in a recovery domain. In a recovery domain, there can be multiple backup nodes. You specify one node as first backup, another as second backup, and so on. If a primary node fails, the access point for the resilient resources switches to the first active backup node.

Each node in the recovery domain has a role with respect to the current operational environment of the cluster. This is called its **current role** in the recovery domain. As the cluster goes through operational changes such as nodes ending, nodes starting, and nodes failing, the node's current role is changed accordingly. Each node in the recovery domain also has a role with respect to the preferred or ideal cluster environment. This is called its **preferred role** in the recovery domain. The preferred role is a static definition that is initially set when the cluster resource group is created. As the cluster environment changes, this role is not changed. The preferred role is only changed when nodes are added or removed from the recovery domain, or when a node is removed from the cluster. You can also manually change the preferred roles. See "Change the recovery domain for a cluster resource group" on page 83 for details.

Conceptually, you can view the recovery domain as follows:

Node	Current role	Preferred role
A	Backup 1	Primary
B	Backup 2	Backup 1
C	Primary	Backup 2
D	Replicate	Replicate

In this example, Node C is serving as the current primary node. Because it has a preferred role of the second backup, Node C's current role as primary would have to have resulted from two failover/switchover actions. Upon the first failover or switchover action, the primary role moved from Node A to Node B since Node B is defined as the first backup. The second failover/switchover triggered Node C to become the primary node since it is defined as the second backup node.

Note: The role of each node in the recovery domain can also be changed manually. The above example illustrates how the roles in the recovery domain change when switchover/failover actions occur and no manual changes are made to the designation of the roles in the recovery domain.

Site name and data port IP addresses for geographic mirroring

When using geographic mirroring, the nodes in the recovery domain of a device CRG require a site name and data port IP addresses. For details, see Site name and data port IP addresses.

Cluster version

A **cluster version** represents the level of function available on the cluster. Versioning is a technique that allows the cluster to contain nodes at multiple release levels and fully interoperate by determining the communications protocol level to be used. If you are implementing a cluster that will contain nodes of varying release levels, see "Multiple-release clusters" on page 30.

There are actually two cluster versions:

Potential cluster version
: Represents the most advanced level of cluster function available for a given node. This is the version at which the node is capable of communicating with the other cluster nodes.

Current cluster version
: Represents the version currently being used for all cluster operations. This is the version of communications between the nodes in the cluster.

The potential cluster version is incremented on every OS/400(R) release which has significant new clustering functionality not available in earlier cluster versions. If the current cluster version is less than the potential cluster version, then that function cannot be used since some nodes would not be able to recognize or process the request. To take advantage of such new function, every node in the cluster would need to be at the same potential cluster version and the current cluster version must also be set to that level.

When a node attempts to join a cluster, its potential cluster version is compared against the current cluster version. If the value of the potential cluster version is not the same as current (N) or not equal to the next version level (N+1), then the node is not allowed to join the cluster. Note that the current cluster version is initially set by the first node defined in the cluster using the value specified on the create cluster API or command. See "Create a cluster" on page 78 for more information.

For example if you want V5R2 nodes to exist with V5R3 nodes you can do one of the following:

- Create the cluster on a V5R2 node and add in the V5R3 node.
- Create the cluster on a V5R3 node specifying to allow previous nodes to be added to the cluster, then add V5R2 nodes to the cluster.

In a multiple-release cluster, cluster protocols will always be run at the lowest node release level, the current cluster version. This is defined when the cluster is first created. N can either be set to the potential node version running on the node that originated the create cluster request or one cluster version previous to the originators potential node version. Nodes in the cluster can differ by at most one cluster version level.

Once all nodes in the cluster have been upgraded to the next release, the cluster version can be upgraded so that new functions are available. This can be accomplished by adjusting the cluster version. See "Adjust the cluster version of a cluster" on page 82 for more information.

Attention: When you are using switchable independent disk pools in your cluster, you cannot perform a switchover between OS/400 releases. The nodes in your cluster must be at the same release. Once you switch the independent disk pools to a later release, it cannot be switched back to the prior release.

Read more on cluster versions in the Cluster APIs documentation, including information on restrictions and how cluster versions correspond to OS/400 releases.

Upgrading to a new release

If you are upgrading a node to new release, you must ensure that the node you are upgrading has the appropriate cluster version. Clustering only supports nodes with one version difference in the same cluster. If you are upgrading in a cluster environment where all nodes are at the same release, you should upgrade all the nodes to the new release before changing the cluster version. This ensures that all functions associated to the new release will be available to you.

Before performing the actions associated with the upgrade, you should first determine the current cluster version for your cluster. You can display pertinent information about your cluster, including the cluster version, by using the Display Cluster Information (DSPCLUINF) command. You can also determine your current cluster version by using iSeries Navigator. See the topic, Adjust cluster version for instructions on verifying your current cluster version or changing the cluster version with iSeries Navigator.

The following table provides actions you will need to take when performing an upgrade in a cluster environment.

Current release of node you are upgrading	Current cluster version	Actions
V5R2	3	1. Upgrade the node to V5R3. 2. "Start a cluster node" on page 80 the upgraded node.
V5R2	2	1. "Adjust the cluster version of a cluster" on page 82 cluster version to 3. 2. Upgrade the node to V5R3. 3. "Start a cluster node" on page 80 the upgraded node. **Note:** If other nodes in your cluster are at V5R2, see Scenarios 4 and 5 below for details.
V5R1	less that or equal to 2	**Option A** 1. "Remove a cluster node" on page 82 that is being upgraded from the cluster. 2. "Adjust the cluster version of a cluster" on page 82 cluster version to 3. 3. Upgrade the node to V5R3. 4. "Add a node to a cluster" on page 80 the upgraded node back into the cluster. **Option B** 1. Upgrade all nodes to V5R2. 2. "Adjust the cluster version of a cluster" on page 82 cluster version to 3. 3. 4. Upgrade all nodes to V5R3.

Scenarios

The following scenarios can help you determine what you need to do to perform an upgrade in your cluster environment. Before performing the upgrade or any actions you should first determine the current cluster version for your cluster.

Scenario 1: Node to be upgraded is at V5R2. At least one other node in the cluster is at V5R3. Current cluster version is 3.

Action: Upgrade node to V5R3. After upgrading the node, start clustering on the upgraded node.

Scenario 2: Node to be upgraded is at V5R2. At least one other node in the cluster is at V5R2. Current cluster version is 2.

Action: Change current cluster version to 3. Upgrade the node to V5R3. Start clustering on the upgraded node.

Scenario 3: Node to be upgraded is V5R1. At least one other node in the cluster is at V5R2. Current cluster version is 2.

Action: Remove the node being upgraded to V5R3 from the cluster before the upgrade. Change the current cluster version to 3. Upgrade the node to V5R3 and add it back into the cluster.

Scenario 4: Node to be upgraded is at V5R2. Currently have V5R1 and V5R2 nodes in cluster. Current cluster version is 2. V5R2 node being upgraded to V5R3 is LESS important than the nodes staying at V5R1.

Actions:

1. "Remove a cluster node" on page 82 that is being upgraded from the cluster.
2. Upgrade the node to V5R3.
3. Upgrade the remaining nodes to at least V5R2.
4. "Adjust the cluster version of a cluster" on page 82 cluster version to 3.
5. "Add a node to a cluster" on page 80 the upgraded node back into the cluster.

Scenario 5: Node to be upgraded is at V5R2. Currently have V5R1 and V5R2 nodes in cluster. Current cluster version is 2. V5R2 node being upgraded to V5R3 is MORE important than the nodes staying at V5R1.

Actions:

1. "Remove a cluster node" on page 82 all the V5R1 nodes from the cluster.
2. "Adjust the cluster version of a cluster" on page 82 cluster version to 3.
3. Upgrade the node to V5R3.
4. "Start a cluster node" on page 80 the upgraded node.
5. As the remaining v5R1 nodes are upgraded to V5R2, they can be"Add a node to a cluster" on page 80 the back into the cluster.

Scenario 6: Node to be upgraded is at V5R1. At least one other node in the cluster is at V5R1. Current cluster version is less than or equal to 2.

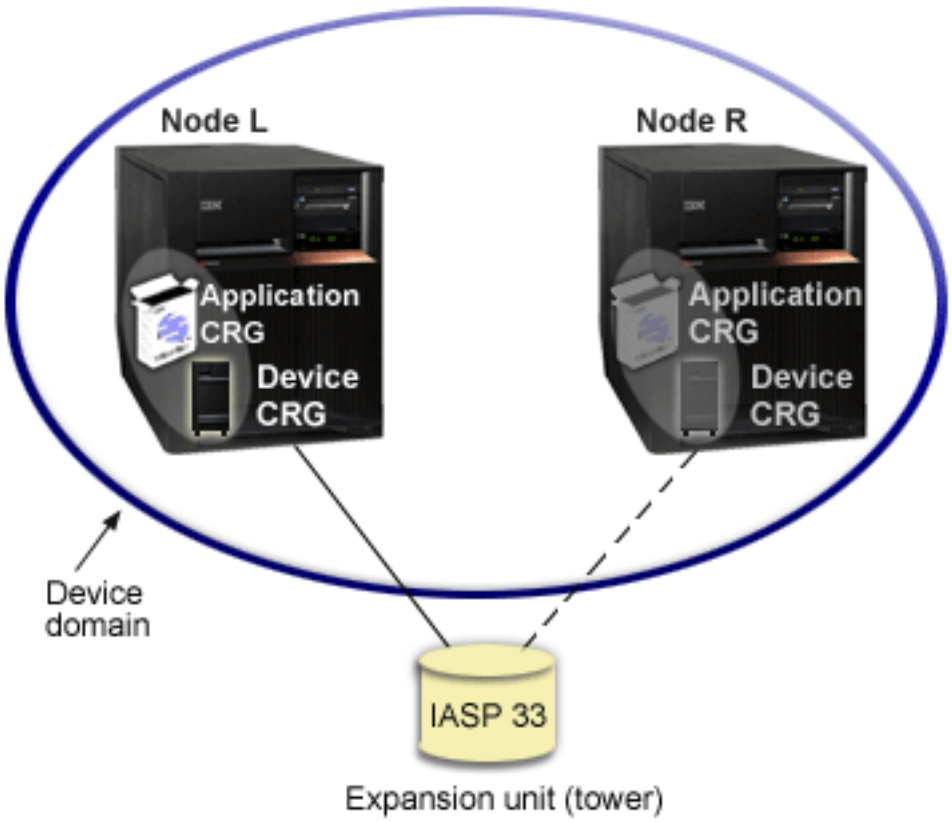
Action: Upgrade all the nodes to V5R2. Change the cluster version to 3. Upgrade all nodes to V5R3.

Device domains

A **device domain** is a subset of nodes in a cluster that share device resources. More specifically, nodes in a device domain can participate in switching action for some collection of resilient device resources. Device domains are identified and managed through a set of interfaces that allow you to add a node to a device domain or remove a node from a device domain.

Example: A switched disk cluster using independent disk pools contains an example configuration showing a device domain within a cluster.

Device domains are used to manage certain global information necessary to switch a resilient device from one node to another. All nodes in the device domain need this information to ensure that no conflicts occur when devices are switched. For example, for a collection of switchable independent disk pools, the independent disk pool identification, disk unit assignments, and virtual address assignments must be unique across the entire device domain.



A cluster node can belong to at most one device domain. Before a node can be added to the recovery domain for a device CRG, the node must be first defined as a member of a device domain. All nodes that will be in the recovery domain for a device CRG must be in the same device domain.

To create and manage device domains, you must have Option 41 (OS/400(R) - HA Switchable Resources) installed and a valid license key on your system.

For information on managing device domains, see:

- "Add a node to a device domain" on page 84
- "Remove a node from a device domain" on page 85

Resilient resources

Resilient resources are system resources, such as data, devices and applications, that are highly available if you have implemented clustering on your systems. If a cluster node that is the primary access point for a particular set of resilient resources in the cluster should incur an outage, another cluster node that is defined as the backup for that set of resources now becomes the access point.

The types of system resources that can be resilient are:

1. Data being replicated between nodes.
2. Applications using an IP address, which can be switched from one node to another.

3. Hardware devices which can be switched from one node to another.

The definition of the relationship between the nodes that are associated with a set of resilient resources is found in the **cluster resource group (CRG)** object. "Cluster resource group" on page 7 are replicated and coordinated across the nodes in the cluster through cluster resource services.

For more information see:

- "Resilient applications"
- "Resilient data"
- "Resilient devices"

Resilient applications

A **resilient application** is an application that can be restarted on a different cluster node without requiring you to reconfigure the clients. See "Make application programs resilient" on page 39 to learn about what characteristics make an application resilient.

A resilient application needs the ability to recognize the temporary loss of the Internet Protocol (IP) connection between the client and the server. The client application must be aware that the IP connection will be temporarily unavailable and must retry access rather than ending or initiating a failover. Similarly, if you are performing a switchover, server applications need to be aware that the IP connection is no longer available. Eventually, an error condition is returned to the server application. Once this error condition is received, it is best if the server application recognizes the condition and ends normally.

IP address takeover is a high availability function that is used to protect clients from application server outages. An **application takeover IP address** is a floating address that is to be associated with an application. The concept is to use IP address aliasing to define a floating IP address that is associated with multiple application servers or hosts. When one application server in a cluster fails, another cluster node assumes the responsibilities of the application server without requiring you to reconfigure the clients.

Also introduced in support of IP address takeover is the concept of application cluster resource groups (CRGs). Application CRGs are cluster resource groups that contain an application takeover IP address resource and a "Recovery domain" on page 8. The recovery domain contains the list of application servers within the cluster that support a particular application. If a single resource fails, cluster resource services initiates a failover on the group to which the failing resource belongs.

See "Cluster applications" on page 37 for more information.

Resilient data

Resilient data is data that is replicated (copied) on more than one node in a cluster. Each node in the recovery domain contains a copy of the resilient data maintained through some "Replication" on page 19 mechanism. Nodes that are defined as backup in the recovery domain can assume the role of primary point of access for the resilient data. Nodes that are defined as replicates also contain a copy of the data, but cannot assume the role of primary. Typically, data copied to a replicate node is used to offload work, such as backup or read-only queries, from the primary node.

Resilient devices

Resilient devices are physical resources, represented by a configuration object, such as a device description, that are accessible from more than one node in a cluster. In the event of an outage, the point of access for the resource is switched to the first backup node in the cluster resource group recovery domain. The type of devices that can be defined as resilient are:

Independent disk pools
: also referred to as independent ASPs, independent disk pools can go offline or come online independent of the rest of the system storage.

A **resilient device cluster resource group** can contain a list of switchable devices. Each device in the list identifies a switchable independent disk pool. The entire collection of devices are switched to the backup node when an outage occurs. Optionally, the devices can also be varied on as part of the switchover/failover process. There are limitations related to the physical configuration associated with the list of switchable devices. See Independent disk pools for more information on how to set up the appropriate configuration for an independent disk pool defined to be resilient.

A resilient device CRG is very similar to the other types of CRGs. One difference, the list of switchable devices, was mentioned above. Another difference is that the exit program is optional for a device CRG. If environment or data specific processing is needed, an exit program can be used for the CRG. See the Create Cluster Resource Group (QcstCreateClusterResourceGroup) API for additional information on this type of CRG.

Failover

A **failover** occurs when a server in a cluster automatically switches over to one or more backup servers in the event of a system failure. Contrast this with a "Switchover" on page 17, which happens when you manually switch access from one server to another. A switchover and a failover function identically once they have been triggered. The only difference is how the event is triggered.

When a failover occurs, access is switched from the cluster node currently acting as the primary node in the recovery domain of the cluster resource group to the cluster node designated as the first backup. See "Recovery domain" on page 8 for information on how the switchover order is determined.

When multiple "Cluster resource group" on page 7 (CRGs) are involved in a failover action, the system processes the device CRGs (switchable hardware groups) first, the data CRGs (switchable data groups) second, and the application CRGs (switchable software products) last.

See "Example: Failure" for different reasons of why a failover might occur.

The failover message queue receives messages regarding failover activity. You can use it to control the failover processing of a cluster resource group. See failover message queue for details.

Example: Failure

Usually, a failover results from a node failure, but there are other reasons that can also generate a failover. It is possible for a problem to affect only a single cluster resource group that can cause a failover for that cluster resource group (CRG) but not for any other CRG.

The following table shows various types of failures and the category they fit in:

Failure	General category
CEC hardware failure (CPU, for example)	2
Communications adapter, line, or router failure; or ENDTCPIFC affecting all IP interface addresses defined for the node	4
Power loss to the CEC	1
Operating system software machine check	2
ENDTCP(*IMMED or *CNTRLD with a time limit) is issued	1
ENDSBS QSYSWRK(*IMMED or *CNTRLD with a time limit) is issued	1
ENDSBS(*ALL, *IMMED, or *CNTRLD with a time limit) is issued	1
ENDSYS (*IMMED or *CNTRLD with a time limit) is issued	1
PWRDWNSYS(*IMMED or *CNTRLD with a time limit) is issued	1
Initial program load (IPL) button is pushed while cluster resource services is active on system	1

Cancel Job *IMMED or *CNTRLD with a time limit of the QCSTCTL job is issued	1
Cancel Job *IMMED or *CNTRLD with a time limit of the QCSTCRGM job is issued	1
Cancel Job *IMMED or *CNTRLD with a time limit of a cluster resource group job is issued	3
End Cluster Node API is called	1
Remove Cluster Node API is called	1
Cluster resource group job has a software error that causes it to end abnormally	3
Enters function 8 or function 3 from control panel to power down system	2
Enters function 7 for a delayed power down of a partition	1
Application program failure for an application cluster resource group	3

General category:

1. All of cluster resource services (CRS) fails on a node and is detected as a node failure. The node may actually be operational or the node may have failed (for example, a system failure due to power loss). If all of cluster resource services fails, then the resources that are managed by CRS will go through the failover process.
2. All of CRS fails on a node but it is detected as a cluster partition. The node may or may not be operational.
3. A failure occurs on an individual cluster resource group. These conditions are always detected as a failure.
4. A failure occurs but the node and cluster resource services are still operational and it is detected as a cluster partition.

When a failure occurs, the action taken by cluster resource services for a specific cluster resource group depends on the type of failure and the state of the cluster resource group. However, in all cases, the exit program is called. A failover may have to work with a list of failed nodes. When the exit program is called, it needs to determine if it must deal with only a single node failure or with a list of failed nodes.

If the cluster resource group is *inactive*, the membership status of the failed node in the cluster resource group's recovery domain is changed to either an *Inactive* or *Partition* status. However, the node roles are not changed, and the backup nodes are not reordered. The backup nodes are reordered in an inactive cluster resource group when the Start Cluster Resource Group (STRCRG) command or the Start Cluster Resource Group (QcstStartClusterResourceGroup) API is called. But, the Start Cluster Resource Group API will fail if the primary node is not active. You must issue the Change Cluster Resource Group (CHGCRG) command or the Change Cluster Resource Group (QcstChangeClusterResourceGroup) API to designate an active node as the primary node, then call the Start Cluster Resource Group API again.

If the cluster resource group is *active* and the failing node is *not* the primary node, the failover updates the status of the failed recovery domain member in the cluster resource group's recovery domain. If the failing node is a backup node, the list of backup nodes is reordered so that active nodes are at the beginning of the list.

If the cluster resource group is *active* and the recovery domain member *is* the primary node, the following actions are performed based on which type of failure has occurred:

Category 1 Failure

Failover occurs. The primary node is marked *inactive* in each cluster resource group and made the last backup node. The node that was the first backup becomes the new primary node. All device cluster resource groups failover first. Then, all data cluster resource groups failover. Finally, all application cluster resource groups failover. If a failover for any CRG detects that none of the backup nodes are active, the status of the CRG is set to *indoubt*.

Category 2 Failure

Failover occurs but the primary node does not change. All nodes in the cluster partition that do not have the primary node as a member of the partition will end the active cluster resource group. The status of the nodes in the recovery domain in the cluster resource group are set to a *partition* status for each node that is in the primary partition. If a node really failed but is

detected only as a partition problem and the failed node was the primary node, you lose all the data and application services on that node and no automatic failover is started. You must either bring the node back up and start clustering on that node again. See “Change partitioned nodes to failed” on page 94 for more information.

Category 3 Failure
If only a single cluster resource group is affected, failover occurs on an individual basis because cluster resource groups are independent of each other. It may happen that several cluster resource groups are affected at the same time due to someone canceling several cluster resource jobs. However, the type of failure is handled on a CRG by CRG basis, and no coordinated failover between CRGs is performed. The primary node is marked as *inactive* in each cluster resource group and made the last backup node. The node that was the first backup node becomes the new primary node. If there is no active backup node, the status of the cluster resource group is set to *indoubt*.

Category 4 Failure
This category is similar to category 2. However, while all nodes and cluster resource services on the nodes are still operational, not all nodes can communicate with each other. The cluster is partitioned, but the primary node or nodes are still providing services. However, because of the partition, you may experience various problems. For example, if the primary node is in one partition and all the backup nodes or replicate nodes are in another partition, you are no longer replicating data and have no protection should the primary node fail. In the partition that *contains* the primary node, the failover process updates the status of the nodes in the cluster resource group's recovery domain to *partition* for all nodes in the other partition. In the partition that *does not* contain the primary node, the status of the nodes in the cluster resource group's recovery domain for all nodes in the other partition is set to *partition.*

Switchover

A **switchover** happens when you manually switch access to a resource from one server to another. You would usually initiate a manual switchover if you wanted to perform system maintenance, such as applying program temporary fixes (PTFs), installing a new release, or upgrading your system. Contrast this with a “Failover” on page 15, which happens automatically when a outage occurs on the primary node.

When a switchover occurs, access is switched from the cluster node currently acting as the primary node in the recovery domain of the cluster resource group to the cluster node designated as the first backup. See “Recovery domain” on page 8 for information on how the switchover order is determined.

If you are doing an administrative switchover of multiple CRGs, the order you specify should consider the relationships between the CRGs. For example, if you have an application CRG that depends on data associated with a device CRG, the steps to an ordered switchover are:

1. Stop the application on the old primary node (to quiesce changes to the data).
2. Switch the device CRG to the new primary node.
3. Switch the application CRG to the new primary node.
4. Restart the application on the new primary node.

Rejoin

Rejoin means to become an active member of a cluster after having been a nonparticipating member. For example, when clustering is restarted on a node after the node has been inactive, the cluster node rejoins the cluster. You start cluster resource services on a node by starting it from a node that is already active in the cluster. Beginning with cluster version 3, a node can start itself and will be able to rejoin the current active cluster, provided it can find an active node in the cluster. See “Start a cluster node” on page 80 for details.

Suppose that nodes A, B, and C make up a cluster. Node A fails. The active cluster is now nodes B and C. Once the failed node is operational again, it can rejoin the cluster when the node is started from any

cluster node, including itself. The rejoin operation is done on a cluster resource group basis, which means that each cluster resource group (CRG) joins the cluster independently.

The primary function of rejoin ensures that the CRG object is replicated on all active recovery domain nodes. The rejoining node, as well as all existing active cluster nodes, must have an identical copy of the CRG object. In addition, they must have an identical copy of some internal data.

When a node fails, the continued calling of cluster resource services on the remaining nodes in the cluster can modify the data in a CRG object. The modification must occur due to the calling of an API or a subsequent node failure. For simple clusters, the rejoining node is updated with a copy of the CRG from some node that is currently active in the cluster. However, this may not be true in all cases.

For more details on the rejoin operation, see "Example: Rejoin."

Example: Rejoin

The following diagram describes the actions taken whenever a node rejoins the cluster. In addition, the state of the rejoining nodes will be changed from *inactive* to *active* in the membership status field in the recovery domain of the CRG. The exit program is called on all nodes in the CRG's recovery domain and is passed an action code of Rejoin.

Rejoin operation			
Rejoining node		Cluster nodes	
Contains copy of CRG	Does not contain copy of CRG	Contain copy of CRG	Do not contain copy of CRG
(1)	(2)	(3)	(4)

Using the diagram above, the following situations are possible:

1. 1 and 3
2. 1 and 4
3. 2 and 3
4. 2 and 4

If a node in the cluster has a copy of the CRG, the general rule for rejoin is that the CRG is copied from an active node in the cluster to the rejoining node.

Rejoin Situation 1

A copy of the CRG object from a node in the cluster is sent to the joining node. The result is:

- The CRG object is updated on the joining node with the data sent from the cluster.
- The CRG object may be deleted from the joining node. This can occur if the joining node was removed from the CRG's recovery domain while the joining node was out of the cluster.

Rejoin Situation 2

A copy of the CRG object from the joining node is sent to all cluster nodes. The result is:

- No change if none of the cluster nodes are in the CRG's recovery domain.
- The CRG object may be created on one or more of the cluster nodes. This can occur in the following scenario:
  - Nodes A, B, C, and D make up a cluster.
  - All four nodes are in the recovery domain of the CRG.
  - While node A is out of the cluster, the CRG is modified to remove B from the recovery domain.
  - Nodes C and D fail.

– The cluster is only node B which does not have a copy of the CRG.
– Node A rejoins the cluster.
– Node A has the CRG (although it is down level by now) and Node B does not. The CRG is created on node B. When nodes C and D rejoin the cluster, the copy of the CRG in the cluster updates node C and D and the previous change to remove node B from the recovery domain is lost.

Rejoin Situation 3

A copy of the CRG object from a node in the cluster is sent to the joining node. The result is:

- No change if the joining node is not in the CRG's recovery domain.
- The CRG object may be created on the joining node. This can occur if the CRG was deleted on the joining node while cluster resource services is not active on the node.

Rejoin Situation 4

Some internal information from one of the nodes in the cluster may be used to update information on the joining node but nothing occurs that is visible to you.

Merge

A merge operation is similar to a "Rejoin" on page 17 operation except that it occurs when nodes that are partitioned begin communicating again. The partition may be a true partition in that cluster resource services is still active on all nodes. However, some nodes can't communicate with other nodes due to a communication line failure. Or, the problem may be that a node actually failed, but was not detected as a failure.

In the first case, the partitions are merged back together automatically once the communication problem is fixed. This happens when both partitions periodically try to communicate with the partitioned nodes and eventually reestablish contact with each other. In the second case, cluster resource services must be restarted on the failed node by starting the node from any node in the cluster. See "Start a cluster node" on page 80 for details.

See Example: Merge for examples of how a merge occurs.

Replication

Replication makes a copy of something in real time. It means copying objects from one node in a cluster to one or more other nodes in the cluster. Replication makes and keeps the objects on your systems identical. If you make a change to an object on one node in a cluster, this change is **replicated** to other nodes in the cluster.

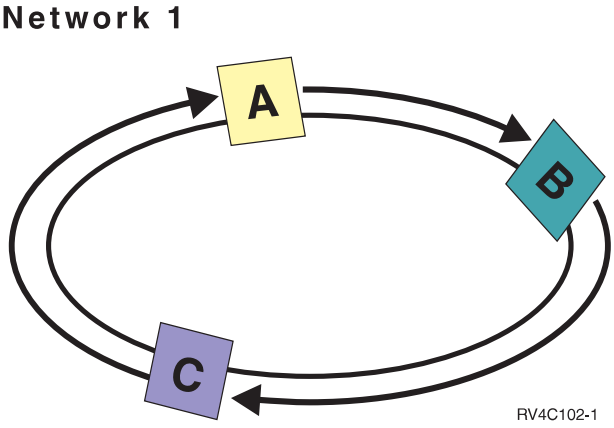
See "Plan for replication" on page 33 in order to determine how to implement replication.

Heartbeat monitoring

Heartbeat monitoring is a cluster resource services function that ensures that each node is active by sending a signal from every node in the cluster to every other node in the cluster to convey that they are still active. When the heartbeat for a node fails, the condition is reported so the cluster can automatically begin the failover process to move resilient resources to a backup node.

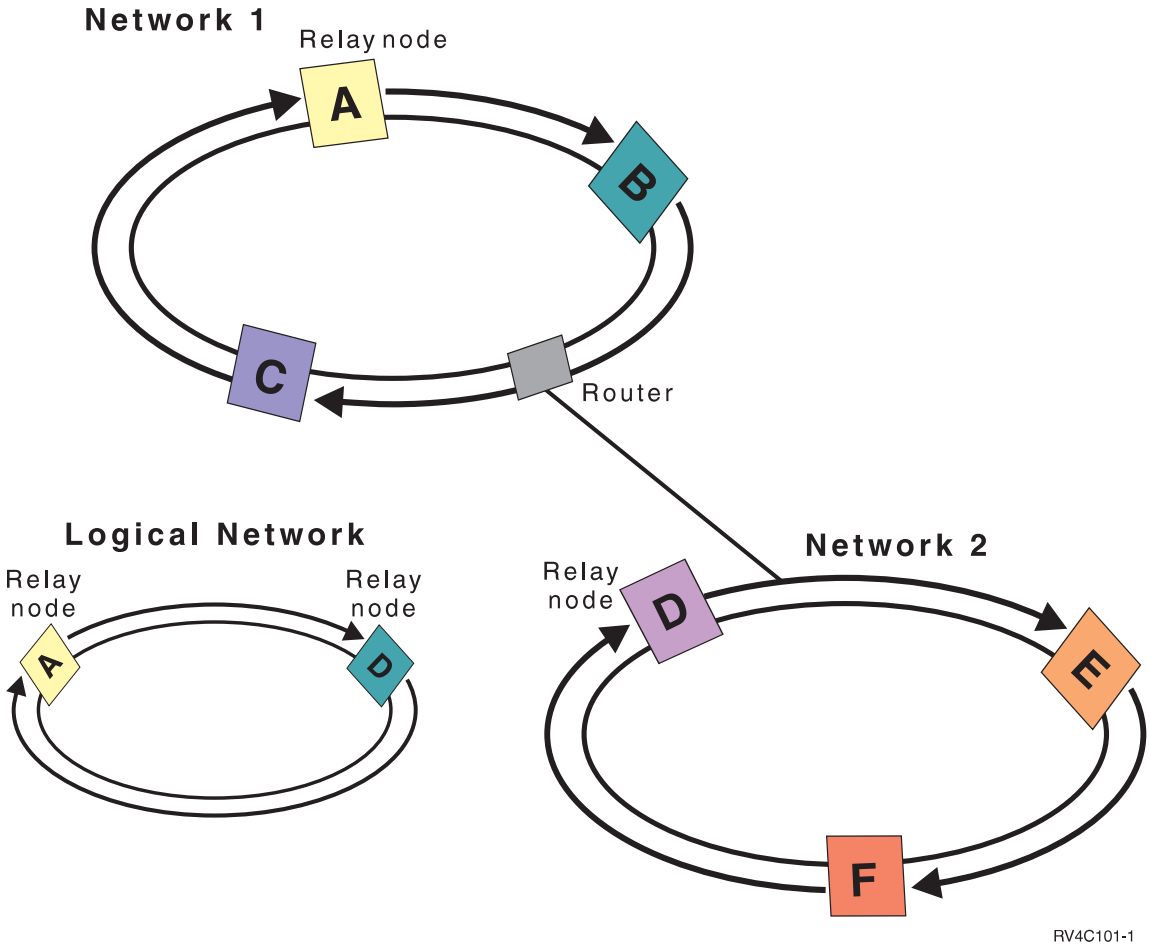
Consider the following examples to understand how heartbeat monitoring works:

Example 1



With the default (or normal) settings, a heartbeat message is sent every 3 seconds from every node in the cluster to its upstream neighbor. For example, if you configure Node A, Node B, and Node C on Network 1, Node A would send a message to Node B, Node B would send a message to Node C, and Node C would send a message to Node A. Node A expects an acknowledgment to the heartbeat from Node B as well as an incoming heartbeat from the downstream Node C. In effect, the heartbeating ring goes both ways. If Node A did not receive a heartbeat from Node C, Node A and Node B would continue to send a heartbeat every 3 seconds. If Node C missed four consecutive heartbeats, a heartbeat failure would be signaled.

Example 2



Let's add another network to this example to show how routers and relay nodes are used. You configure Node D, Node E, and Node F on Network 2. Network 2 is connected to Network 1 using a router. A router can be another iSeries(TM) server or a router box that directs communications to another router somewhere else. Every local network is assigned a relay node. This relay node is assigned to the node that has the lowest node ID in the network. Node A is assigned as the relay node on Network 1, and Node D is assigned as the relay node on Network 2. A logical network containing Node A and Node D would then be created, thereby letting Node A and Node D send heartbeats to each other. By using routers and relay nodes, the nodes on these two networks can monitor each other and signal any node failures.

Reliable message function

The **reliable message function** of cluster resource services keeps track of each node in a cluster and ensures that all nodes have consistent information about the state of cluster resources. Reliable messaging uses retry and timeout values that are unique to clustering. These values are preset to values that should suit most environments. However, they can be changed through the "Change cluster resource services settings" on page 22 interface. The message retry and timeout values are used to determine how many times a message is sent to a node before a failure or partition situation is signaled. For a local area network (LAN), the amount of time it takes to go through the number of retries before a failure or partition condition is signaled is approximately 45 seconds using the default retry and timeout values. For a remote network, more time is allowed to determine whether a failure or partition condition exists. You can figure approximately 4 minutes and 15 seconds for a remote network.

Change cluster resource services settings

The default values affecting message timeout and retry are set to account for most typical installations. However, it is possible to modify these values to more closely match your communications environment.

The values can be adjusted in one of these ways:

- Set a general performance level that matches your environment
- Set values for specific message tuning parameters for more specific adjustment.

In the first method above, the message traffic is adjusted to one of three communications levels. The normal level is the default and is described in detail in "Heartbeat monitoring" on page 19.

The second method should normally be done only under the advisement of an expert.

The Change Cluster Resource Services (QcstChgClusterResourceServices) API describes details on both methods.

Cluster partition

A **cluster partition** is a subset of the active cluster nodes that results from a communications failure. Members of a partition maintain connectivity with each other.

A cluster partition occurs in a cluster whenever communication is lost between one or more nodes in the cluster and a failure of the lost nodes cannot be confirmed. When a cluster partition condition is detected, cluster resource services limits the types of actions that you can perform on the nodes in the cluster partition. Restricting function during a partition is done so cluster resource services will be able to merge the partitions once the problem that caused.

For more on cluster partitions, see:

- "Avoid a cluster partition" on page 29
- "Partition errors" on page 93

How a cluster works

The cluster infrastructure provided as a part of OS/400(R), called cluster resource services, provides "Failover" on page 15 and "Switchover" on page 17 capabilities for your servers that are used as database servers or application servers in a client-server environment. If a system outage or a site loss occurs, the functions that are provided on a clustered database server can be switched over to one or more designated backup systems that either:

- Contain a current copy, provided through "Replication" on page 19, of your critical application data.
- Become the primary point of access for the "Resilient devices" on page 14 containing that critical data.

In either scenario, the data and applications remain available. The switching of the access point can be automatic if a system failure, or failover, should happen, or you can control how and when the transfer will take place by manually initiating a switchover.

The switchover and failover will not impact you as a system user or impact applications that you have running on an application server. You can automatically reroute data requests to the new primary node. You can easily maintain multiple replicates of the same data or store the data in a resilient device. If your clusters contain more than two nodes, you can group a system's "Resilient data" on page 14 (replicated data) together to allow different nodes to act as the backups for each group's resilient data. Multiple backup nodes can be defined. Once a node has been restarted after a failure, cluster resource services provides the means to reintroduce "Rejoin" on page 17 nodes to the cluster and restore their operational capabilities.

See "Comparison of replication, switched disks, and cross-site mirroring" on page 31 for a comparison of these technologies.

Plan for clusters

This topic covers the requirements that you will need before you can implement clustering. The following topics provide you with the general concepts, requirements, and considerations for designing a clustering solution.

For information on planning for clusters, see the following:

"Solutions for configuring and managing clusters"

Cluster resource services provides the basic cluster infrastructure. There are several methods that will allow you to take advantage of the clustering capabilities provided by cluster resource services.

"Cluster requirements" on page 26

This topic outlines the hardware, software, and communications requirements for implementing clusters.

"Design your cluster" on page 28 Identify your needs to determine how to design your cluster.

"Cluster security" on page 34

Consider some of the security issues you need to consider when you plan to implement clustering on your systems.

"Cluster configuration checklist" on page 35

Before you begin to configure your cluster, complete this checklist to ensure that your environment is prepared properly.

Solutions for configuring and managing clusters

OS/400(R) cluster resource services on the iSeries(TM) provides the basic infrastructure that allows you to implement a cluster. Cluster resource services provides a set of integrated services that maintain cluster topology, perform heartbeating, and allow creation and administration of cluster configuration and cluster resource groups. Cluster resource services also provides reliable messaging functions that keep track of each node in the cluster and ensure that all nodes have consistent information about the state of cluster resources.

While cluster resource services provides the basic cluster infrastructure, there are several methods that will allow you to take advantage of these clustering capabilities. Each has distinct benefits and capabilities. Depending upon your clustering needs, one of the following solutions will be the best fit for configuring and managing your cluster environment:

"iSeries Navigator cluster management" on page 24
IBM(R) offers a cluster management graphical user interface that allows you to create and manage a simple cluster, including one that uses switchable independent disk pools (switchable independent ASPs) to ensure data availability.

"Cluster commands and APIs" on page 25
OS/400 cluster resource services provides a set of control language (CL) commands, application program interfaces (APIs) and facilities that can be used by iSeries application providers or customers to enhance their application availability.

"Cluster middleware business partners and available clustering products" on page 25
You can purchase a product from an IBM cluster middleware business partner that provides the replication functions that are integral to clustering and simplifies cluster creation and management.

Important: Use only one of these solutions exclusively. Conflicts, problems and unpredictability can arise when attempting to use more than one solution to create and manage a cluster. The information you find in the iSeries Information Center documents procedures specific to iSeries Navigator and the cluster resource services CL commands and APIs. If you use a cluster middleware business partner solution, see the documentation provided with the product for procedural information on performing tasks.

iSeries Navigator cluster management

IBM(R) offers a cluster management interface that is available through iSeries(TM) Navigator and accessible through Option 41 (OS/400 - HA Switchable Resources). This interface allows you to create and manage a cluster that uses switchable independent disk pools (switchable independent ASPs) to ensure data availability. See iSeries Navigator for more information on the iSeries Navigator interface.

Important: The iSeries Navigator cluster management interface does not contain all of the capabilities provided by cluster resource services. While iSeries Navigator provides many functions necessary to configure and manage a cluster, be aware that there are some capabilities that are only available through the cluster commands and APIs, or perhaps through a cluster middleware business partner application, depending upon the particular application. For example, the iSeries clustering architecture supports up to 128 nodes in a cluster, however the iSeries Navigator interface only supports up to four nodes in a cluster. With iSeries Navigator, you can create a simple cluster consisting of one or two nodes. Once you have established a cluster in iSeries Navigator, you can then add a node to an existing cluster, up to as many as four total nodes. If your clustering needs exceed this, you should consider using the "Cluster commands and APIs" on page 25 or "Cluster middleware business partners and available clustering products" on page 25.

iSeries Navigator cluster management features a wizard which steps you through the creation of a simple, two-node cluster. Additional cluster management can be accomplished using this interface, including tasks such as:

- Adding a node to an existing cluster
- Adding a switchable hardware group to a cluster
- Adding a switchable software product to a cluster
- Adding a switchable data group to a cluster
- Change the role of nodes in the recovery domain
-

  Edit the site name and data port IP addresses for a node in the recovery domain of a switchable hardware group

- Changing the cluster description
- Changing the exit program name for a cluster resource group
- Changing the takeover IP address for a switchable software product
- Deleting a cluster
- Starting clustering
- Stopping clustering
- Switching cluster resources from the primary node to the backup node
- Viewing messages about cluster activity

The online help available in iSeries Navigator provides step-by-step procedures on how to accomplish these tasks.

Note: The iSeries Navigator cluster management interface does not support logical object replication. For replication, you should consider the clustering products available from high availability business partners. See "Cluster middleware business partners and available clustering products" for details.

For more information on iSeries Navigator cluster management, see "Frequently asked questions about iSeries Navigator cluster management" on page 98.

Cluster commands and APIs

You can write your own custom application to configure and manage your cluster by utilizing cluster control language (CL) commands and application programming interfaces (APIs). These commands and APIs take advantage of the technology provided by cluster resource services provided as a part of OS/400(R).

For a complete listing of the capabilities of the cluster commands and APIs, see Cluster CL command and API descriptions.

QUSRTOOL

Cluster resource services also provides a set of example commands in the QUSRTOOL library that map to the APIs that have no supported command interface. The QUSRTOOL commands might be useful in some environments. For example, you can change heartbeating or send information around the cluster. See the member TCSTINFO in the file QUSRTOOL/QATTINFO for more information on these example commands. An example application CRG exit program is also included in the QUSRTOOL library. The sample source code can be used as the basis for writing an exit program. Sample source, TCSTDTAEXT, in file QATTSYSC contains a source for a program to create the QCSTHAAPPI and QCSTHAAPP0 dataareas, and QACSTOSDS (object specifier) file.

Cluster middleware business partners and available clustering products

IBM(R) cluster middleware business partners provide software solutions for dedicated replication and cluster management functions. If you would like to purchase a product that provides replication functions that are integral to clustering and simplifies cluster creation and management, contact your IBM marketing representative or business partner. They can provide a complete list of clustering enabling products provided by IBM cluster middleware business partners.

The cluster middleware business partner cluster management product:

- Provides the user interface to define and maintain the cluster configuration
- Provides the user interface to define and manage the device, data, and application cluster resource groups
- Maintains the knowledge through the use of cluster APIs, of what cluster resource groups are defined in the cluster and what relationships are required.
- Creates the device, data, and application cluster resource groups.

The cluster middleware business partner replication product:

- Builds the middleware's control structures that identify the data and objects that are required to be resilient.
- Creates the cluster resource group for critical data and associates that object with its control structures.
- Provides the exit program for the data cluster resource group.

Cluster requirements

This topic outlines the requirements for implementing clusters. The requirements vary depending upon which cluster capabilities you choose to implement. For example, you may choose to implement a simple, two-node cluster to take advantage of replication. Or you may choose to implement a cluster designed to take advantage of switched disks and switchable independent disk pools. See “Examples: Cluster configurations” on page 89 for details on some common cluster implementations.

Review the following cluster requirements:

- “Hardware requirements for clusters”
- “Software and licensing requirements for clusters”
- “Communications requirements for clusters” on page 27

Hardware requirements for clusters

Any iSeries(TM) model that is capable of running OS/400(R) V4R4M0 or later is compatible for implementing clustering.

In addition, you should provide protection from a power loss through an external uninterruptible power supply or equivalent. Otherwise, a sudden power loss on a cluster node could result in a “Cluster partition” on page 22 state rather than a “Failover” on page 15.

Clustering utilizes Internet Protocol (IP) multicast capabilities. Multicast does not map well to all types of physical media. For more information on multicast restrictions that may apply to your particular hardware, see the TCP/IP Configuration and Reference

.

You can protect your disks with mirrored protection or device parity protection. Using these solutions on your primary system prevents a failover from occurring should a protected disk fail. It also is a good idea to have these solutions on your backup system in case a failover should occur. See Disk protection for details.

Note: If you plan to use independent disk pools in your cluster, refer to the independent disk pools Hardware requirements topic.

Software and licensing requirements for clusters

In order to implement clustering, you must have the following software and licenses:

1. OS/400(R) V4R4M0[1] or later configured with TCP/IP (TCP/IP Connectivity Utilities)
2. A cluster configuration and management software solution. This can be any of the following:
   - iSeries(TM) Navigator cluster management
   - A cluster middleware business partner solution
   - Your own cluster management application program written using cluster resource services commands and APIs

   See “Solutions for configuring and managing clusters” on page 23 for details on choosing a solution that is best for you.

Important: If you plan to implement independent disk pools to take advantage of switchable devices, there are additional requirements. See Plan for independent disk pools for details.

[1] OS/400 V5R1M0 can be used for implementing independent disk pools containing user-defined file systems (UDFS) only. Support for library-based objects is only available starting with OS/400 V5R2M0. See “Cluster version” on page 10 for a discussion of multiple-release clusters and how to adjust your cluster version.

Communications requirements for clusters

You can use any type of communications media in your clustering environment as long as it supports Internet Protocol (IP). Cluster resource services uses only TCP/IP protocols to communicate between nodes. Local area networks (LANs), wide area networks (WANs), OptiConnect system area networks (SANs), or any combination of these connectivity devices are supported. Your choice should be based upon:

- Volume of transactions
- Response time requirements
- Distance between the nodes
- Cost considerations

You can use these same considerations when determining the connection media to be used to connect primary and backup locations of resources. When planning your cluster, it is recommended that you designate one or more of your backup nodes in remote locations in order to survive a site loss disaster.

To avoid performance problems that might be caused by inadequate capacity, you need to evaluate the communication media that is used to handle the volumes of information that is sent from node to node. You can choose which physical media you prefer to use such as token ring, Ethernet, asynchronous transfer mode (ATM), SPD OptiConnect, High-Speed Link (HSL) OptiConnect, or Virtual OptiConnect (a high-speed internal connection between logical partitions).

HSL OptiConnect is a technology provided by OptiConnect for OS/400(R) software (OS/400 Option 23 - OS/400 OptiConnect). It can be used to construct highly available solutions. HSL OptiConnect is a system area network that provides high-speed, point-to-point connectivity between cluster nodes by using High Speed Link (HSL) Loop technology. HSL OptiConnect requires standard HSL cables, but no additional hardware. For additional information on HSL OptiConnect, see OptiConnect for OS/400

.

For switchable hardware, also referred to as resilient device CRGs, you need to have an independent disk pool that is switchable in your environment. In a logical partition environment, this is a collection of disk units that are on the bus that is being shared by the logical partitions, or that are attached to an input/output processor that has been assigned to an I/O pool. For a multi-system environment, this is one or more switchable expansion units (towers) properly configured on the HSL loop also containing the systems in the recovery domain. The switchable tower can also be used in an LPAR environment. For more planning information on switchable hardware and independent disk pools, see Plan for independent disk pools.

Note: If you are using 2810 LAN adapters using **only** TCP/IP, and not using Systems Network Architecture (SNA) or IPX, you can increase your adapter performance on a V4R5M0 server by specifying Enable only for TCP(*YES) for your specific line description using the Work with Line Descriptions (WRKLIND) command. Enable only for TCP(*YES) is set automatically in V5R1M0 and later releases.

Design your cluster

Because there are a variety of ways to implement clustering depending on what you are hoping to achieve, it is important to spend some time identifying your needs to determine how to design your cluster. Use the following topics to help you determine exactly how to design your cluster:

- "Design your network for clusters"
- "Multiple-release clusters" on page 30
- "Identify servers to include in a cluster" on page 30
- "Identify applications to include in a cluster" on page 30
- "Plan for data resilience" on page 30

Design your network for clusters

Before you configure your networks for clustering, you need to plan carefully and do some initial pre-cluster configuration involving TCP/IP. It is important that you read these topics before configuring your cluster. They will tell you when or how to:

- "Set up IP addresses"
- "Set TCP/IP configuration attributes"
- "Avoid a cluster partition" on page 29

For information about setting up redundant communications paths and whether you need to have a dedicated network for clustering, see "Dedicate a network for clusters" on page 29.

See "Tips: Cluster communications" on page 29 for general cluster communications hints.

Set up IP addresses: All the nodes in a cluster must be interconnected using Internet Protocol (IP). Because cluster resource services uses **only** IP to communicate with other cluster nodes, all cluster nodes must be *IP-reachable*. This means that you must have IP interfaces configured to connect the nodes in your cluster. These IP addresses must be set up either manually by the network administrator in the TCP/IP routing tables on each cluster node or they may be generated by routing protocols running on the routers in the network. This TCP/IP routing table is the map that clustering uses to find each node; therefore, each node must have its own **unique** IP address. Each node may have up to two IP addresses assigned to it. These addresses must not be changed in any way by other network communications applications. Be sure when you assign each address that you take into account which address use which kind of communication line. If you have a preference for using a specific type of communication media, make sure to configure the first IP address using your preferred media. The first IP address is what is treated preferentially by the "Reliable message function" on page 21 and "Heartbeat monitoring" on page 19.

Note: You need to be sure that the loop back address (127.0.0.1) is active for clustering. This address, which is used to send any messages back to the local node, is normally active by default. However, if it has been ended by mistake, cluster messaging cannot function until this address has been restarted.

Set TCP/IP configuration attributes: To enable cluster resource services, certain attribute settings are required in the TCP/IP configuration of your network. You must set these attributes before you can add any node to a cluster:

- Set IP datagram forwarding to *YES using the CHGTCPA (Change TCP/IP Attributes) command if you plan to use an iSeries(TM) server as the router to communicate with other networks and you have no other routing protocols running on that server.
- Set the INETD server to START. See INETD server for information on starting the INETD server.
- Set User Datagram Protocol (UDP) CHECKSUM to *YES using the CHGTCPA (Change TCP/IP Attributes) command.
- Set MCAST forwarding to *YES if you are using bridges to connect your token ring networks.
- If you are using Opticonnect for OS/400(R) to communicate between cluster nodes, start the QSOC subsystem by specifying STRSBS(QSOC/QSOC).

Tips: Cluster communications: Consider these tips when you set up your communications paths:

- Be sure you have adequate bandwidth on your communication lines to handle the non-cluster activity along with the clustering heartbeating function and continue to monitor for increased activity.
- For best reliability, do not configure a single communication path linking one or more nodes.
- Do not overburden the line that is responsible for ensuring that you are still communicating with a node.
- Eliminate as many single points of failure as possible such as having two communication lines coming into a single adapter, same input-output processor (IOP), or same tower.
- If you have an extremely high volume of data being passed over your communication lines, you may want to consider putting "Plan for replication" on page 33 and "Reliable message function" on page 21 on separate networks.

- If you are using Internet Protocol (IP) multicast, you should see the TCP/IP Configuration and Reference

  for multicast restrictions that may apply to different types of physical media.
- User Datagram Protocol (UDP) multicast is the preferred protocol that the cluster communications infrastructure uses to send cluster management information between nodes in a cluster. When the physical media supports multicast capabilities, cluster communications utilizes the UDP multicast to send management messaging from a given node to all local cluster nodes that support the same subnet address. Messages that are sent to nodes on remote networks are always sent using UDP point-to-point capabilities. Cluster communications does not rely on routing capability for multicast messages.
- The multicast traffic that supports cluster management messaging tends to fluctuate by nature. Depending on the number of nodes on a given LAN (that supports a common subnet address) and the complexity of the cluster management structure that is chosen by the cluster administrator, cluster related multicast packets can easily exceed 40 packets per second. Fluctuations of this nature could have a negative impact on older networking equipment. One example would be congestion problems on devices on the LAN that serve as Simple Network Management Protocol (SNMP) agents that need to evaluate every UDP multicast packet. Some of the earlier networking equipment does not have adequate bandwidth to keep up with this type of traffic. You need to ensure that you or the network administrator has reviewed the capacity of the networks to handle UDP multicast traffic to make certain that clustering does not have a negative impact on the performance of the networks.

Avoid a cluster partition: A "Cluster partition" on page 22 is not aways avoidable. Power loss and hardware failure are two examples. However, the typical network related cluster partition can best be avoided by configuring redundant commuincations paths between all nodes in the cluster. A **redundant communications path** means that you have two lines configured between two nodes in a cluster. If a failure on the first communication path should occur, the second communication path can take over to keep communications running between the nodes, thereby minimizing conditions that could put one or more nodes of the cluster into a cluster partition. One thing to consider when configuring these paths is that if both of your communications lines go into the same adapter on the system, these lines are still at risk if this single adapter fails.

Read "Tips: Cluster communications" for general cluster communications hints.

See "Partition errors" on page 93 if you have encountered a cluster partition.

Dedicate a network for clusters: Clustering does not require you to have a dedicated network just for clustering use. During normal operations, base clustering communication traffic will be minimal. It is, however, highly recommended that you have redundant communication paths configured for each node in a cluster. By configuring two lines, you can dedicate one line for clustering traffic and the other line can handle the normal traffic and also be the backup line if the dedicated line for clustering goes down.

See “Avoid a cluster partition” on page 29 for more information on why configuring two communication paths is a good idea.

Multiple-release clusters

If creating a cluster that will include nodes at multiple “Cluster version” on page 10, then certain steps are required when you create the cluster. By default, the current cluster version will be set to the potential cluster version of the first node added to the cluster. This approach is appropriate if this node is at the lowest version level to be in the cluster. However, if this node is at a later version level, then you will subsequently be unable to add nodes with a lower version level. The alternative is to use the target cluster version value on create cluster to set the current cluster version to one less than the potential cluster version of the first node added to the cluster.

For example, consider the case where a two-node cluster is to be created. The nodes for this cluster are:

Node Identifier	Release	Potential Cluster Version
Node A	V5R2	3
Node B	V5R3	4

If the cluster is to be created from Node B, care must be taken to indicate that this will be a mixed release cluster. The target cluster version must be set to indicate that the nodes of the cluster will communicate at one less than the requesting node’s potential node version.

Identify servers to include in a cluster

In order to identify the servers that you want to include in a cluster, you need to decide which servers are capable of providing adequate backup for the data and applications you need to run your business. You need to determine:

- Which servers contain your critical data and critical applications?
- Which servers will be the backup for those systems?

Once you have determined this, these are the servers that you will want to include in your cluster.

Identify applications to include in a cluster

Not every application will give you the availability benefits of clustering. An application must be resilient in order to take advantage of the switchover and failover capabilities provided by clustering. Application resilience allows the application to be restarted on the backup node without having to reconfigure the clients using the application. Therefore your application must meet certain requirements to take full advantage of the capabilities offered by clustering.

See the “Cluster applications” on page 37 topic for more information on resilient applications.

Plan for data resilience

Data resilience is achieved when data is always available to an end user or application. You can achieve data resilience through the use of replication or switchable independent disk pools. The following topics will help you prepare your cluster for data resilience:

“Determine which data should be made resilient” on page 31
Understand what types of data you should consider making resilient.

“Comparison of replication, switched disks, and cross-site mirroring” on page 31
Determine which technology is right for your cluster.

“Plan for replication” on page 33
Multiple copies of the data are maintained with replication. Data is replicated, or copied, from the

primary node in the cluster to the backup nodes designated in the recovery domain. When an outage occurs on the primary node, the data remains available as a designated backup node takes over as the primary point of access.

"Plan for switchable independent disk pools and geographic mirroring" on page 33
A single copy of the data is maintained on switchable hardware; either an expansion unit (tower) or an IOP in a logical partition environment. When an outage occurs on the primary node, access to the data on the switchable hardware switches to a designated backup node.

Additionally, independent disk pools can be used in a cross-site mirroring (XSM) environment. This allows a mirror copy of the independent disk pool to be maintained on a system that is (optionally) geographically distant from the originating site for availability or protection purposes.

Determine which data should be made resilient: Determining which data you need to make resilient is similar to determining which kind of data you need to backup and save when you prepare a back up and recovery strategy for your systems. You need to determine which data in your environment is critical to keeping your business up and running.

For example, if you are running a business on the Web, your critical data may be:

- Today's orders
- Inventory
- Customer records

In general, information that does not change often or that you do not need to use on a daily basis probably does not need to be made resilient. See Plan a backup and recovery strategy in the Backup and recovery topic for more information on what types of data should be made resilient.

Comparison of replication, switched disks, and cross-site mirroring: The primary advantages a clustered environment provides are those of replication, switchability, and cross-site mirroring (XSM).

Replicated resource

Replication is the process of copying objects from one node in a cluster to one or more other nodes in the cluster, which makes the objects on all the systems identical. In the figure above, two identical copies of the data are kept on two separate cluster nodes.

A replicated resource allows for objects, such as an application and its data, to be copied from one node in the cluster to one or more other nodes in the cluster. This process keeps the objects on all servers in the resource's recovery domain identical. If you make a change to an object on one node in a cluster, the change is replicated to other nodes in the cluster. Then, should a failover or switchover occur, the backup node can seamlessly take on the role of the primary node. The server or servers that act as backups are defined in the recovery domain. When an outage occurs on the server defined as the primary node in the recovery domain and a switchover or failover is initiated, the node designated as the backup in the recovery domain becomes the primary access point for the resource.

Replication requires the use of either a custom-written application or a software application written by a cluster middleware business partner. See "Plan for replication" on page 33 for details.

Switchable resource

Switchable resources enable the resources, such as data and applications, residing on an expansion unit or on an input-output processor (IOP) on a shared bus or in an I/O Pool for a logical partition, to be switched between a cluster's primary node and backup node. This allows for a set of disk units to be accessed from a second server, a server defined as a backup node in the cluster resource group's recovery domain, when the server currently using those disk units experiences an outage and a failover or switchover occurs. In the figure above, there is only one copy of the data for which both nodes can serve as the primary access point.

Taking advantage of switchable resources in your cluster requires the use of independent disk pools. See "Plan for switchable independent disk pools and geographic mirroring" on page 33 for more information.

Cross-site mirroring

Cross-site mirroring, combined with the geographic mirroring function, enables you to mirror data on disks at sites that can be separated by a significant geographic distance. This technology can be used to extend the functionality of a device cluster resource group (CRG) beyond the limits of physical component connection. Geographic mirroring provides the ability to replicate changes made to the production copy of an independent disk pool to a mirror copy of that independent disk pool. As data is written to the production copy of an independent disk pool, the operating system mirrors that data to a second copy of the independent disk pool through another system. This process keeps multiple identical copies of the data.

Through the device CRG, should a failover or switchover occur, the backup node can seamlessly take on the role of the primary node. The server or servers that act as backups are defined in the recovery domain. The backup nodes could be at the same or different physical location as the primary. When an outage occurs on the server defined as the primary node in the recovery domain and a switchover or failover is initiated, the node designated as the backup in the recovery domain becomes the primary access point for the resource and will then own the production copy of the independent disk pool. Thus, you can gain protection from the single point of failure associated with switchable resources.

Use the following table to help you understand the benefits and advantages of replication, switched disk, and cross-site mirroring technology.

Factor	Replication	Switched disk	Cross-site mirroring
Flexibility	10s of servers	2 servers	4 servers
Single point of failure	None	Disk subsystem	None
Cost	Additional disk capacity required. Replication software.	Switchable I/O expansion unit (tower) Option 41	Additional disk for mirror copy of independent disk pool Optionally switchable I/O expansion unit Option 41
Performance	Replication overhead	Little impact	Geographic mirroring overhead
Real time coverage	Journaled objects	Objects contained in independent disk pool	Objects contained in independent disk pool

Geographic dispersion	Limited by performance considerations	Limited attach distance as servers and expansion units must be attached to HSL OptiConnect loop (250 meters maximum)	Limited by performance considerations (No limits are imposed by the system. However, response time and throughput over selected communications lines can dictate some practical limit.)
Disaster recovery protection	Yes	No	Yes
Concurrent backup	Yes	No	No
Setup	Replication environment. Determining what to replicate.	Independent disk pool environment. Populate independent disk pool	Independent disk pool environment (include setting up geographic mirroring) Populate independent disk pool

Plan for replication: **Replication** makes a copy of something in real time. It is the process of copying objects from one node in a cluster to one or more other nodes in the cluster. Replication makes and keeps the objects on your systems identical. If you make a change to an object on one node in a cluster, this change is replicated to other nodes in the cluster.

You must decide on a software technology to use for replication. The following solutions are available for achieving replication in your cluster:

- **"Cluster middleware business partners and available clustering products" on page 25**
  Data replication software from recognized cluster business partners enables you to replicate objects across multiple nodes.
- **A custom-written replication application**
  IBM(R) journal management provides a means by which you can record the activity of objects on your system. You can write an application taking advantage of journal management to achieve replication. See iSeries(TM) journal management for details on how journal management works.

Once you have chosen a mechanism to achieve replication, you must also:

- "Determine which systems to use for replication"

Determine which systems to use for replication: Key considerations for determining which systems to use for replication are:

- Performance capacity
- Disk capacity
- Critical data
- Disaster prevention

If your system fails over, you need to know what data and applications you have running on your primary system and your backup system. You want to put the critical data on the system that is most capable of handling the workload in case it fails over. You do not want to run out of disk space. If your primary system runs out of space and fails over, it is highly likely that your backup system is also going to fail over due to lack of disk space. To ensure your data center is not completely destroyed in case of a natural disaster, such as a flood, tornado, or hurricane, you should locate the replicated system in a remote location.

Plan for switchable independent disk pools and geographic mirroring: Careful planning is required if you plan to take advantage of switchable resources residing on switchable independent disk pools or

geographic mirroring. The requirements for implementing independent disk pools and geogaphic mirroring are detailed in the Plan for independent disk pools topic.

Cluster security

This topic discusses some of the security issues you need to consider when you plan to implement clustering on your systems.

- "Enable a node to be added to a cluster"
- "Distribute cluster-wide information"
- "Maintain user profiles on all nodes" on page 35

Enable a node to be added to a cluster

Before you can add a node to a cluster, you need to set a value for the Allow add to cluster (ALWADDCLU) network attribute. Use the Change Network Attributes (CHGNETA) command on any server that you want to set up as a cluster node. The Change Network Attributes (CHGNETA) command changes the network attributes of a system. The ALWADDCLU network attribute specifies whether a node will allow another system to add it as a node in a cluster.

Note: You must have *IOSYSCFG authority to change the network attribute ALWADDCLU.

You can pick one of these values:

***SAME**
The value does not change. The system is shipped with a value of *NONE.

***NONE**
No other system can add this system as a node in a cluster.

***ANY** Any other system can add this system as a node in a cluster.

***RQSAUT**
Any other system can add this system as a node in a cluster only after the cluster add request has been authenticated.

The ALWADDCLU network attribute is checked to see if the node that is being added is allowed to be part of the cluster and whether to validate the cluster request through the use of X.509 digital certificates. A **digital certificate** is a form of personal identification that can be verified electronically. If validation is required, the requesting node and the node that is being added must have the following installed on the systems:

- OS/400(R) Option 34 (Digital Certificate Manager)
- Cryptographic Access Provider licensed program (5722-AC2 or 5722-AC3)

When *RQSAUT is selected, the certificate authority trust list for the OS/400 cluster security server application must be properly set up. The server application identifier is QIBM_QCST_CLUSTER_SECURITY. At a minimum, add certificate authorities for those nodes that you allow to join the cluster.

See Digital Certificate Management for more information.

Distribute cluster-wide information

The Distribute Information (QcstDistributeInformation) API can be used to send messages from one node in a cluster resource group recovery domain to other nodes in that recovery domain. This can be useful in exit program processing. However, it should be noted that there is no encrypting of that information. Secure information should not be sent using this mechanism unless you are using a secure network.

Non-persistent data can be shared and replicated between cluster nodes using the Clustered Hash Table APIs. The data is stored in non-persistent storage. This means the data is retained only until the cluster

node is no longer part of the clustered hash table. These APIs can only be used from a cluster node that is defined in the clustered hash table domain. The cluster node must be active in the cluster.

Other information distributed via cluster messaging is similarly not secured. This includes the low level cluster messaging. As such, when changes are made to the exit program data, there is no encrypting of the message containing that data.

Maintain user profiles on all nodes

Because there is no central security administration to update nodes, user profiles are not automatically updated across cluster nodes. Be sure you update the security information across all nodes to ensure that any public or private authorities associated with any cluster objects, cluster resource groups, applications, or data have the right security level.

One mechanism to achieve this is to use Management Central in iSeries(TM) Navigator to perform administrator or operator functions across multiple systems and groups of systems. This support includes some common user-administration tasks that operators need to perform across the multiple systems in their cluster. Management Central allows user profile functions to be performed against groups of systems. The administrator can specify a post-propagation command to be run on the target systems when creating a user profile.

See Manage users and groups with Management Central for details.

Cluster configuration checklist

Before you begin to configure your cluster, complete this checklist to ensure that your environment is prepared properly.

TCP/IP requirements	
__	Start TCP/IP on every node you plan to include in the cluster using the Start TCP/IP (STRTCP) Command.
__	Configure the TCP loopback address (127.0.0.1) and verify that it shows a status of *Active*. Verify using the Work with TCP/IP Network Status (WRKTCPSTS) Command on every node in the cluster.
__	Verify that the IP addresses used for clustering to a given node must show a status of *Active* using the Work with TCP/IP Network Status (WRKTCPSTS) Command on the subject node.
__	Verify that INETD is active on all nodes in the cluster (STRTCPSVR *INETD). This can be verified by the presence of a QTOGINTD (User QTCP) job in the Active Jobs list on the subject node. See INETD server for details on starting the INETD server.
__	Verify that the local and any remote nodes are reachable (PING) using the IP addresses used for clustering to ensure network routing is active.
__	Verify that ports 5550 and 5551 are not being used by other applications. These ports are reserved for IBM(R) clustering. Port usage can be viewed using the Work with TCP/IP Network Status (WRKTCPSTS) command. Port 5550 will be opened and in a 'Listen' state by clustering once INETD is started.

If you plan to implement switchable devices in your cluster, the following requirements must be satisfied:

Resilient device requirements	
__	Verify that Option 41 (OS/400 - HA Switchable Resources) is installed and a valid license key exists on all cluster nodes that will be in the device domain. Note that any use of the "iSeries Navigator cluster management" on page 24 interface requires this option.
__	In order to access disk management functions in iSeries(TM) Navigator, configure the service tools server (STS) with DST access and user profiles. See Set up communication for details.

Resilient device requirements	
__	If you are switching resilient devices between logical partitions on a system, and you are using something other than the HMC to manage your logical partitions, enable Virtual OptiConnect for the partitions. This is done at dedicated service tools (DST) signon. See Virtual OptiConnect for details. If you are using the Hardware Management Console to manage your partitions, change your partition profile properties on the OptiConnect tab to enable Virtual OptiConnect for each partition in the switchable configuration. You must activate the partition profile to reflect the change.
__	If a tower on an HSL OptiConnect loop is switched between two systems, and one of the systems has logical partitions, enable HSL OptiConnect for the partitions. If you are using something other than the HMC to manage your logical partitions, this is done at dedicated service tools (DST) signon. If you are using the Hardware Management Console to manage your partitions, change your partition profile properties on the OptiConnect tab to enable HSL OptiConnect for each partition in the switchable configuration. You must activate the partition profile to reflect the change.
__	If you are switching resilient devices between logical partitions, and you are using something other than the HMC to manage your logical partitions, you must configure the bus to be shared between the partitions or configure an I/O Pool. The bus must be configured as "own bus shared" by one partition, and all other partitions that will participate in the device switching must be configured as "use bus shared." If you are using the Hardware Management Console to manage your logical partitions, you must configure an I/O pool that includes the I/O processor, I/O adapter, and all attached resources to allow an independent disk pool to be switchable between partitions. Each partition must have access to the I/O pool. See Make your hardware switchable for more details.
__	When switching a tower on a HSL loop between two different systems, the tower must be configured as switchable. See Make your hardware switchable for details.
__	When a tower is added to an existing HSL loop, restart all servers on that same loop.
__	The maximum transmission unit (MTU) for your communication paths must be greater than the cluster communications tuneable parameter, message fragment size. MTU for a cluster IP address can be verified using the Work with TCP/IP Network Status (WRKTCPSTS) command on the subject node. The MTU must also be verified at each step along the entire communications path. It may be easier to lower the message fragment size parameter once the cluster is created than raise the MTU for the communications path. See Tunable cluster communications parameters for more information on message fragment size. You can use the Retrieve Cluster Resource Services Information (QcstRetrieveCRSInfo) API to view the current settings of the tuning parameters and the Change Cluster Resource Services (QcstChgClusterResourceServices) API to change the settings.

Security requirements	
__	The ALWADDCLU (Allow Add to Cluster) network attribute must be appropriately set on the target node if trying to start a remote node. This should be set to *ANY or *RQSAUT depending on your environment. If set to *RQSAUT, then OS/400 option 34 (Digital Certificate Manager) and Cryptographic Access Provided Product (AC2 or AC3) must be installed. See “Enable a node to be added to a cluster” on page 34 for details on setting the ALWADDCLU network attribute.
__	Enable the status of the QUSER user profile. It must not have *SECADM or *ALLOBJ special authorities.

Security requirements	
__	Verify that the user profile invoking the cluster resource services APIs exists on all cluster nodes and has *IOSYSCFG authority.
__	Verifty that the user profile to run the exit program for a cluster resource group (CRG) exists on all recovery domain nodes.

Job considerations	
__	Jobs can be submitted by the cluster resource services APIs to process requests. The jobs will either run under the user profile to run the exit program specified when creating the a cluster resource group, or under the user profile that requested the API (for varying on devices in resilient device CRGs only). The user must ensure that the subsystem which services the job queue associated with the user profile is configured as: *NOMAX for the number of jobs it can run from that job queue.
__	Jobs will be submitted to the job queue specified by the job description which is obtained from the user profile defined for a CRG. The default job description will cause the jobs to be sent to the QBATCH job queue. Since this job queue is used for many user jobs, the exit program job may not run in a timely fashion. Users should consider a unique job description with a unique user queue.
__	When exit program jobs are run, they will use routing data from the job description to choose which main storage pool and run time attributes they will use. The default values will result in jobs that are run in a pool with other batch jobs with a run priority of 50. Neither of these may produce the desired performance for exit program jobs. The subsystem initiating the exit program jobs (the same subsystem that is using the unique job queue) should assign the exit program jobs to a pool that is not used by other jobs initiated by the same subsystem or other subsystems. In addition, the exit program jobs should be assigned a run priority of 15 so that they will run before almost all other user jobs.

There are several software solutions available for configuring and managing your cluster. One of these solutions is “iSeries Navigator cluster management” on page 24. If you choose to use iSeries Navigator, the following requirements must be satisfied:

iSeries Navigator cluster management considerations	
__	Option 41 (OS/400 - HA Switchable Resources) must be installed and a valid license key must exist on all cluster nodes that will be in the device domain.
__	Verifty that all host servers are started using the STRHOSTSVR (Start Host Server) Command: STRHOSTSVR SERVER(*ALL)
__	Verify that the Management Central server is started using the STRTCPSVR (Start TCP/IP Server) Command: STRTCPSVR SERVER(*MGTC)

Cluster applications

A key element of a clustered environment is application resilience. By taking advantage of “Resilient applications” on page 14 in your cluster, an application can be restarted on a different cluster node without requiring you to reconfigure the clients. In addition, the data that is associated with the application will be available after switchover or failover. This means that the application end user can experience minimal, or even seamless, interruption while the application and its data switch from the primary node to the backup node. The user does not need to know that the application and data have moved on the back end.

In order to achieve application resiliency in your cluster, applications that meet certain availability specifications must be used. Certain characteristics must be present in the application in order for it to be switchable, and therefore always available to the end users of the application in the cluster. Because these requirements exist, you have the following options for implementing a switchable software product in your cluster:

1. **Purchase a cluster-enabled software application**
   Software products that are cluster-enabled meet certain high-availability requirements. See "OS/400 architecture for cluster-enabled applications" for more details.
2. **Write or modify your own application to make it highly available**
   Independent software vendors and application programmers can customize applications to allow them to be switchable in an iSeries(TM) clustered environment. See "Writing a highly available cluster application" for details.

Once you have a resilient application, it must be managed within your cluster. See "Application CRG considerations" on page 41 for more information.

OS/400 architecture for cluster-enabled applications

Additional end-user value is provided by any application that is highly available, recognizing applications that continue to be available in the event of an outage, planned or unplanned. OS/400(R) has provided an application resilience architecture that supports various degrees of highly available application. At the high end of this spectrum applications will be enhanced with intergrated functions that demonstrate highly available characteristics and automation of the highly available environment, controlled by cluster management utilities.

These applications have the following characteristics:

- The application can switch over to a backup cluster node when the primary node becomes unavailable.
- The application defines the resilient environment in the Resilient Definition and Status Data Area to enable automatic configuration and activation of the application by a cluster management application.
- The application provides application resilience by means of an application CRG exit program to handle cluster related events, taking advantage of the capabilities of the OS/400 cluster resource services.
- The application provides an application restart function that repositions the user to an application menu screen or beyond.

Applications that demonstrate more stringent availability and restart characteristics have the following characteristics:

- The application provides enhanced application resilience through more robust handling of cluster events (action codes) by the application CRG exit program.
- The application provides a greater level of application restart support. For host-centric applications, the user will be repositioned to a transaction boundary by commitment control or checkpoint functions. For client-centric applications, the user will experience a seamless failover with minimal service interruption.

For more information on this application resilience architecture, see the iSeries(TM) High Availability and Clusters

web site.

Writing a highly available cluster application

A highly available application is one that can be resilient to a system outage in a clustered environment. Several levels of application availability are possible:

1. If an application error occurs, the application restarts itself on the same node and corrects any potential cause for error (such as corrupt control data). You would view the application as though it had started for the first time.
2. The application performs some amount of checkpoint-restart processing. You would view the application as if it were close to the point of failure.

3. If a system outage occurs, the application is restarted on a backup server. You would view the application as though it had started for the first time.
4. If a system outage occurs, the application is restarted on a backup server and performs some amount of checkpoint-restart processing across the servers. You would view the application as if it were close to the point of failure.
5. If a system outage occurs, a coordinated failover of both the application and its associated data to another node or nodes in the cluster would take place. You would view the application as though it had started for the first time.
6. If a system outage occurs, a coordinated failover of both the application and its associated data to another node or nodes in the cluster would take place. The application performs some amount of checkpoint-restart processing across the servers. You would view the application as if it were close to the point of failure.

Note: In cases 1 through 4 above, you are responsible for recovering the data.

For further considerations for application resiliency, see the following topics:

- "Make application programs resilient"
- "Restart highly available cluster applications" on page 40
- "Calling a cluster resource group exit program" on page 40

Make application programs resilient

A resilient application is expected to have the following characteristics:

- The application can be restarted on this node or another node
- The application is accessible to the client through the IP address
- The application is stateless or state information is known
- Data that is associated with the application is available after switchover

The three essential elements that make an application resilient to system outages in a clustered environment are:

The application itself

How tolerant is the application to errors or to system outages, and how transparently can the application restart itself?
The application can handle this through the use of new clustering capabilities.

Associated data

When an outage occurs, does it affect the availability of any associated data?
A "Cluster middleware business partners and available clustering products" on page 25 replication product which takes advantage of the clustering capabilities can handle this. Alternatively, the data can be stored in a switchable independent disk pool (switchable independent ASP).

Control capabilities and administration

How easy is it to define the environment that supports the availability of the data and the application?
A Cluster Middleware business partner cluster management product that uses the clustering APIs and also combines resilient applications with resilient data can handle this.

Restart highly available cluster applications

To restart an application, the application needs to know its state at the time of the failover or switchover. State information is application specific; therefore, the application must determine what information is needed. Without any state information, the application can be restarted on your PC. However, you will have to reestablish your position within the application.

Several methods are available to save application state information for the backup system. Each application needs to determine which method works best for it.

- The application can transfer all state information to the requesting client system. When a switchover or failover occurs, the application uses the stored state on the client to reestablish the state in the new server. This could be accomplished by using the Distribute Information API or Clustered Hash Table APIs. See "Distribute cluster-wide information" on page 34 for details.
- The application can replicate state information (such as job information and other control structures that are associated with the application) on a real-time basis. For every change in the structures, the application ships the change over to the backup system.
- The application can store pertinent state information that is associated with its application in the exit program data portion of the cluster resource group for that application. This method assumes that a small amount of state information is required. You can use the Change Cluster Resource Group (QcstChangeClusterResourceGroup) API to do this.
- The application can store state information in a data object that is being replicated to the backup systems along with the application's data.
- The application can store state information in a data object contained in the switchable IASP that also contains the application's data.
- The application can store the state information on the client.
- No state information is saved, and you need to perform the recovery.

Note: The amount of information that is required to be saved is lessened if the application uses some form of checkpoint-restart processing. State information is only saved at predetermined application checkpoints. Restart then takes you back to the last known checkpoint which is similar to how database's commitment control processing works.

Calling a cluster resource group exit program

The cluster resource group exit program is called during different phases of a cluster environment. This program establishes and manages the environment necessary for data, application, or device resiliency within a cluster. The exit program is optional for a resilient device CRG but is required for the other CRG types. When a cluster resource group exit program is used, it is called on the occurence of cluster-wide events, including when:

- A node leaves the cluster unexpectedly.
- A node leaves the cluster as a result of the End Cluster Node (QcstEndClusterNode) API or Remove Cluster Node Entry (QcstRemoveClusterNodeEntry) API.
- The cluster is deleted as a result of the Delete Cluster (QcstDeleteCluster) API.
- A node is activated by the Start Cluster Node (QcstStartClusterNode) API.
- Communication with a partitioned node is re-established.

This exit program:

- Runs in a named activation group or the caller's activation group (*CALLER).
- Ignores the restart parameter if the exit program has an unhandled exception or is cancelled.
- Provides a cancel handler.

When a cluster resource group API is run, the exit program is called from a separate job with the user profile specified on the Create Cluster Resource Group (QcstCreateClusterResourceGroup) API. The separate job is automatically created by the API when the exit program is called. If the exit program for a

data CRG is unsuccessful or ends abnormally, the cluster resource group exit program is called on all active nodes in the recovery domain with an action code of Undo. This action code allows any unfinished activity to be backed out and the original state of the cluster resource group to be recovered.

If the exit program for an application CRG is unsuccessful or ends abnormally, cluster resource services will attempt to restart the application if the status of the CRG is active. The cluster resource group exit program is called with an action code of Restart. If the application cannot be restarted in the specified maximum number of attempts, the cluster resource group exit program is called with an action code of Failover. The restart count is reset only when the exit program is called with an action code of start, which can be the result of a start CRG, a failover, or a switchover.

When the cluster resource group is started, the application CRG exit program called on the primary node is not to return control to cluster resource services until the application itself ends or an error occurs. After an application CRG is active, if cluster resource services must notify the application CRG exit program of some event, another instance of the exit program is started in a different job. Any other action code other than Start or Restart is expected to return.

When a cluster resource group exit program is called, it is passed a set of parameters that identify the cluster event being processed, the current state of the cluster resources, and the expected state of the cluster resources.

For complete information on cluster resource group exit programs, including what information is passed to the exit program for each action code, see Cluster Resource Group Exit Program in the Cluster API documentation. Sample source code has been provided in the QUSRTOOL library which can be used as a basis for writing an exit program. See the TCSTAPPEXT member in the QATTSYSC file.

Application CRG considerations

An application cluster resource group manages application resiliency. Consider the following topics when using resilient applications in your cluster.

"Manage application CRG IP addresses"
Cluster resource services will manage CRG IP addresses for you. You can also manage them manually.

"Example: Application cluster resource group failover actions" on page 42
See how an example failover scenario works.

"Example: Application exit program" on page 42
Use this code example contains code for a sample application cluster resource group exit program

Note: Read the "Code disclaimer information" on page 110 for important legal information.

Manage application CRG IP addresses

There are two ways to have the application takeover IP address associated with an application CRG managed. The easiest way, which is the default, is to let cluster resource services manage the IP address. This method will direct cluster resource services to create the IP address on all nodes in the recovery domain, including nodes subsequently added to the recovery domain. When this method is selected, the IP address cannot currently be defined on any node in the recovery domain.

The alternative way is to manage the IP addresses yourself. This method directs cluster resource services to not take any steps to configure the IP address; the user is responsible for configuration. You must add the takeover IP address on all nodes in the recovery domain (except on replicate nodes) prior to starting the cluster resource group. Any node to be added to the recovery domain of an active CRG must have the IP address configured prior to being added.

Multiple subnets

It is possible to have the application takeover IP address work across multiple subnets, although the default is to have all recovery domain nodes on the same subnet. See Enabling application switchover across subnets for the steps to configure the application takeover IP address when the nodes in the recovery domain span subnets.

Example: Application cluster resource group failover actions

The following happens when a cluster resource group for a resilient application fails over due to exceeding the retry limit or if the job is cancelled:

- The "Cluster resource group exit programs" on page 8 is called on all active nodes in the recovery domain for the CRG with an action code of failover. This indicates that cluster resource services is preparing to failover the application's point of access to the first backup.
- Cluster resource services ends the takeover Internet Protocol (IP) connection on the primary node. For more information on the takeover IP address, see "Manage application CRG IP addresses" on page 41.
- Cluster resource services starts the takeover IP address on the first backup (new primary) node.
- Cluster resource services submits a job that calls the cluster resource group exit program only on the new primary node with an action code of Start. This action restarts the application.

The above example shows how one failover scenario works. Other failover scenarios can work differently.

Example: Application exit program

The following code example contains code for a sample application cluster resource group exit program. You can find this code example in the QUSRTOOL library.

For information on the use of code examples, see the "Code disclaimer information" on page 110.

```
/*****************************************************************************/
/*                                                                           */
/* Library:  QUSRTOOL                                                        */
/* File:     QATTSYSC                                                        */
/* Member:   TCSTAPPEXT                                                      */
/* Type:     ILE C                                                           */
/*                                                                           */
/* Description:                                                              */
/* This is an example application CRG exit program which gets called for    */
/* various cluster events or cluster APIs.  The bulk of the logic must      */
/* still be added because that logic is really dependent upon the unique    */
/* things that need to be done for a particular application.                */
/*                                                                           */
/* The intent of this example to to provide a shell which contains the      */
/* basics for building a CRG exit program.  Comments throughout the example*/
/* highlight the kinds of issues that need to be addressed by the real      */
/* exit program implementation.                                              */
/*                                                                           */
/* Every action code that applies to an application CRG is handled in this */
/* example.                                                                  */
/*                                                                           */
/* The tcstdtaara.h include is also shipped in the QUSRTOOL library.  See   */
/* the TCSTDTAARA member in the QATTSYSC file.                               */
/*                                                                           */
/* Change log:                                                               */
/* Flag Reason   Ver    Date   User Id   Description                         */
/* ____ _______ ______ ______ ________ ____________________________________*/
/* ...  D98332   v5r1m0 000509 ROCH     Initial creation.                    */
/* $A1  P9950070 v5r2m0 010710 ROCH     Dataarea fixes                       */
/* $A2  D99055   v5r2m0 010913 ROCH     Added CancelFailover action code     */
/* $A3  D98854   v5r2m0 010913 ROCH     Added VerificationPhase action code*/
/* $A4  P9A10488 v5r3m0 020524 ROCH     Added example code to wait for data*/
/*                                      CRGs on switchover action code       */
/*                                                                           */
/*****************************************************************************/
```

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Header files                                                       */
/*                                                                    */
/*--------------------------------------------------------------------*/
#include             /* Useful when debugging                   */
#include            /* offsetof macro                          */
#include            /* system function                         */
#include            /* String functions                        */
#include            /* Exception handling constants/structures  */
#include              /* Various cluster constants               */
#include          /* Structure of CRG information             */
#include "qusrtool/qattsysc/tcstdtaara" /* QCSTHAAPPI/QCSTHAAPPO data areas*/
#include          /* API to Retrieve contents of a data area  */
#include             /* API error code type definition          */
#include             /* mitime builtin                          */
#include          /* waittime builtin                        */


/*--------------------------------------------------------------------*/
/*                                                                    */
/* Constants                                                          */
/*                                                                    */
/*--------------------------------------------------------------------*/
#define UnknownRole -999
#define DependCrgDataArea "QCSTHAAPPO"
#define ApplCrgDataArea "QCSTHAAPPI"
#define Nulls 0x0000000000000000000


/*--------------------------------------------------------------------*/
/*                                                                    */
/* The following constants are used in the checkDependCrgDataArea()   */
/* function.  The first defines how long to sleep before checking the data */
/* area.  The second defines that maximum time to wait for the data area   */
/* to become ready before failing to start the application when the Start  */
/* CRG function is being run.  The third defines the maximum wait time for */
/* the Initiate Switchover or failover functions.                     */
/*                                                                    */
/*--------------------------------------------------------------------*/
#define WaitSecondsIncrement 30
#define MaxStartCrgWaitSeconds 0
#define MaxWaitSeconds 900

/*--------------------------------------------------------------------*/
/*                                                                    */
/* As this exit program is updated to handle new action codes, change the  */
/* define below to the value of the highest numbered action code that is   */
/* handled.                                                           */
/*                                                                    */
/*--------------------------------------------------------------------*/
#define MaxAc 21

/*--------------------------------------------------------------------*/
/*                                                                    */
/* If the exit program data in the CRG has a particular structure to it,   */
/* include the header file for that structure definition and change the    */
/* define below to use that structure name rather than char.          */
/*                                                                    */
/*--------------------------------------------------------------------*/
#define EpData char

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Change the following define to the library the application resides in   */
/* and thus where the QCSTHAAPPO and QCSTHAAPPI data areas will be found.  */
```

```
/*                                                                            */
/*----------------------------------------------------------------------------*/
#define ApplLib "QGPL"


/*----------------------------------------------------------------------------*/
/*                                                                            */
/* Prototypes for internal functions.                                         */
/*                                                                            */
/*----------------------------------------------------------------------------*/
static int getMyRole(Qcst_EXTP0100_t *, int, int);
#pragma argopt(getMyRole)
static int doAction(int, int, int, Qcst_EXTP0100_t *, EpData *);
#pragma argopt(doAction)
static int createCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int startCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int restartCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int endCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int verifyPhase(int, int, Qcst_EXTP0100_t *, EpData *);
static int deleteCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int memberIsJoining(int, int, Qcst_EXTP0100_t *, EpData *);
static int memberIsLeaving(int, int, Qcst_EXTP0100_t *, EpData *);
static int switchPrimary(int, int, Qcst_EXTP0100_t *, EpData *);
static int addNode(int, int, Qcst_EXTP0100_t *, EpData *);
static int rmvNode(int, int, Qcst_EXTP0100_t *, EpData *);
static int chgCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int deleteCrgWithCmd(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoPriorAction(int, int, Qcst_EXTP0100_t *, EpData *);
static int endNode(int, int, Qcst_EXTP0100_t *, EpData *);
static int chgNodeStatus(int, int, Qcst_EXTP0100_t *, EpData *);
static int cancelFailover(int, int, Qcst_EXTP0100_t *, EpData *);
static int newActionCode(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoCreateCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoStartCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoEndCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoMemberIsJoining(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoMemberIsLeaving(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoSwitchPrimary(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoAddNode(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoRmvNode(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoChgCrg(int, int, Qcst_EXTP0100_t *, EpData *);
static int undoCancelFailover(int, int, Qcst_EXTP0100_t *, EpData *);
static void bldDataAreaName(char *, char *, char *);
#pragma argopt(bldDataAreaName)
static int checkDependCrgDataArea(unsigned int);
#pragma argopt(checkDependCrgDataArea)
static void setApplCrgDataArea(char);
#pragma argopt(setApplCrgDataArea)
static void cancelHandler(_CNL_Hndlr_Parms_T *);
static void unexpectedExceptionHandler(_INTRPT_Hndlr_Parms_T *);
static void endApplication(unsigned int, int, int, Qcst_EXTP0100_t *, EpData *);
#pragma argopt(endApplication)

/*----------------------------------------------------------------------------*/
/*                                                                            */
/* Some debug routines                                                        */
/*                                                                            */
/*----------------------------------------------------------------------------*/
static void printParms(int, int, int, Qcst_EXTP0100_t *, EpData *);
static void printActionCode(unsigned int);
static void printCrgStatus(int);
static void printRcvyDomain(char *,
                            unsigned int,
                            Qcst_Rcvy_Domain_Array1_t *);
static void printStr(char *, char *, unsigned int);
```

```
/*--------------------------------------------------------------------------*/
/*                                                                          */
/* Type definitions                                                         */
/*                                                                          */
/*--------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------*/
/*                                                                          */
/* This structure defines data that will be passed to the exception and     */
/* cancel handlers.  Extend it with information unique to your application.*/
/*                                                                          */
/*--------------------------------------------------------------------------*/
typedef struct {
  int *retCode;             /* Pointer to return code                     */
  EpData *epData;           /* Exit program data from the CRG             */
  Qcst_EXTP0100_t *crgData; /* CRG data                                   */
  unsigned int actionCode;  /* The action code                            */
  int role;                 /* This node's recovery domain role           */
  int priorRole;            /* This node's prior recovery domain role     */
} volatile HandlerDataT;


/*--------------------------------------------------------------------------*/
/*                                                                          */
/* Function pointer array for handling action codes.  When the exit program*/
/* is updated to handle new action codes, add the new function names to     */
/* this function pointer array.                                             */
/*                                                                          */
/*--------------------------------------------------------------------------*/
static int (*fcn[MaxAc+1]) (int role,
                            int priorRole,
                            Qcst_EXTP0100_t *crgData,
                            EpData *epData) = {
  newActionCode,   /* 0 - currently reserved */
  createCrg,       /* 1  */
  startCrg,        /* 2  */
  restartCrg,      /* 3  */
  endCrg,          /* 4  */
  verifyPhase,     /* 5 - currently reserved */
  newActionCode,   /* 6 - currently reserved */
  deleteCrg,       /* 7  */
  memberIsJoining, /* 8  */
  memberIsLeaving, /* 9  */
  switchPrimary,   /* 10 */
  addNode,         /* 11 */
  rmvNode,         /* 12 */
  chgCrg,          /* 13 */
  deleteCrgWithCmd,/* 14 */
  undoPriorAction, /* 15 */
  endNode,         /* 16 */
  newActionCode,   /* 17 - applies only to a device CRG */
  newActionCode,   /* 18 - applies only to a device CRG */
  newActionCode,   /* 19 - applies only to a device CRG */
  chgNodeStatus,   /* 20 */
  cancelFailover   /* 21 */
};


/*--------------------------------------------------------------------------*/
/*                                                                          */
/* Function pointer array for handling prior action codes when called with */
/* the Undo action code.  When the exit program is updated to handle       */
/* Undo for new action codes, add the new function names to this function  */
/* pointer array.                                                           */
/*                                                                          */
/*--------------------------------------------------------------------------*/
static int (*undoFcn[MaxAc+1]) (int role,
```

```
                              int priorRole,
                              Qcst_EXTP0100_t *crgData,
                              EpData *epData) = {
  newActionCode,       /* 0 - currently reserved */
  undoCreateCrg,       /* 1  */
  undoStartCrg,        /* 2  */
  newActionCode,       /* 3  */
  undoEndCrg,          /* 4  */
  newActionCode,       /* 5 - no undo for this action code */
  newActionCode,       /* 6 - currently reserved */
  newActionCode,       /* 7  */
  undoMemberIsJoining, /* 8  */
  undoMemberIsLeaving, /* 9  */
  undoSwitchPrimary,   /* 10 */
  undoAddNode,         /* 11 */
  undoRmvNode,         /* 12 */
  undoChgCrg,          /* 13 */
  newActionCode,       /* 14 */
  newActionCode,       /* 15 */
  newActionCode,       /* 16 */
  newActionCode,       /* 17 - applies only to a device CRG */
  newActionCode,       /* 18 - applies only to a device CRG */
  newActionCode,       /* 19 - applies only to a device CRG */
  newActionCode,       /* 20 */
  undoCancelFailover   /* 21 */
};


/*****************************************************************************/
/*                                                                           */
/* This is the entry point for the exit program.                             */
/*                                                                           */
/*****************************************************************************/
void main(int argc, char *argv[]) {

  HandlerDataT hdlData;

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* Take each of the arguments passed in the argv array and cast it to      */
  /* the correct data type.                                                  */
  /*                                                                         */
  /*-------------------------------------------------------------------------*/
  int *retCode    = (int *)argv[1];
  unsigned int *actionCode = (unsigned int *)argv[2];
  EpData *epData           = (EpData *)argv[3];
  Qcst_EXTP0100_t *crgData = (Qcst_EXTP0100_t *)argv[4];
  char *formatName         = (char *)argv[5];

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* Ensure the format of the data being passed is what we are expecting.    */
  /* If not, a change has been made and this exit program needs to be        */
  /* updated to accomodate the change.  Add appropriate error logging for    */
  /* your application design.                                                */
  /*                                                                         */
  /*-------------------------------------------------------------------------*/
  if (0 != memcmp(formatName, "EXTP0100", 8))
    abort();

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* Set up the data that will be passed to the exception and cancel         */
  /* handlers.                                                               */
  /*                                                                         */
  /*-------------------------------------------------------------------------*/
```

```
  hdlData.retCode    = retCode;
  hdlData.epData     = epData;
  hdlData.crgData    = crgData;
  hdlData.actionCode = *actionCode;
  hdlData.role       = UnknownRole;
  hdlData.priorRole  = UnknownRole;
  _VBDY();  /* force changed variables to home storage location           */

  /*------------------------------------------------------------------------*/
  /*                                                                        */
  /* Enable an exception handler for any and all exceptions.                */
  /*                                                                        */
  /*------------------------------------------------------------------------*/
#pragma exception_handler(unexpectedExceptionHandler, hdlData, \
                          _C1_ALL, _C2_ALL, _CTLA_INVOKE )

  /*------------------------------------------------------------------------*/
  /*                                                                        */
  /* Enable a cancel handler to recover if this job is cancelled.           */
  /*                                                                        */
  /*------------------------------------------------------------------------*/
#pragma cancel_handler(cancelHandler, hdlData)

  /*------------------------------------------------------------------------*/
  /*                                                                        */
  /* Extract the role and prior role of the node this exit program is       */
  /* running on.  If the cluster API or event changes the recovery domain   */
  /* (node role or membership status), the new recovery domain's offset is */
  /* passed in Offset_Rcvy_Domain_Array and the offset of the recovery      */
  /* domain as it looked prior to the API or cluster event is passed in     */
  /* Offset_Prior_Rcvy_Domain_Array.  If the recovery domain isn't changed,*/
  /* only Offset_Rcvy_Domain_Array can be used to address the recovery      */
  /* domain.                                                                */
  /*                                                                        */
  /*------------------------------------------------------------------------*/
  hdlData.role = getMyRole(crgData,
                           crgData->Offset_Rcvy_Domain_Array,
                           crgData->Number_Nodes_Rcvy_Domain);
  if (crgData->Offset_Prior_Rcvy_Domain_Array)
    hdlData.priorRole =
                 getMyRole(crgData,
                           crgData->Offset_Prior_Rcvy_Domain_Array,
                           crgData->Number_Nodes_Prior_Rcvy_Domain);
  else
    hdlData.priorRole = hdlData.role;
  _VBDY();  /* force changed variables to home storage location           */

  /*------------------------------------------------------------------------*/
  /*                                                                        */
  /* Enable the following to print out debug information.                   */
  /*                                                                        */
  /*------------------------------------------------------------------------*/
  /*
  printParms(*actionCode, hdlData.role, hdlData.priorRole, crgData, epData);
  */

  /*------------------------------------------------------------------------*/
  /*                                                                        */
  /* Do the correct thing based upon the action code.  The return code      */
  /* is set to the function result of doAction().                           */
  /*                                                                        */
  /*------------------------------------------------------------------------*/
  *retCode = doAction(*actionCode,
                      hdlData.role,
                      hdlData.priorRole,
                      crgData,
                      epData);
```

```
  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* The exit program job will end when control returns to the operating*/
  /* system at this point.                                              */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  return;

#pragma disable_handler  /* unexpectedExceptionHandler                  */
#pragma disable_handler  /* cancelHandler                               */
}  /* end main()                                                        */


/**********************************************************************/
/*                                                                    */
/* Get the role of this particular node from one of the views of the  */
/* recovery domain.                                                   */
/*                                                                    */
/* APIs and cluster events which pass the updated and prior recovery domain*/
/* to the exit program are:                                           */
/* QcstAddNodeToRcvyDomain                                            */
/* QcstChangeClusterNodeEntry                                         */
/* QcstChangeClusterResourceGroup                                     */
/* QcstEndClusterNode (ending node does not get the prior domain)     */
/* QcstInitiateSwitchOver                                             */
/* QcstRemoveClusterNodeEntry (removed node does not get the prior domain) */
/* QcstRemoveNodeFromRcvyDomain                                       */
/* QcstStartClusterResourceGroup (only if inactive backup nodes are   */
/*                                reordered)                          */
/* a failure causing failover                                         */
/* a node rejoining the cluster                                       */
/* cluster partitions merging                                         */
/*                                                                    */
/* All other APIs pass only the updated recovery domain.              */
/*                                                                    */
/**********************************************************************/
static int getMyRole(Qcst_EXTP0100_t *crgData, int offset, int count) {

  Qcst_Rcvy_Domain_Array1_t *nodeData;
  unsigned int iter = 0;

  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* Under some circumstances, the operating system may not be able to  */
  /* determine the ID of this node and passes *NONE.  An example of such a */
  /* circumstance is when cluster resource services is not active on a  */
  /* node and the DLTCRG CL command is used.                            */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  if (0 == memcmp(crgData->This_Nodes_ID, QcstNone, sizeof(Qcst_Node_Id_t)))
    return UnknownRole;

  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* Compute a pointer to the first element of the recovery domain array.  */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  nodeData = (Qcst_Rcvy_Domain_Array1_t *)((char *)crgData + offset);

  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* Find my node in the recovery domain array.  I will not be in the   */
  /* prior recovery domain if I am being added by the Add Node to Recovery */
  /* Domain API.                                                        */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
```

```
  while (  0 != memcmp(crgData->This_Nodes_ID,
                       nodeData->Node_ID,
                       sizeof(Qcst_Node_Id_t))
         &&
           iter &lt; count
        ) {
    nodeData++;
    iter++;
  }

  if (iter &lt; count)
    return nodeData->Node_Role;
  else
    return UnknownRole;
}  /* end getMyRole()                                                    */


/*****************************************************************************/
/*                                                                           */
/* Call the correct function based upon the cluster action code.  The        */
/* doAction() function was split out from main() in order to clarify the    */
/* example.  See the function prologues for each called function for         */
/* information about a particular cluster action.                            */
/*                                                                           */
/* Each action code is split out into a separate function only to help       */
/* clarify this example.  For a particular exit program, some action codes   */
/* may perform the same function in which case multiple action codes could   */
/* be handled by the same function.                                          */
/*                                                                           */
/*****************************************************************************/
static int doAction(int actionCode,
                    int role,
                    int priorRole,
                    Qcst_EXTP0100_t *crgData,
                    EpData *epData) {

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* For action codes this exit program knows about, call a function to      */
  /* do the work for that action code.                                       */
  /*                                                                         */
  /*-------------------------------------------------------------------------*/
  if (actionCode &lt;= MaxAc )
    return (*fcn[actionCode]) (role, priorRole, crgData, epData);
  else
    /*-----------------------------------------------------------------------*/
    /*                                                                       */
    /* IBM has defined a new action code in a new operating system release   */
    /* and this exit program has not yet been updated to handle it.  Take a  */
    /* default action for now.                                               */
    /*                                                                       */
    /*-----------------------------------------------------------------------*/
    return newActionCode(role, priorRole, crgData, epData);
}  /* end doAction()                                                     */


/*****************************************************************************/
/*                                                                           */
/* Action code = QcstCrgAcInitialize                                         */
/*                                                                           */
/* The QcstCreateClusterResourceGroup API was called.  A new cluster         */
/* resource group object is being created.                                   */
/*                                                                           */
/* Things to consider:                                                       */
/*   - Check that the application program and all associated objects are on  */
/*     the primary and backup nodes.  If the objects are not there,          */
/*     consider sending error/warning messages or return a failure return    */
```

```
/*     code.                                                              */
/*   - Check that required data or device CRGs are on all nodes in the    */
/*     recovery domain.                                                   */
/*   - Perform any necessary setup that would be required to run the      */
/*     the application on the primary or backup nodes.                    */
/*   - If this CRG is enabled to use the QcstDistributeInformation API,   */
/*     the user queue needed by that API could be created at this time.   */
/*                                                                        */
/**************************************************************************/
static int createCrg(int role,
                     int doesNotApply,
                     Qcst_EXTP0100_t *crgData,
                     EpData *epData) {

  return QcstSuccessful;
}  /* end createCrg()                                                     */


/**************************************************************************/
/*                                                                        */
/* Action code = QcstCrgAcStart                                           */
/*                                                                        */
/* The QcstStartClusterResourceGroup API was called.  A cluster resource  */
/* group is being started.                                                */
/* The QcstInitiateSwitchOver API was called and this is the second action */
/* code being passed to the exit program.                                 */
/* The fail over event occurred and this is the second action code being  */
/* passed to the exit program.                                            */
/*                                                                        */
/* A maximum wait time is used when checking to see if all dependent CRGs */
/* are active.  This is a short time if the CRG is being started because of*/
/* the QcstStartClusterResourceGroup API.  It is a longer time if it is   */
/* because of a failover or switchover.  When failover or switchover are  */
/* being done, it make take a while for data or device CRGs to become     */
/* ready so the wait time is long.  If the Start CRG API is being used, the*/
/* dependent CRGs should already be started or some error occurred, the   */
/* CRGs were started out of order, etc. and there is no need for a long   */
/* wait.                                                                  */
/*                                                                        */
/* Things to consider:                                                    */
/*   - If this node's role is primary, the application should be started. */
/*     This exit program should either call the application so that it runs*/
/*     in this same job or it should monitor any job started by this      */
/*     exit program so the exit program knows when the application job    */
/*     ends.  By far, the simplest approach is run the application in this */
/*     job by calling it.                                                 */
/*     Cluster Resource Services is not expecting this exit program to    */
/*     return until the application finishes running.                     */
/*   - If necessary, start any associated subsystems, server jobs, etc.   */
/*   - Ensure that required data CRGs have a status of active on all nodes */
/*     in the recovery domain.                                            */
/*                                                                        */
/**************************************************************************/
static int startCrg(int role,
                    int doesNotApply,
                    Qcst_EXTP0100_t *crgData,
                    EpData *epData) {

  unsigned int maxWaitTime;

  /* Start the application if this node is the primary                    */
  if (role == QcstPrimaryNodeRole) {
    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Determine if all CRGs that this application CRG is dependent upon  */
    /* are ready.  If the check fails, return from the Start action code.  */
    /* Cluster Resource Services will change the state of the CRG to      */
```

```
    /* Inactive.                                                          */
    /*                                                                    */
    /*--------------------------------------------------------------------*/
    if (crgData->Cluster_Resource_Group_Status == QcstCrgStartCrgPending)
      maxWaitTime = MaxStartCrgWaitSeconds;
    else
      maxWaitTime = MaxWaitSeconds;
    if (QcstSuccessful != checkDependCrgDataArea(maxWaitTime))
      return QcstSuccessful;


    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Just before starting the application, update the data area to      */
    /* indicate the application is running.                               */
    /*                                                                    */
    /*--------------------------------------------------------------------*/
    setApplCrgDataArea(Appl_Running);

    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Add logic to call application here.  It is expected that control   */
    /* will not return until something causes the application to end: a   */
    /* normal return from the exit program, the job is cancelled, or an   */
    /* unhandled exception occurs.  See the cancelHandler() function for  */
    /* some common ways this job could be cancelled.                      */
    /*                                                                    */
    /*--------------------------------------------------------------------*/



    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* After the application has ended normally, update the data area to  */
    /* indicate the application is no longer running.                     */
    /*                                                                    */
    /*--------------------------------------------------------------------*/
    setApplCrgDataArea(Appl_Ended);
  }
  else
    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* On backup or replicate nodes, mark the status of the application in */
    /* the data area as not running.                                      */
    /*                                                                    */
    /*--------------------------------------------------------------------*/
    setApplCrgDataArea(Appl_Ended);


  return QcstSuccessful;
}  /* end startCrg()                                                     */


/************************************************************************/
/*                                                                      */
/* Action code = QcstCrgAcRestart                                       */
/*                                                                      */
/* The previous invocation of the exit program failed and set the return */
/* code to QcstFailWithRestart or it failed due to an exception and the */
/* exception was allowed to percolate up the invocation stack.  In either */
/* case, the maximum number of times for restarting the exit program has */
/* not been reached yet.                                                */
/*                                                                      */
/* This action code is passed only to application CRG exit programs which */
/* had been called with the Start action code.                          */
/*                                                                      */
/************************************************************************/
```

```
static int restartCrg(int role,
                      int doesNotApply,
                      Qcst_EXTP0100_t *crgData,
                      EpData *epData) {

  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* Perform any unique logic that may be necessary when restarting the */
  /* application after a failure and then call the startCrg() function to */
  /* do the start functions.                                            */
  /*                                                                    */
  /*--------------------------------------------------------------------*/


  return startCrg(role, doesNotApply, crgData, epData);
}  /* end restartCrg()                                                 */


/**********************************************************************/
/*                                                                    */
/* Action code = QcstCrgAcEnd                                         */
/*                                                                    */
/* The end action code is used for one of the following reasons:      */
/*   - The QcstEndClusterResourceGroup API was called.                */
/*   - The cluster has become partitioned and this node is in the secondary*/
/*     partition.  The End action code is used regardless of whether the */
/*     CRG was active or inactive.  Action code dependent data of     */
/*     QcstPartitionFailure will also be passed.                      */
/*   - The application ended.   Action code dependent data of         */
/*     QcstResourceEnd will also be passed.  All nodes in the recovery */
/*     domain will see the same action code (including the primary).  */
/*   - The CRG job has been cancelled.  The exit program on this node will */
/*     be called with the End action code.  QcstMemberFailure will be */
/*     passed as action code dependent data.                          */
/*                                                                    */
/*                                                                    */
/*                                                                    */
/* Things to consider:                                                */
/*   - If the CRG is active, the job running the application is cancelled */
/*     and the IP takeover address is ended AFTER the exit program is */
/*     called.                                                        */
/*   - If subsystems or server jobs were started as a result of the   */
/*     QcstCrgAcStart action code, end them here or consolidate all logic */
/*     to end the application in the cancelHandler() since it will be */
/*     invoked for all Cluster Resource Services APIs which must end the */
/*     application on the current primary.                            */
/*                                                                    */
/**********************************************************************/
static int endCrg(int role,
                  int priorRole,
                  Qcst_EXTP0100_t *crgData,
                  EpData *epData) {

  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* End the application if it is running on this node.                 */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  endApplication(QcstCrgAcRemoveNode, role, priorRole, crgData, epData);

  return QcstSuccessful;
}  /* end endCrg()                                                    */


/**********************************************************************/
/*                                                                    */
/* Action code = QcstCrgAcVerificationPhase                           */
```

```
/*                                                                    */
/* The verification phase action code is used to allow the exit program to */
/* do some verification before proceeding with the requested function      */
/* identified by the action code depended data. If the exit program        */
/* determines that the requested function cannot proceed it should return  */
/* QcstFailWithOutRestart.                                                 */
/*                                                                         */
/*                                                                         */
/* NOTE: The exit program will NOT be called with Undo action code.        */
/*                                                                         */
/*************************************************************************/
static int verifyPhase(int role,
                       int doesNotApply,
                       Qcst_EXTP0100_t *crgData,
                       EpData *epData) {

  /*---------------------------------------------------------------------*/
  /*                                                                     */
  /* Do verification                                                     */
  /*                                                                     */
  /*---------------------------------------------------------------------*/
  if (crgData->Action_Code_Dependent_Data == QcstDltCrg) {
 /* do verification  */
        /* if ( fail ) */
          /* return QcstFailWithOutRestart */
  }

  return QcstSuccessful;
}  /* end verifyPhase()                                                */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcDelete                                         */
/*                                                                       */
/* The QcstDeleteClusterResourceGroup or QcstDeleteCluster API was called. */
/* A cluster resource group is being deleted while Cluster Resource      */
/* Services is active.                                                   */
/* If the QcstDeleteCluster API was used, action code dependent data of  */
/* QcstDltCluster is passed.                                             */
/* If the QcstDeleteCluster API was used and the CRG is active, the exit */
/* program job which is still active for the Start action code is cancelled*/
/* after the Delete action code is processed.                            */
/*                                                                       */
/* Things to consider:                                                   */
/*   - Delete application programs and objects from nodes where they are */
/*     no longer needed such as backup nodes.  Care needs to be exercised */
/*     when deleting application objects just because a CRG is being     */
/*     deleted since a particular scenario may want to leave the         */
/*     application objects on all nodes.                                 */
/*                                                                       */
/*************************************************************************/
static int deleteCrg(int role,
                     int doesNotApply,
                     Qcst_EXTP0100_t *crgData,
                     EpData *epData) {

  return QcstSuccessful;
}  /* end deleteCrg()                                                  */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcReJoin                                         */
/*                                                                       */
/* One of three things is occurring-                                     */
/*  1. The problem which caused the cluster to become partitioned has been */
```

```
/*     corrected and the 2 partitions are merging back together to become  */
/*     a single cluster.  Action code dependent data of QcstMerge will be  */
/*     passed.                                                             */
/*  2. A node which either previously failed or which was ended has had    */
/*     cluster resource services started again and the node is joining the */
/*     cluster.  Action code dependent data of QcstJoin will be passed.    */
/*  3. The CRG job on a particular node which may have been cancelled or   */
/*     ended has been restarted.  Action code dependent data of QcstJoin   */
/*     will be passed.                                                     */
/*                                                                         */
/* Things to consider:                                                     */
/*   - If the application replicates application state information to other*/
/*     nodes when the application is running, this state information will  */
/*     need to be resynchronized with the joining nodes if the CRG is      */
/*     active.                                                             */
/*   - Check for missing application objects on the joining nodes.         */
/*   - Ensure the required data CRGs are on the joining nodes.             */
/*   - If the application CRG is active, ensure the required data CRGs are */
/*     active.                                                             */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static int memberIsJoining(int role,
                           int priorRole,
                           Qcst_EXTP0100_t *crgData,
                           EpData *epData) {

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* Ensure the data area status on this node starts out indicating      */
  /* the application is not running if this node is not the primary.     */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/
  if (role != QcstPrimaryNodeRole) {
    setApplCrgDataArea(Appl_Ended);
  }

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* If a single node is rejoining the cluster, you may do a certain set of*/
  /* actions.  Whereas if the nodes in a cluster which became partitioned  */
  /* are merging back together, you may have a different set of actions.   */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/
  if (crgData->Action_Code_Dependent_Data == QcstJoin) {
    /* Do actions for a node joining.                                      */
  }
  else {
    /* Do actions for partitions merging.                                  */
  }

  return QcstSuccessful;
}  /* end memberIsJoining()                                               */


/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Action code = QcstCrgAcFailover                                         */
/*                                                                         */
/* Cluster resource services on a particular node(s) has failed or ended   */
/* for this cluster resource group.  The Failover action code is passed    */
/* regardless of whether the CRG is active or inactive.  Failover can      */
/* happen for a number of reasons:                                         */
/*                                                                         */
/*   - an operator cancelled the CRG job on a node.  Action code dependent */
/*     data of QcstMemberFailure will be passed.                           */
/*   - cluster resource services was ended on the node (for example, the   */
/*     QSYSWRK subsystem was ended with CRS still active).  Action code    */
```

```
/*     dependent data of QcstNodeFailure will be passed.                  */
/*   - the application for an application CRG has failed on the primary    */
/*     node and could not be restarted there.  The CRG is Active.         */
/*     Action code dependent data of QcstApplFailure will be passed.      */
/*   - the node failed (such as a power failure).  Action code dependent   */
/*     data of QcstNodeFailure will be passed.                            */
/*   - The cluster has become partitioned due to some communication failure*/
/*     such as a communication line or LAN failure.  The Failover action   */
/*     code is passed to recovery domain nodes in the majority partition.  */
/*     Nodes in the minority partition see the End action code.  Action    */
/*     code dependent data of QcstPartitionFailure will be passed.        */
/*   - A node in the CRG's recovery domain is being ended with the         */
/*     QcstEndClusterNode API.  The node being ended will see the End Node */
/*     action code.  All other nodes in the recovery domain will see the   */
/*     Failover action code.  Action code dependent data of QcstEndNode    */
/*     will be passed for the Failover action code.                       */
/*   - An active recovery domain node for an active CRG is being removed   */
/*     from the cluster with the QcstRemoveClusterNodeEntry API.   Action  */
/*     code dependent data of QcstRemoveNode will be passed.  If an        */
/*     inactive node is removed for an active CRG, or if the CRG is        */
/*     inactive, an action code of Remove Node is passed.                 */
/*                                                                        */
/* The exit program is called regardless of whether or not the CRG is     */
/* active.  The exit program may have nothing to do if the CRG is not     */
/* active.                                                                */
/*                                                                        */
/* If the CRG is active and the leaving member was the primary node,      */
/* perform the functions necessary for failover to a new primary.         */
/*                                                                        */
/* The Action_Code_Dependent_Data field can be used to determine if:      */
/*  - the failure was due to a problem that caused the cluster to become   */
/*     partitioned (all CRGs which had the partitioned nodes in the       */
/*     recovery domain are affected)                                      */
/*  - a node failed or had cluster resource services ended on the node (all*/
/*    CRGs which had the failed/ended node in the recovery  domain are     */
/*    affected)                                                           */
/*  - only a single CRG was affected (for example a single CRG job was     */
/*    cancelled on a node or a single application failed)                 */
/*                                                                        */
/*                                                                        */
/* Things to consider:                                                    */
/*   - Prepare the new primary node so the application can be started.     */
/*   - The application should NOT be started at this time.  The exit       */
/*     program will be called again with the QcstCrgAcStart action code if */
/*     the CRG was active when the failure occurred.                      */
/*   - If the application CRG is active, ensure the required data CRGs are */
/*     active.                                                            */
/*                                                                        */
/**************************************************************************/
static int memberIsLeaving(int role,
                           int priorRole,
                           Qcst_EXTP0100_t *crgData,
                           EpData *epData) {

  /*----------------------------------------------------------------------*/
  /*                                                                      */
  /* If the CRG is active, perform failover.  Otherwise, nothing to do.   */
  /*                                                                      */
  /*----------------------------------------------------------------------*/
  if (crgData->Original_Cluster_Res_Grp_Stat == QcstCrgActive) {

    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* The CRG is active.  Determine if my role has changed and I am now  */
    /* the new primary.                                                   */
    /*                                                                    */
    /*--------------------------------------------------------------------*/
```

```
    if (priorRole != role && role == QcstPrimaryNodeRole) {

      /*------------------------------------------------------------------*/
      /*                                                                  */
      /* I was not the primary but am now.  Do failover actions but don't */
      /* start the application at this time because this exit program will */
      /* be called again with the Start action code.                      */
      /*                                                                  */
      /*------------------------------------------------------------------*/

      /*------------------------------------------------------------------*/
      /*                                                                  */
      /* Ensure the data area status on this node starts out indicating   */
      /* the application is not running.                                  */
      /*                                                                  */
      /*------------------------------------------------------------------*/
      setApplCrgDataArea(Appl_Ended);

      /*------------------------------------------------------------------*/
      /*                                                                  */
      /* If the application has no actions to do on the Start action code */
      /* and will become active as soon as the takeover IP address is     */
      /* activated, then this code should be uncommented.  This code will */
      /* determine if all CRGs that this application CRG is dependent upon */
      /* are ready.  If this check fails, return failure from the action  */
      /* code.                                                            */
      /*                                                                  */
      /*------------------------------------------------------------------*/
/*      if (QcstSuccessful != checkDependCrgDataArea(MaxWaitSeconds))  */
/*         return QcstFailWithOutRestart;                              */

    }
  }

  return QcstSuccessful;
}  /* end memberIsLeaving()                                             */


/**************************************************************************/
/*                                                                        */
/* Action code = QcstCrgAcSwitchover                                      */
/*                                                                        */
/* The QcstInitiateSwitchOver API was called.  The first backup node in   */
/* the cluster resource group's recovery domain is taking over as the     */
/* primary node and the current primary node is being made the last backup.*/
/*                                                                        */
/* Things to consider:                                                    */
/*   - Prepare the new primary node so the application can be started.    */
/*   - The application should NOT be started at this time.  The exit      */
/*     program will be called again with the QcstCrgAcStart action code.  */
/*   - The job running the application is cancelled and the IP takeover   */
/*     address is ended prior to the exit program being called on the     */
/*     current primary.                                                   */
/*   - Ensure required data or device CRGs have switched over and are     */
/*     active.                                                            */
/*                                                                        */
/**************************************************************************/
static int switchPrimary(int role,
                         int priorRole,
                         Qcst_EXTP0100_t *crgData,
                         EpData *epData) {

  /*------------------------------------------------------------------*/
  /*                                                                  */
  /* See if I am the old primary.                                     */
  /*                                                                  */
```

```
  /*--------------------------------------------------------------------*/
  if (priorRole == QcstPrimaryNodeRole) {
    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Do what ever needs to be done to cleanup the old primary before the */
    /* switch.  Remember that that job which was running the exit program  */
    /* which started the application was cancelled already.             */
    /*                                                                  */
    /* One example may be to clean up any processes holding locks on the   */
    /* database.  This may have been done by the application cancel        */
    /* handler if one was invoked.                                      */
    /*------------------------------------------------------------------*/
  }

  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* I'm not the old primary.  See if I'm the new primary.              */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  else if (role == QcstPrimaryNodeRole) {
    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Do what ever needs to be done on the new primary before the      */
    /* application is started with the QcstCrgAcStart action code.      */
    /*                                                                  */
    /*------------------------------------------------------------------*/

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Ensure the data area status on this nodes starts out indicating  */
    /* the application is not running.                                  */
    /*                                                                  */
    /*------------------------------------------------------------------*/
    setApplCrgDataArea(Appl_Ended);

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* If the application has no actions to do on the Start action code */
    /* and will become active as soon as the takeover IP address is     */
    /* activated, then this code should be uncommented.  This code will */
    /* determine if all CRGs that this application CRG is dependent upon */
    /* are ready.  If this check fails, return failure from the action  */
    /* code.                                                            */
    /*                                                                  */
    /*------------------------------------------------------------------*/
/*      if (QcstSuccessful != checkDependCrgDataArea(MaxWaitSeconds))   */
/*         return QcstFailWithOutRestart;                               */

  }
  else {
    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* This node is one of the other backup nodes or it is a replicate  */
    /* node.  If there is anything those nodes have to do, do it here.  If */
    /* not, remove this else block.                                     */
    /*                                                                  */
    /*------------------------------------------------------------------*/

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Ensure the data area status on this nodes starts out indicating  */
    /* the application is not running.                                  */
    /*                                                                  */
    /*------------------------------------------------------------------*/
    setApplCrgDataArea(Appl_Ended);
  }
```

```
  return QcstSuccessful;
}  /* end switchPrimary()                                                  */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcAddNode                                        */
/*                                                                       */
/* The QcstAddNodeToRcvyDomain API was called.  A new node is being added  */
/* to the recovery domain of a cluster resource group.                   */
/*                                                                       */
/* Things to consider:                                                   */
/*   - A new node is being added to the recovery domain.  See the        */
/*     considerations in the createCrg() function.                       */
/*   - If this CRG is enabled to use the QcstDistributeInformation API,  */
/*     the user queue needed by that API could be created at this time.  */
/*                                                                       */
/*************************************************************************/
static int addNode(int role,
                   int priorRole,
                   Qcst_EXTP0100_t *crgData,
                   EpData *epData) {

  /*---------------------------------------------------------------------*/
  /*                                                                     */
  /* Determine if I am the node being added.                             */
  /*                                                                     */
  /*---------------------------------------------------------------------*/
  if (0 == memcmp(&crgData->This_Nodes_ID,
                  &crgData->Changing_Node_ID,
                  sizeof(Qcst_Node_Id_t)))
  {
    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Set the status of the data area on this new node.                 */
    /*                                                                   */
    /*-------------------------------------------------------------------*/
    setApplCrgDataArea(Appl_Ended);

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Create the queue needed by the Distribute Information API.        */
    /*                                                                   */
    /*-------------------------------------------------------------------*/
    if (0 == memcmp(&crgData->DI_Queue_Name,
                    Nulls,
                    sizeof(crgData->DI_Queue_Name)))
    {
    }
  }

  return QcstSuccessful;
}  /* end addNode()                                                      */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcRemoveNode                                     */
/*                                                                       */
/* The QcstRemoveNodeFromRcvyDomain or the QcstRemoveClusterNodeEntry    */
/* API was called.  A node is being removed from the recovery domain of  */
/* a cluster resource group or it is being removed entirely from the     */
/* cluster.                                                              */
/*                                                                       */
/* This action code is seen by:                                          */
/*  For the QcstRemoveClusterNodeEntry API:                              */
/*    - If the removed node is active and the CRG is Inactive, all nodes in*/
```

```
/*      the recovery domain including the node being removed see this      */
/*      action code.  The nodes NOT being removed see action code dependent*/
/*      data of QcstNodeFailure.                                           */
/*    - If the removed node is active and the CRG is Active, the node being*/
/*      removed sees the Remove Node action code.  All other nodes in the  */
/*      recovery domain see an action code of Failover and action code     */
/*      dependent data of QcstNodeFailure.                                 */
/*    - If the node being removed is not active in the cluster, all nodes  */
/*      in the recovery domain will see this action code.                  */
/*  For the QcstRemoveNodeFromRcvyDomain API:                              */
/*    - All nodes see the Remove Node action code regardless of whether or */
/*      not the CRG is Active.  Action code dependent data of              */
/*      QcstRmvRcvyDmnNode will also be passed.                            */
/*                                                                         */
/* Things to consider:                                                     */
/*   - You may want to cleanup the removed node by deleting objects no     */
/*     longer needed there.                                                */
/*   - The job running the application is cancelled and the IP takeover    */
/*     address is ended after the exit program is called if this is the    */
/*     primary node and the CRG is active.                                 */
/*   - If subsystems or server jobs were started as a result of the        */
/*     QcstCrgAcStart action code, end them here or consolidate all logic  */
/*     to end the application in the cancelHandler() since it will be      */
/*     invoked for all Cluster Resource Services APIs which must end the   */
/*     application on the current primary.                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static int rmvNode(int role,
                   int priorRole,
                   Qcst_EXTP0100_t *crgData,
                   EpData *epData) {

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* Determine if I am the node being removed.                             */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/
  if (0 == memcmp(&crgData->This_Nodes_ID,
                  &crgData->Changing_Node_ID,
                  sizeof(Qcst_Node_Id_t)))
  {
      /*-------------------------------------------------------------------*/
      /*                                                                   */
      /* End the application if it is running on this node.                */
      /*                                                                   */
      /*-------------------------------------------------------------------*/
      endApplication(QcstCrgAcRemoveNode, role, priorRole, crgData, epData);

  }
  return QcstSuccessful;
}  /* end rmvNode                                                         */


/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Action code = QcstCrgAcChange                                           */
/*                                                                         */
/* The QcstChangeClusterResourceGroup API was called.  Some attribute      */
/* or information stored in the cluster resource group object is being     */
/* changed.  Note that not all changes to the CRG object cause the exit    */
/* program to be called.  As of V5R1M0, only these changes will cause the  */
/* exit program to be called-                                              */
/*   - the current recovery domain is being changed                        */
/*   - the preferred recovery domain is being changed                      */
/*                                                                         */
/* If any of the above changes are being made but additionally the exit    */
/* program is being changed to *NONE, the exit program is not called.      */
```

```
/*                                                                    */
/* Things to consider:                                                */
/*   - None unless changing the recovery domain affects information or*/
/*     processes for this cluster resource group.  Note that the primary*/
/*     node cannot be changed with the QcstChangeClusterResourceGroup API*/
/*     if the CRG is active.                                          */
/*                                                                    */
/**********************************************************************/
static int chgCrg(int role,
                  int priorRole,
                  Qcst_EXTP0100_t *crgData,
                  EpData *epData) {

  return QcstSuccessful;
}  /* end chgCrg()                                                    */


/**********************************************************************/
/*                                                                    */
/* Action code = QcstCrgAcDeleteCommand                               */
/*                                                                    */
/* The Delete Cluster Resource Group (DLTCRG) CL command has been called*/
/* to delete a cluster resource group object, the QcstDeleteCluster API*/
/* has been called, or the QcstRemoveClusterNodeEntry API has been called.*/
/* In each case, cluster resource services is not active on the cluster*/
/* node where the command or API was called.  Thus, this function is not*/
/* distributed cluster wide but occurs only on the node where the CL  */
/* command or API was called.                                         */
/*                                                                    */
/* If the QcstDeleteCluster API was used, action code dependent data of*/
/* QcstDltCluster is passed.                                          */
/*                                                                    */
/* See the considerations in the deleteCrg() function                 */
/*                                                                    */
/**********************************************************************/
static int deleteCrgWithCmd(int role,
                            int doesNotApply,
                            Qcst_EXTP0100_t *crgData,
                            EpData *epData) {

  return QcstSuccessful;
}  /* end deleteCrgWithCmd()                                          */


/**********************************************************************/
/*                                                                    */
/* Action code = QcstCrgEndNode                                       */
/*                                                                    */
/* The QcstEndClusterNode API was called or a CRG job was cancelled.  */
/*                                                                    */
/* The QcstCrgEndNode action code is passed to the exit program only on the*/
/* node being ended or where the CRG job was cancelled.  On the node where */
/* a Cluster Resource Services job is cancelled, action code dependent data*/
/* of QcstMemberFailure will be passed.                               */
/* When Cluster Resource Services ends on this node or the CRG job ends, it*/
/* will cause all other nodes in the cluster to go through failover   */
/* processing.  The action code passed to all other nodes will be     */
/* QcstCrgAcFailover.   Those nodes will see action code dependent data of */
/* QcstMemberFailure if a CRG job is cancelled or QcstNodeFailure if the  */
/* node is ended.                                                     */
/*                                                                    */
/* Things to consider:                                                */
/*   - The job running the application is cancelled and the IP takeover*/
/*     address is ended after the exit program is called if this is the*/
/*     primary node and the CRG is active.                            */
/*   - If subsystems or server jobs were started as a result of the   */
/*     QcstCrgAcStart action code, end them here.                     */
```

```
/*                                                                    */
/**********************************************************************/
static int endNode(int role,
                   int priorRole,
                   Qcst_EXTP0100_t *crgData,
                   EpData *epData) {

  /*------------------------------------------------------------------*/
  /*                                                                  */
  /* End the application if it is running on this node.               */
  /*                                                                  */
  /*------------------------------------------------------------------*/
  endApplication(QcstCrgEndNode, role, priorRole, crgData, epData);

  return QcstSuccessful;
}  /* end endNode()                                                 */


/**********************************************************************/
/*                                                                    */
/* Action code = QcstCrgAcChgNodeStatus                               */
/*                                                                    */
/* The QcstChangeClusterNodeEntry API was called.  The status of a node */
/* is being changed to failed.  This API is used to inform cluster resource*/
/* services that the node did not partition but really failed.        */
/*                                                                    */
/* Things to consider:                                                */
/*   - The exit program was called previously with an action code of  */
/*     QcstCrgAcEnd if the CRG was active or an action code of        */
/*     QcstCrgAcFailover if the CRG was inactive because cluster resource */
/*     services thought the cluster had become partitioned.  The user is */
/*     now telling cluster resource services that the node really failed */
/*     instead of partitioned.  The exit program has something to do only */
/*     if it performed some action previously that needs to be changed now */
/*     that node failure can be confirmed.                            */
/*                                                                    */
/**********************************************************************/
static int chgNodeStatus(int role,
                         int priorRole,
                         Qcst_EXTP0100_t *crgData,
                         EpData *epData) {

  return QcstSuccessful;
}  /* end chgNodeStatus()                                           */


/**********************************************************************/
/*                                                                    */
/* Action code = QcstCrgAcCancelFailover                              */
/*                                                                    */
/* Cluster resource services on the primary node has failed or ended  */
/* for this cluster resource group.  A message was sent to the failover */
/* message queue specified for the CRG, and the result of that message */
/* was to cancel the failover.  This will change the status of the CRG to */
/* inactive and leave the primary node as primary.                    */
/*                                                                    */
/* Things to consider:                                                */
/*   - The primary node is no longer participating in cluster activities. */
/*     The problem which caused the primary node to fail should be fixed */
/*     so that the CRG may be started again.                          */
/*                                                                    */
/**********************************************************************/
static int cancelFailover(int role,
                          int priorRole,
                          Qcst_EXTP0100_t *crgData,
                          EpData *epData) {
```

```
  return QcstSuccessful;
}  /* end cancelFailover()                                                  */


/*************************************************************************/
/*                                                                       */
/* Action code = exit program does not know it yet                       */
/*                                                                       */
/* A new action code has been passed to this exit program.  This can occur */
/* after a new OS/400 release has been installed and some new cluster API  */
/* was called or some new cluster event occurred.  The logic in this exit  */
/* program has not yet been updated to understand the new action code.     */
/*                                                                       */
/* Two different strategies could be used for the new action code.  The    */
/* correct strategy is dependent upon the kinds of things this particular  */
/* exit program does for the application.                                */
/*                                                                       */
/* One strategy is to not do anything and return a successful return code. */
/* This allows the new cluster API or event to run to completion.  It      */
/* allows the function to be performed even though this exit program       */
/* did not understand the new action code.  The risk, though, is that the  */
/* exit program should have done something and it did not.  At a minimum,  */
/* you may want to log some kind of error message about what happened so   */
/* that programming can investigate and get the exit program updated.      */
/*                                                                       */
/* The opposite strategy is to return an error return code such as         */
/* QcstFailWithRestart.  Of course doing this means that the new cluster   */
/* API or event cannot be used until the exit program is updated for the   */
/* new action code.  Again, logging some kind of error message for         */
/* programming to investigate would be worthwhile.                       */
/*                                                                       */
/* Only the designer of the exit program can really decide which is the    */
/* better course of action.                                              */
/*                                                                       */
/*************************************************************************/
static int newActionCode(int role,
                         int doesNotApply,
                         Qcst_EXTP0100_t *crgData,
                         EpData *epData) {

  /*---------------------------------------------------------------------*/
  /*                                                                     */
  /* Add logic to log an error somewhere - operator message queue, job     */
  /* log, application specific error log, etc. so that the exit program    */
  /* gets updated to properly handle the new action code.                */
  /*                                                                     */
  /* Note that if this is left coded as it is, this is the "don't do       */
  /* anything" strategy described in the prologue above.                 */
  /*                                                                     */
  /*---------------------------------------------------------------------*/

  return QcstSuccessful;
}  /* end newActionCode()                                                  */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcUndo                                           */
/*                                                                       */
/* Note: The exit program is never called with an undo action code for     */
/* any of these prior action codes:                                      */
/*   QcstCrgAcChgNodeStatus                                              */
/*   QcstCrgAcDelete                                                     */
/*   QcstCrgAcDeleteCommand                                              */
/*   QcstCrgEndNode                                                      */
/*   QstCrgAcRemoveNode (If the node being removed is active in the        */
/*                       cluster and the API is Remove Cluster Node.       */
```

```
/*                      The Remove Node From Recovery Domain will call    */
/*                      with Undo and the Remove Cluster Node API will    */
/*                      call with Undo if the node being removed is       */
/*                      inactive.                                         */
/*   QcstCrgAcRestart                                                     */
/*   QcstCrgAcUndo                                                        */
/*                                                                        */
/* APIs that call an exit program do things in 3 steps.                   */
/*   1. Logic which must be done prior to calling the exit program.       */
/*   2. Call the exit program.                                            */
/*   3. Logic which must be done after calling the exit program.          */
/*                                                                        */
/* Any errors that occur during steps 2 or 3 result in the exit program   */
/* being called again with the undo action code.  This gives the exit     */
/* program an opportunity to back out any work performed when it was first */
/* called by the API.  The API will also be backing out any work it       */
/* performed trying to return the state of the cluster and cluster objects */
/* to what it was before the API was called.                              */
/*                                                                        */
/* It is suggested that the following return codes be returned for the    */
/* specified action code as that return code will result in the most      */
/* appropriate action being taken.                                        */
/*                                                                        */
/*   QcstCrgAcInitialize: QcstSuccessful; The CRG is not created.         */
/*   QcstCrgAcStart:      QcstSuccessful; The CRG is not started.         */
/*   QcstCrgAcEnd:        QcstFailWithOutRestart; The CRG is set to Indoubt*/
/*                                        The cause of the failure needs to*/
/*                                        investigated.                   */
/*   QcstCrgAcReJoin:     QcstFailWithOutRestart; The CRG is set to Indoubt*/
/*                                        The cause of the failure needs to*/
/*                                        investigated.                   */
/*   QcstCrgAcFailover:   QcstFailWithOutRestart; The CRG is set to Indoubt*/
/*                                        The cause of the failure needs to*/
/*                                        investigated.                   */
/*   QcstCrgAcSwitchover: QcstFailWithOutRestart; The CRG is set to Indoubt*/
/*                                        The cause of the failure needs to*/
/*                                        investigated.                   */
/*   QcstCrgAcAddNode:    QcstSuccessful; The node is not added.          */
/*   QcstCrgAcRemoveNode: QcstFailWithOutRestart; The CRG is set to Indoubt*/
/*                                        The cause of the failure needs to*/
/*                                        investigated.                   */
/*   QcstCrgAcChange:     QcstSuccessful; The recovery domain is not      */
/*                                        changed.                        */
/*                                                                        */
/**************************************************************************/
static int undoPriorAction(int role,
                           int priorRole,
                           Qcst_EXTP0100_t *crgData,
                           EpData *epData) {

  /*----------------------------------------------------------------------*/
  /*                                                                      */
  /* The prior action code defines what the exit program was doing when   */
  /* it failed, was cancelled, or returned a non successful return code.  */
  /*                                                                      */
  /*----------------------------------------------------------------------*/
  if (crgData->Prior_Action_Code &lt;= MaxAc )
    return (*undoFcn[crgData-&lt;Prior_Action_Code])
                                      (role, priorRole, crgData, epData);
  else
    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* IBM has defined a new action code in a new operating system release */
    /* and this exit program has not yet been updated to handle it.  Take a*/
    /* default action for now.                                            */
    /*                                                                    */
    /*--------------------------------------------------------------------*/
```

```
    return newActionCode(role, priorRole, crgData, epData);
}  /* end undoPriorAction()                                                  */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcUndo                                           */
/*                                                                       */
/* Prior action code = QcstCrgAcInitialize                               */
/*                                                                       */
/* Things to consider:                                                   */
/*   The CRG will not be created.  Objects that might have been created  */
/*   on nodes in the recovery domain should be deleted since a subsequent*/
/*   create could fail if those objects already exist.                   */
/*                                                                       */
/*************************************************************************/
static int undoCreateCrg(int role,
                         int doesNotApply,
                         Qcst_EXTP0100_t *crgData,
                         EpData *epData) {

  return QcstSuccessful;
}  /* end undoCreateCrg()                                                */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcUndo                                           */
/*                                                                       */
/* Prior action code = QcstCrgAcStart                                    */
/*                                                                       */
/* Things to consider:                                                   */
/*   Cluster Resource Services failed when it was finishing the Start CRG*/
/*   API after it had already called the exit program with the Start     */
/*   Action code.                                                        */
/*                                                                       */
/*   On the primary node, the exit program job which is running the      */
/*   application will be cancelled.  The exit program will then be called*/
/*   with the Undo action code.                                          */
/*                                                                       */
/*   All other nodes in the recovery domain will be called with the Undo */
/*   action code.                                                        */
/*                                                                       */
/*************************************************************************/
static int undoStartCrg(int role,
                        int doesNotApply,
                        Qcst_EXTP0100_t *crgData,
                        EpData *epData) {

  return QcstSuccessful;
}  /* end undoStartCrg()                                                 */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcUndo                                           */
/*                                                                       */
/* Prior action code = QcstCrgAcEnd                                      */
/*                                                                       */
/* Things to consider:                                                   */
/*   The CRG will not be ended.  If the exit program did anything to bring*/
/*   down the application it can either restart the application or it can*/
/*   decide to not restart the application.  If the application is not   */
/*   restarted, the return code should be set to QcstFailWithOutRestart so*/
/*   the status of the CRG is set to Indoubt.                            */
/*                                                                       */
/*************************************************************************/
```

```
static int undoEndCrg(int role,
                      int doesNotApply,
                      Qcst_EXTP0100_t *crgData,
                      EpData *epData) {

  return QcstFailWithOutRestart;
}  /* end undoEndCrg()                                                    */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcUndo                                           */
/*                                                                       */
/* Prior action code = QcstCrgAcReJoin                                   */
/*                                                                       */
/* Things to consider:                                                   */
/*   An error occurred which won't allow the member to join this CRG     */
/*   group.  Anything done for the Join action code needs to be looked at*/
/*   to see if something must be undone if this member is not an active  */
/*   member of the CRG group.                                            */
/*                                                                       */
/*************************************************************************/
static int undoMemberIsJoining(int role,
                               int doesNotApply,
                               Qcst_EXTP0100_t *crgData,
                               EpData *epData) {

  return QcstFailWithOutRestart;
}  /* end undoMemberIsJoining()                                          */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcUndo                                           */
/*                                                                       */
/* Prior action code = QcstCrgAcFailover                                 */
/*                                                                       */
/* Things to consider:                                                   */
/*   This does not mean that the node failure or failing member is being */
/*   undone.  That failure is irreversible.  What it does mean is that the*/
/*   exit program returned an error from the Failover action code or     */
/*   Cluster Resource Services ran into a problem after it called the exit*/
/*   program.  If the CRG was active when Failover was attempted, it is  */
/*   not at this point.  End the resilient resource and expect a human to*/
/*   look into the failure.  After the failure is corrected, the CRG will*/
/*   have to be started with the Start CRG API.                          */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
static int undoMemberIsLeaving(int role,
                               int doesNotApply,
                               Qcst_EXTP0100_t *crgData,
                               EpData *epData) {

  return QcstFailWithOutRestart;
}  /* end undoMemberIsLeaving()                                          */


/*************************************************************************/
/*                                                                       */
/* Action code = QcstCrgAcUndo                                           */
/*                                                                       */
/* Prior action code = QcstCrgAcSwitchover                               */
/*                                                                       */
/* Things to consider:                                                   */
/*   Some error occurred after the point of access was moved from the    */
/*   original primary and before it could be brought up on the new primary.*/
```

```
/*   The IP address was ended on the original primary before moving the    */
/*   point of access but is started on the original primary again.  Cluster*/
/*   Resource Services will now attempt to move the point of access back   */
/*   to the original primary.  The application exit program and IP takeover*/
/*   address will be started on the original primary.                      */
/*                                                                         */
/*                                                                         */
/*************************************************************************/
static int undoSwitchPrimary(int role,
                             int doesNotApply,
                             Qcst_EXTP0100_t *crgData,
                             EpData *epData) {

  return QcstFailWithOutRestart;
}  /* end undoSwitchPrimary()                                             */


/*************************************************************************/
/*                                                                         */
/* Action code = QcstCrgAcUndo                                             */
/*                                                                         */
/* Prior action code = QcstCrgAcAddNode                                    */
/*                                                                         */
/* Things to consider:                                                     */
/*   If objects were created on the new node, they should be removed so    */
/*   that a subsequent Add Node to aRecovery Domain does not fail if it     */
/*   attempts to create objects again.                                     */
/*                                                                         */
/*                                                                         */
/*************************************************************************/
static int undoAddNode(int role,
                       int doesNotApply,
                       Qcst_EXTP0100_t *crgData,
                       EpData *epData) {

  return QcstSuccessful;
}  /* end undoAddNode()                                                   */


/*************************************************************************/
/*                                                                         */
/* Action code = QcstCrgAcUndo                                             */
/*                                                                         */
/* Prior action code = QcstCrgAcRemoveNode                                 */
/*                                                                         */
/* Things to consider:                                                     */
/*   The node is still in the recovery domain.  If objects were removed    */
/*   from the node, they should be added back.                             */
/*                                                                         */
/*************************************************************************/
static int undoRmvNode(int role,
                       int doesNotApply,
                       Qcst_EXTP0100_t *crgData,
                       EpData *epData) {

  return QcstFailWithOutRestart;
}  /* end undoRmvNode()                                                   */


/*************************************************************************/
/*                                                                         */
/* Action code = QcstCrgAcUndo                                             */
/*                                                                         */
/* Prior action code = QcstCrgAcChange                                     */
/*                                                                         */
/* Things to consider:                                                     */
/*   Changes to the CRG will be backed out so that the CRG and its         */
```

```
/*   recovery domain look just like it did prior to the attempted change.  */
/*   Any changes the exit program made should also be backed out.          */
/*                                                                         */
/***************************************************************************/
static int undoChgCrg(int role,
                      int doesNotApply,
                      Qcst_EXTP0100_t *crgData,
                      EpData *epData) {

  return QcstSuccessful;
}  /* end undoChgCrg()                                                     */


/***************************************************************************/
/*                                                                         */
/* Action code = QcstCrgAcUndo                                             */
/*                                                                         */
/* Prior action code = QcstCrgAcCancelFailover                             */
/*                                                                         */
/* Things to consider:                                                     */
/*   This does not mean that the node failure or failing member is being   */
/*   undone.  That failure is irreversible.  What it does mean is that     */
/*   Cluster Resource Services ran into a problem after it called the exit */
/*   program.  The CRG will be InDoubt regardless of what is returned from */
/*   this exit program call.  Someone will need to manually look into the  */
/*   the failure.  After the failure is corrected, the CRG will have to be */
/*   started with the Start CRG API.                                       */
/*                                                                         */
/*                                                                         */
/***************************************************************************/
static int undoCancelFailover(int role,
                              int doesNotApply,
                              Qcst_EXTP0100_t *crgData,
                              EpData *epData) {

  return QcstSuccessful;
}  /* end undoCancelFailover()                                             */


/***************************************************************************/
/*                                                                         */
/* A simple routine to take a null terminated object name and a null       */
/* terminated library name and build a 20 character non-null terminated    */
/* qualified name.                                                         */
/*                                                                         */
/***************************************************************************/
static void bldDataAreaName(char *objName, char* libName, char *qualName) {

  memset(qualName, 0x40, 20);
  memcpy(qualName, objName, strlen(objName));
  qualName += 10;
  memcpy(qualName, libName, strlen(libName));
  return;
}  /* end bldDataAreaName                                                  */


/***************************************************************************/
/*                                                                         */
/* The data area is checked to see if all the CRGs that this application   */
/* is dependent upon are ready.  If they are not ready, a wait for a       */
/* certain amount of time is performed and the data area is checked again. */
/* This check, wait loop continues until all dependent CRGs become ready or*/
/* until the maximum wait time has been reached.                           */
/* The length of the wait can be changed to some other value if a          */
/* particular situation would be better with shorter or longer wait times. */
/*                                                                         */
/*                                                                         */
```

```
/*****************************************************************************/
static int checkDependCrgDataArea(unsigned int maxWaitTime) {

  Qus_EC_t errCode = { sizeof(Qus_EC_t), 0 };
  char dataAreaName[20];
  struct {
    Qwc_Rdtaa_Data_Returned_t stuff;
    char ready;
  } data;

  /*---------------------------------------------------------------------------*/
  /*                                                                           */
  /* This is an accumulation of the time waited for the dependent CRGs to      */
  /* become ready.                                                             */
  /*                                                                           */
  /*---------------------------------------------------------------------------*/
  unsigned int timeWaited = 0;

  /*---------------------------------------------------------------------------*/
  /*                                                                           */
  /* Build definition of the amount of time to wait.                           */
  /*                                                                           */
  /*---------------------------------------------------------------------------*/
  _MI_Time   timeToWait;
  int hours     = 0;
  int minutes   = 0;
  int seconds   = WaitSecondsIncrement;
  int hundreths = 0;
  short int options = _WAIT_NORMAL;
  mitime( &timeToWait, hours, minutes, seconds, hundreths );

  /*---------------------------------------------------------------------------*/
  /*                                                                           */
  /* Build the qualified name of the data area.                                */
  /*                                                                           */
  /*---------------------------------------------------------------------------*/
  bldDataAreaName(DependCrgDataArea, ApplLib, dataAreaName);

  /*---------------------------------------------------------------------------*/
  /*                                                                           */
  /* Get the data from the data area that indicates whether or not the         */
  /* CRGs are all ready.  This data area is updated by the High                */
  /* Availability Business Partners when it is ok for the application to       */
  /* proceed.                                                                  */
  /*                                                                           */
  /*---------------------------------------------------------------------------*/
  QWCRDTAA(&data,
           sizeof(data),
           dataAreaName,
           offsetof(Qcst_HAAPPO_t,Data_Status)+1,  /* API wants a 1 origin */
           sizeof(data.ready),
           &errCode);

  /*---------------------------------------------------------------------------*/
  /*                                                                           */
  /* If the dependent CRGs are not ready, wait for a bit and check again.      */
  /*                                                                           */
  /*---------------------------------------------------------------------------*/
  while (data.ready != Data_Available) {

    /*-------------------------------------------------------------------------*/
    /*                                                                         */
    /* If the dependent CRGs have not become ready by the time we have         */
    /* waited our maximum wait time, return an error.  Consider logging        */
    /* some message to describe why the application did not start so that      */
    /* the problem can be looked into.                                         */
    /*                                                                         */
```

```
    /*------------------------------------------------------------------*/
    if (timeWaited >= maxWaitTime)
      return QcstFailWithOutRestart;

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Wait to allow the data CRGs to become ready.                     */
    /*                                                                  */
    /*------------------------------------------------------------------*/
    waittime(&timeToWait, options);
    timeWaited += WaitSecondsIncrement;

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Get information from the data area again to see if the data CRGs are*/
    /* ready.                                                           */
    /*                                                                  */
    /*------------------------------------------------------------------*/
    QWCRDTAA(&data,
             sizeof(data),
             dataAreaName,
             offsetof(Qcst_HAAPPO_t,Data_Status)+1,  /* API wants a 1 origin */
             sizeof(data.ready),
             &errCode);
  }

  return QcstSuccessful;
}  /* end checkDependCrgDataArea                                    */


/**********************************************************************/
/*                                                                    */
/* The application CRG data area is updated to indicate that the      */
/* application is running or to indicate it is not running.  This data area*/
/* information is used by the High Availability Business Partners to  */
/* coordinate the switchover activities between CRGs that have dependencies*/
/* on each other.                                                     */
/*                                                                    */
/**********************************************************************/
static void setApplCrgDataArea(char status) {

  char cmd[54];
  char cmdEnd[3] = {0x00, ')', 0x00};

  /*------------------------------------------------------------------*/
  /*                                                                  */
  /* Set up the CL command string with the data area library name, the data*/
  /* area name, and the character to put into the data area.  Then run the */
  /* CL command.                                                      */
  /*                                                                  */
  /*------------------------------------------------------------------*/
  memcpy(cmd, "CHGDTAARA DTAARA(", strlen("CHGDTAARA DTAARA(")+1);
  strcat(cmd, ApplLib);
  strcat(cmd, "/");
  strcat(cmd, ApplCrgDataArea);
  strcat(cmd, " (425 1)) VALUE(");                          /* @A1C */
  cmdEnd[0] = status;
  strcat(cmd, cmdEnd);

  system(cmd);

  return;
}  /* end setApplCrgDataArea                                        */


/**********************************************************************/
/*                                                                    */
```

```
/* This function is called any time the exit program receives an exception */
/* not specifically monitored for by some other exception handler.  Add    */
/* appropriate logic to perform cleanup functions that may be required.    */
/* A failure return code is then set and control returns to the operating  */
/* system.  The job this exit program is running in will then end.         */
/*                                                                         */
/* When this function gets called, myData->role may still contain the      */
/* UnknownRole value if an exception occurred before this node's role      */
/* value was set.  To be completely correct, the role should be tested     */
/* for UnknownRole before making any decisions based upon the value of     */
/* role.                                                                   */
/*                                                                         */
/*-------------------------------------------------------------------------*/
static void unexpectedExceptionHandler(_INTRPT_Hndlr_Parms_T *exData) {

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* Get a pointer to the structure containing data we passed to the       */
  /* exception handler.                                                    */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/
  HandlerDataT *myData = (HandlerDataT *)exData->Com_Area;

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* Perform as much cleanup function as necessary.  Some global state     */
  /* information may have to be kept so the exception handler knows what   */
  /* steps were completed before the failure occurred and thus knows what  */
  /* cleanup steps must be performed.  This state information could be     */
  /* kept in the HandlerDataT structure or it could be kept in some other  */
  /* location that this function can address.                              */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* If this is the primary node and the application was started, end it.  */
  /* The application is ended because the exit program will be called again*/
  /* with the Restart action code and we want the restartCrg() function to */
  /* always work the same way.  In addition, ending the application may    */
  /* clear up the condition that caused the exception that got us here.    */
  /* If possible, warn users and have them stop using the application so   */
  /* things are done in an orderly manner.                                 */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/
  endApplication(myData->actionCode,
                 myData->role,
                 myData->priorRole,
                 myData->crgData,
                 myData->epData);

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* Set the exit program return code.                                     */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/
  *myData->retCode = QcstFailWithRestart;

  /*-----------------------------------------------------------------------*/
  /*                                                                       */
  /* Let the exception percolate up the invocation stack.                  */
  /*                                                                       */
  /*-----------------------------------------------------------------------*/
  return;
}  /* end unexpectedExceptionHandler                                     */
```

```
/*****************************************************************************/
/*                                                                           */
/* This function is called any time the job this exit program is running in*/
/* is cancelled.  The job could be cancelled due to any of the following   */
/* (the list is not intended to be all inclusive)-                           */
/*   - an API cancels an active application CRG.  The End CRG, Initiate    */
/*     Switchover, End Cluster Node, Remove Cluster Node or Delete Cluster */
/*     API cancels the job which was submitted when the exit program was   */
/*     called with a Start action code.                                      */
/*   - operator cancels the job from some operating system display such as */
/*     Work with Active Jobs                                                 */
/*   - the subsystem this job is running in is ended                         */
/*   - all subsystems are ended                                              */
/*   - the system is powered down                                            */
/*   - an operating system machine check occurred                            */
/*                                                                           */
/* When this function gets called, myData->role may still contain the      */
/* UnknownRole value if cancelling occurred before this node's role        */
/* value was set.  To be completely correct, the role should be tested     */
/* for UnknownRole before making any decisions based upon the value of     */
/* role.                                                                     */
/*                                                                           */
/*****************************************************************************/
static void cancelHandler(_CNL_Hndlr_Parms_T *cnlData) {

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* Get a pointer to the structure containing data we passed to the       */
  /* cancel handler.                                                         */
  /*                                                                         */
  /*-------------------------------------------------------------------------*/
  HandlerDataT *myData = (HandlerDataT *)cnlData->Com_Area;

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* Perform as much cleanup function as necessary.  Some global state     */
  /* information may have to be kept so the cancel handler knows what      */
  /* steps were completed before the job was cancelled and thus knows if   */
  /* the function had really completed successfully or was only partially  */
  /* complete and thus needs some cleanup to be done.  This state          */
  /* information could be kept in the HandlerDataT structure or it could   */
  /* be kept in some other location that this function can address.        */
  /*                                                                         */
  /*-------------------------------------------------------------------------*/

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* This job is being cancelled.  If I was running the application as a   */
  /* result of the Start or Restart action codes, end the application now. */
  /* This job is being cancelled because a Switch Over or some other       */
  /* Cluster Resource Services API was used which affects the primary node */
  /* or someone did a cancel job with a CL command, from a system display, */
  /* etc.                                                                    */

  /*-------------------------------------------------------------------------*/
  endApplication(myData->actionCode,
                 myData->role,
                 myData->priorRole,
                 myData->crgData,
                 myData->epData);

  /*-------------------------------------------------------------------------*/
  /*                                                                         */
  /* Set the exit program return code.                                       */
  /*                                                                         */
  /*-------------------------------------------------------------------------*/
  *myData->retCode = QcstSuccessful;
```

```
  /*--------------------------------------------------------------------*/
  /*                                                                    */
  /* Return to the operating system for final ending of the job.        */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  return;
}  /* end cancelHandler                                                */


/**********************************************************************/
/*                                                                    */
/* A common routine used to end the application by various action code */
/* functions, the exception handler, and the cancel handler.          */
/*                                                                    */
/**********************************************************************/
static void endApplication(unsigned int actionCode,
                           int role,
                           int priorRole,
                           Qcst_EXTP0100_t *crgData,
                           EpData *epData) {

  if (  role == QcstPrimaryNodeRole
      &&
        crgData->Original_Cluster_Res_Grp_Stat == QcstCrgActive) {
    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Add logic to end the application here.  You may need to add logic */
    /* to determine if the application is still running because this    */
    /* function could be called once for an action code and again from  */
    /* the cancel handler (End CRG is an example).                      */
    /*                                                                  */
    /*------------------------------------------------------------------*/



    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* After the application has ended, update the data area to indicate */
    /* the application is no longer running.                            */
    /*                                                                  */
    /*------------------------------------------------------------------*/
    setApplCrgDataArea(Appl_Ended);
  }

  return;
}  /* end endApplication                                               */


/**********************************************************************/
/*                                                                    */
/* Print out the data passed to this program.                         */
/*                                                                    */
/**********************************************************************/
static void printParms(int actionCode,
                       int role,
                       int priorRole,
                       Qcst_EXTP0100_t *crgData,
                       EpData *epData) {

  unsigned int i;
  char *str;

  /* Print the action code.                                           */
  printf("%s", "Action_Code = ");
  printActionCode(actionCode);
```

```
/* Print the action code dependent data.                              */
printf("%s", "    Action_Code_Dependent_Data = ");
switch (crgData->Action_Code_Dependent_Data) {
  case QcstNoDependentData:  str = "QcstNoDependentData";
                             break;
  case QcstMerge:            str = "QcstMerge";
                             break;
  case QcstJoin:             str = "QcstJoin";
                             break;
  case QcstPartitionFailure: str = "QcstPartitionFailure";
                             break;
  case QcstNodeFailure:      str = "QcstNodeFailure";
                             break;
  case QcstMemberFailure:    str = "QcstMemberFailure";
                             break;
  case QcstEndNode:          str = "QcstEndNode";
                             break;
  case QcstRemoveNode:       str = "QcstRemoveNode";
                             break;
  case QcstApplFailure:      str = "QcstApplFailure";
                             break;
  case QcstResourceEnd:      str = "QcstResourceEnd";
                             break;
  case QcstDltCluster:       str = "QcstDltCluster";
                             break;
  case QcstRmvRcvyDmnNode:   str = "QcstRmvRcvyDmnNode";
                             break;
  case QcstDltCrg:           str = "QcstDltCrg";
                             break;
  default: str = "unknown action code dependent data";
}
printf("%s \n", str);


/* Print the prior action code.                                       */
printf("%s", "  Prior_Action_Code = ");
if (crgData->Prior_Action_Code)
  printActionCode(crgData->Prior_Action_Code);
printf("\n");

/* Print the cluster name.                                            */
printStr("  Cluster_Name = ",
         crgData->Cluster_Name, sizeof(Qcst_Cluster_Name_t));

/* Print the CRG name.                                                */
printStr("  Cluster_Resource_Group_Name = ",
         crgData->Cluster_Resource_Group_Name, sizeof(Qcst_Crg_Name_t));

/* Print the CRG type.                                                */
printf("%s \n", "  Cluster_Resource_Group_Type = QcstCrgApplResiliency");

/* Print the CRG status.                                              */
printf("%s", "  Cluster_Resource_Group_Status = ");
printCrgStatus(crgData->Cluster_Resource_Group_Status);

/* Print the CRG original status.                                     */
printf("%s", "  Original_Cluster_Res_Grp_Stat = ");
printCrgStatus(crgData->Original_Cluster_Res_Grp_Stat);

/* Print the Distribute Information queue name.                       */
printStr("  DI_Queue_Name = ",
         crgData->DI_Queue_Name, sizeof(crgData->DI_Queue_Name));
printStr("  DI_Queue_Library_Name = ",
         crgData->DI_Queue_Library_Name,
         sizeof(crgData->DI_Queue_Library_Name));

/* Print the CRG attributes.                                          */
```

```
printf("%s", "  Cluster_Resource_Group_Attr = ");
if (crgData->Cluster_Resource_Group_Attr & QcstTcpConfigByUsr)
  printf("%s", "User Configures IP Takeover Address");
printf("\n");

/* Print the ID of this node.                                         */
printStr("  This_Nodes_ID = ",
         crgData->This_Nodes_ID, sizeof(Qcst_Node_Id_t));

/* Print the role of this node.                                       */
printf("%s %d \n", "  this node's role = ", role);

/* Print the prior role of this node.                                 */
printf("%s %d \n", "  this node's prior role = ", priorRole);

/* Print which recovery domain this role comes from.                  */
printf("%s", "  Node_Role_Type = ");
if (crgData->Node_Role_Type == QcstCurrentRcvyDmn)
  printf("%s \n", "QcstCurrentRcvyDmn");
else
  printf("%s \n", "QcstPreferredRcvyDmn");

/* Print the ID of the changing node (if any).                        */
printStr("  Changing_Node_ID = ",
         crgData->Changing_Node_ID, sizeof(Qcst_Node_Id_t));

/* Print the role of the changing node (if any).                      */
printf("%s", "  Changing_Node_Role = ");
if (crgData->Changing_Node_Role == -3)
  printf("%s \n", "*LIST");
else if (crgData->Changing_Node_Role == -2)
 printf("%s \n", "does not apply");
else
  printf("%d \n", crgData->Changing_Node_Role);

/* Print the takeover IP address.                                     */
printStr("  Takeover_IP_Address = ",
         crgData->Takeover_IP_Address, sizeof(Qcst_TakeOver_IP_Address_t));

/* Print the job name.                                                */
printStr("  Job_Name = ", crgData->Job_Name, 10);

/* Print the CRG changes.                                             */
printf("%s \n", "  Cluster_Resource_Group_Changes = ");
if (crgData->Cluster_Resource_Group_Changes & QcstRcvyDomainChange)
  printf("      %s \n", "Recovery domain changed");
if (crgData->Cluster_Resource_Group_Changes & QcstTakeOverIpAddrChange)
  printf("      %s \n", "Takeover IP address changed");

/* Print the failover wait time.                                      */
printf("%s", "Failover_Wait_Time = ");
if (crgData->Failover_Wait_Time == QcstFailoverWaitForever)
   printf("%d %s \n", crgData->Failover_Wait_Time, "Wait forever");
else if (crgData->Failover_Wait_Time == QcstFailoverNoWait)
   printf("%d %s \n", crgData->Failover_Wait_Time, "No wait");
else
   printf("%d %s \n", crgData->Failover_Wait_Time, "minutes");

/* Print the failover default action.                                 */
printf("%s", "Failover_Default_Action = ");
if (crgData->Failover_Default_Action == QcstFailoverProceed)
   printf("%d %s \n", crgData->Failover_Default_Action, "Proceed");
else
   printf("%d %s \n", crgData->Failover_Default_Action, "Cancel");

/* Print the failover message queue name.                             */
printStr("  Failover_Msg_Queue = ",
```

```
          crgData->Failover_Msg_Queue, sizeof(crgData->Failover_Msg_Queue));
  printStr("  Failover_Msg_Queue_Lib = ",
          crgData->Failover_Msg_Queue_Lib,
          sizeof(crgData->Failover_Msg_Queue_Lib));

  /* Print the cluster version.                                         */
  printf("%s %d \n",
         "  Cluster_Version = ", crgData->Cluster_Version);

  /* Print the cluster version mod level                                */
  printf("%s %d \n",
         "  Cluster_Version_Mod_Level = ",
         crgData->Cluster_Version_Mod_Level);

  /* Print the requesting user profile.                                 */
  printStr("  Req_User_Profile = ",
          crgData->Req_User_Profile, sizeof(crgData->Req_User_Profile));

  /* Print the length of the data in the structure.                     */
  printf("%s %d \n",
         "  Length_Info_Returned = ", crgData->Length_Info_Returned);

  /* Print the offset to the recovery domain array.                     */
  printf("%s %d \n",
         "  Offset_Rcvy_Domain_Array = ", crgData->Offset_Rcvy_Domain_Array);

  /* Print the number of nodes in the recovery domain array.            */
  printf("%s %d \n",
         "  Number_Nodes_Rcvy_Domain = ", crgData->Number_Nodes_Rcvy_Domain);

  /* Print the current/new recovery domain.                             */
  printRcvyDomain("   The recovery domain:",
                  crgData->Number_Nodes_Rcvy_Domain,
                  (Qcst_Rcvy_Domain_Array1_t *)
                  ((char *)crgData + crgData->Offset_Rcvy_Domain_Array));

  /* Print the offset to the prior recovery domain array.               */
  printf("%s %d \n",
         "  Offset_Prior_Rcvy_Domain_Array = ",
         crgData->Offset_Prior_Rcvy_Domain_Array);

  /* Print the number of nodes in the prior recovery domain array.      */
  printf("%s %d \n",
         "  Number_Nodes_Prior_Rcvy_Domain = ",
         crgData->Number_Nodes_Prior_Rcvy_Domain);

  /* Print the prior recovery domain if one was passed.                 */
  if (crgData->Offset_Prior_Rcvy_Domain_Array) {
    printRcvyDomain("   The prior recovery domain:",
                    crgData->Number_Nodes_Prior_Rcvy_Domain,
                    (Qcst_Rcvy_Domain_Array1_t *)
                ((char *)crgData + crgData->Offset_Prior_Rcvy_Domain_Array));
  }

  return;
}  /* end printParms                                                  */


/*********************************************************************/
/*                                                                   */
/* Print a string for the action code.                               */
/*                                                                   */
/*********************************************************************/
static void printActionCode(unsigned int ac) {

  char *code;
  switch (ac) {
```

```
    case QcstCrgAcInitialize: code = "QcstCrgAcInitialize";
                              break;
    case QcstCrgAcStart:      code = "QcstCrgAcStart";
                              break;
    case QcstCrgAcRestart:    code = "QcstCrgAcRestart";
                              break;
    case QcstCrgAcEnd:        code = "QcstCrgAcEnd";
                              break;
    case QcstCrgAcDelete:     code = "QcstCrgAcDelete";
                              break;
    case QcstCrgAcReJoin:     code = "QcstCrgAcReJoin";
                              break;
    case QcstCrgAcFailover:   code = "QcstCrgAcFailover";
                              break;
    case QcstCrgAcSwitchover: code = "QcstCrgAcSwitchover";
                              break;
    case QcstCrgAcAddNode:    code = "QcstCrgAcAddNode";
                              break;
    case QcstCrgAcRemoveNode: code = "QcstCrgAcRemoveNode";
                              break;
    case QcstCrgAcChange:     code = "QcstCrgAcChange";
                              break;
    case QcstCrgAcDeleteCommand: code = "QcstCrgAcDeleteCommand";
                              break;
    case QcstCrgAcUndo:       code = "QcstCrgAcUndo";
                              break;
    case QcstCrgEndNode:      code = "QcstCrgEndNode";
                              break;
    case QcstCrgAcAddDevEnt:  code = "QcstCrgAcAddDevEnt";
                              break;
    case QcstCrgAcRmvDevEnt:  code = "QcstCrgAcRmvDevEnt";
                              break;
    case QcstCrgAcChgDevEnt:  code = "QcstCrgAcChgDevEnt";
                              break;
    case QcstCrgAcChgNodeStatus: code = "QcstCrgAcChgNodeStatus";
                              break;
    case QcstCrgAcCancelFailover: code = "QcstCrgAcCancelFailover";
                              break;
    case QcstCrgAcVerificationPhase: code = "QcstCrgAcVerificationPhase";
                              break;
    default:                  code = "unknown action code";
                              break;
  }
  printf("%s", code);

  return;
}  /* end printActionCode                                                  */


/*****************************************************************************/
/*                                                                           */
/* Print the CRG status.                                                     */
/*                                                                           */
/*****************************************************************************/
static void printCrgStatus(int status) {

  char * str;
  switch (status) {
    case QcstCrgActive:               str = "QcstCrgActive";
                                      break;
    case QcstCrgInactive:             str= "QcstCrgInactive";
                                      break;
    case QcstCrgIndoubt:              str = "QcstCrgIndoubt";
                                      break;
    case QcstCrgRestored:             str = "QcstCrgRestored";
                                      break;
    case QcstCrgAddnodePending:       str = "QcstCrgAddnodePending";
```

```
                                    break;
    case QcstCrgDeletePending:      str = "QcstCrgDeletePending";
                                    break;
    case QcstCrgChangePending:      str = "QcstCrgChangePending";
                                    break;
    case QcstCrgEndCrgPending:      str = "QcstCrgEndCrgPending";
                                    break;
    case QcstCrgInitializePending:  str = "QcstCrgInitializePending";
                                    break;
    case QcstCrgRemovenodePending:  str = "QcstCrgRemovenodePending";
                                    break;
    case QcstCrgStartCrgPending:    str = "QcstCrgStartCrgPending";
                                    break;
    case QcstCrgSwitchOverPending:  str = "QcstCrgSwitchOverPending";
                                    break;
    case QcstCrgDeleteCmdPending:   str = "QcstCrgDeleteCmdPending";
                                    break;
    case QcstCrgAddDevEntPending:   str = "QcstCrgAddDevEntPending";
                                    break;
    case QcstCrgRmvDevEntPending:   str = "QcstCrgRmvDevEntPending";
                                    break;
    case QcstCrgChgDevEntPending:   str = "QcstCrgChgDevEntPending";
                                    break;
    case QcstCrgChgNodeStatusPending: str = "QcstCrgChgNodeStatusPending";
                                    break;
    default: str = "unknown CRG status";
  }
  printf("%s \n", str);

  return;
}  /* end printCrgStatus                                                    */


/*************************************************************************/
/*                                                                       */
/* Print the recovery domain.                                            */
/*                                                                       */
/*************************************************************************/
static void printRcvyDomain(char *str,
                            unsigned int count,
                            Qcst_Rcvy_Domain_Array1_t *rd) {

  unsigned int i;
  printf("\n %s \n", str);
  for (i=1; i&lt;=count; i++) {
    printStr("    Node_ID = ", rd->Node_ID, sizeof(Qcst_Node_Id_t));
    printf("%s %d \n", "      Node_Role = ", rd->Node_Role);
    printf("%s", "      Membership_Status = ");
    switch (rd->Membership_Status) {
      case 0: str = "Active";
              break;
      case 1: str = "Inactive";
              break;
      case 2: str = "Partition";
              break;
      default: str = "unknown node status";
    }
    printf("%s \n", str);
    rd++;
  }
  return;
}  /* end printRcvyDomain                                                   */

/*************************************************************************/
/*                                                                       */
/* Concatenate a null terminated string and a non null terminated string */
/* and print it.                                                         */
```

```
/*                                                                    */
/**********************************************************************/
static void printStr(char *s1, char *s2, unsigned int len) {

  char buffer[132];
  memset(buffer, 0x00, sizeof(buffer));
  memcpy(buffer, s1, strlen(s1));
  strncat(buffer, s2, len);
  printf("%s \n", buffer);
  return;
}  /* end printStr                                                    */
```

Configure clusters

IBM(R) and IBM cluster middleware business partners have teamed together to provide state-of-the-art cluster resource services functions along with a graphical user interface (GUI) for cluster management. OS/400(R) cluster resource services provides a set of integrated services that maintain cluster topology, perform heartbeating, and allow creation and administration of cluster configuration and cluster resource groups. Cluster resource services also provides reliable messaging functions that keep track of each node in the cluster and ensures that all nodes have consistent information about the state of cluster resources. In addition, cluster resource services provides a set of control language (CL) commands and application program interfaces (APIs) and facilities that can be used by iSeries(TM) application providers or customers to enhance their application availability. Cluster resource services functions can also be accessed via graphical user interface solutions provided by iSeries Navigator cluster management and cluster middleware business partner products.

Getting started
Follow these steps to configure a cluster:

1. **Select a software solution.**
   See "Solutions for configuring and managing clusters" on page 23 for complete look at the options for configuring and managing clusters.
2. **Satisfy the hardware, software and communications requirements.**
   Review the cluster requirements in "Plan for clusters" on page 23.
3. **Set up your network and server environment for clusters.**
   Use the "Cluster configuration checklist" on page 35 to make sure that you are prepared to configure clusters in your environment.
4. **Configure your cluster.**
   See "Create a cluster" for details.

If you need help during the configuration process, see "Who to call for cluster suppport" on page 104 for a number that you can call.

Create a cluster

Before you attempt to create a cluster, see the "Cluster configuration checklist" on page 35 for details on setting up your environment for clusters.

To create and configure a cluster, you need to include at least one node in the cluster and you must have access to at least one of the nodes that will be in the cluster. If only one node is specified, it must be the server that you are currently accessing. If you will be creating a cluster consisting of nodes at different cluster version levels, see "Multiple-release clusters" on page 30 before creating your cluster.

If you will be using switchable devices in your cluster, there are additional requirements than that of a cluster that does not use switchable devices. To set up a cluster environment that includes switchable devices, care must be taken so that conflicts are avoided across the cluster. See Create a switchable independent disk pool for step-by-step instructions on how to configure a cluster to use switchable devices.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

"iSeries Navigator cluster management" on page 24 features a wizard that takes you through the steps to create and start a simple cluster consisting of one or two cluster nodes. Once you have created a one- or two-node cluster, you can add nodes to it. A cluster created and managed in iSeries Navigator can contain as many as four nodes. This wizard will guide you through the steps to specify servers to include and create cluster resource groups. When you are creating a simple cluster, the server you are creating the cluster on must be one of the nodes.

To create a simple cluster using the New Cluster wizard in iSeries Navigator, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Right-click **Clusters**, and select **New Cluster..**
3. Follow the wizard's instructions to create a cluster.

Once you have created the cluster, be sure to:

1. Add all of the nodes that you would like to include in the cluster. Up to four nodes can be added to a cluster created and managed in iSeries Navigator.
2. Add desired nodes to device domains (for use with switchable hardware groups and independent disk pools).
3. Create and start the switchable resources (switchable hardware, switchable software, and switchable data).

The online help in iSeries Navigator contains step-by-step procedures on completing these tasks.

Using CL commands and APIs

You can also use CL commands or APIs to create a cluster:

1. **Create the cluster.**
   Create Cluster (CRTCLU) command
   Create Cluster (QcstCreateCluster) API
2. **Add nodes to your cluster from the active cluster node.**
   Add Cluster Node Entry (ADDCLUNODE) command
   Add Cluster Node Entry (QcstAddClusterNodeEntry) API
3. **Define device domains.**
   If you plan to use switchable devices, you must include the desired nodes in the device domain.
   Add Device Domain Entry (ADDDEVDMNE) command
   Add Device Domain Entry (QcstAddDeviceDomainEntry) API
4. **Create cluster resource groups (CRG).**
   Create Cluster Resource Group (CRTCRG) command
   Create Cluster Resource Group (QcstCreateClusterResourceGroup) API
5. **Start the cluster resource groups (CRG).**
   Start Cluster Resource Group (STRCRG) command
   Start Cluster Resource Group (QcstStartClusterResourceGroup) API

Manage clusters

This topic contains information that covers some of the tasks that involve managing your clusters. If you have not considered the type of interface you are going to use to manage your clusters, see "Solutions for configuring and managing clusters" on page 23 before going any further.

Some of the changes that you can make to the cluster once you have configured it include the following:

Cluster tasks
- "Add a node to a cluster"
- Remove nodes from a cluster
- "Start a cluster node"
- End a cluster node
- "Adjust the cluster version of a cluster" on page 82 to the latest level
- "Delete a cluster" on page 81

Cluster resource group tasks
- Create new cluster resource groups
- Delete existing cluster resource groups
- Start a cluster resource group
- End a cluster resource group
- "Change the recovery domain for a cluster resource group" on page 83
- "Perform a switchover" on page 83
- "Add a node to a device domain" on page 84
- "Remove a node from a device domain" on page 85

This topic will also help you to "Save cluster configurations" on page 89. You may want to read about how "Job structure and user queues" on page 87 are structured and how cluster APIs use user queues. Read about the correct way to "End cluster jobs" on page 87 and how to "Monitor cluster status" on page 85. You also learn how the "Reliable message function" on page 21 and "Heartbeat monitoring" on page 19 keep you updated on the status of your cluster.

Add a node to a cluster

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

The simple cluster supported by iSeries(TM) Navigator can be made up of a maximum of four nodes. If four nodes already exist in the cluster, the **Add Node...** option is disabled. If your clustering needs extend beyond four nodes, you should use "Cluster commands and APIs" on page 25 or a "Cluster middleware business partners and available clustering products" on page 25 to support up to 128 nodes.

To add a node to an existing cluster, follow these steps:
1. In iSeries Navigator, expand Management Central.
2. Expand **Clusters**.
3. Expand the cluster for which you would like to add a node.
4. Right-click **Nodes**, and select **Add Node...**

Using Cluster commands and APIs

You can also use the following to add a node to a cluster:
- Add Cluster Node Entry (ADDCLUNODE) command
- Add Cluster Node Entry (QcstAddClusterNodeEntry) API

Start a cluster node

Starting a cluster node starts cluster resource services on a node in the cluster. Beginning with cluster version 3, a node can start itself and will be able to rejoin the current active cluster, provided it can find an active node in the cluster.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

When cluster resource services is successfully started on the node specified, the status of the node will be set to *Started*.

To start clustering on a node, follow these steps:
1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Expand the cluster that contains the node on which you would like to start clustering.
4. Click **Nodes**.
5. Right-click the node you would like to start clustering on, and select **Cluster > Start...**

Using CL commands and APIs

You can also use CL commands or APIs to start a node. When cluster resource services is successfully started on the node specified, the status of the node will be set to *Active*.
- Start Cluster Node (STRCLUNOD) command
- Start Cluster Node (QcstStartClusterNode) API

Delete a cluster

When you delete a cluster, cluster resource services will be ended on all active cluster nodes and they will be removed from the cluster.

Important: If you have independent disk pools in your cluster, you should first remove each node from the device domain using the Remove Device Domain Entry (RMVDEVDMNE) command before you delete your cluster.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To delete a cluster, follow these steps:
1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Right-click the cluster you would like to delete, and select **Delete...**

Using CL commands and APIs

You can also use CL commands or APIs to delete a cluster.
- Delete Cluster (DLTCLU) command
- Delete Cluster (QcstDeleteCluster) API

Remove a cluster node

You can remove a node to a cluster using iSeries(TM) Navigator or commands.

Using iSeries(TM) Navigator

This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To adjust the cluster version of a cluster, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Expand the cluster from which you would like to remove a node.
4. right-click **Nodes**, and select **Cluster—> Remove**.

Using Cluster commands and APIs

You can also use the following to remove a node from a cluster:

- Remove Cluster Node Entry (RMVCLUNODE) command
- Remove Cluster Node Entry (QcstRemoveClusterNodeEntry) API

Adjust the cluster version of a cluster

The "Cluster version" on page 10 defines the level at which all the nodes in the cluster are actively communicating with each other. Cluster versioning is a technique that allows the cluster to contain systems at multiple release levels and fully interoperate by determining the communications protocol level to be used.

To change the cluster version, all nodes in the cluster must be at the same potential version. The cluster version can then be changed to match the potential version. This will allow the new function to be used. The version can only be incremented by one. It cannot be decremented without deleting the cluster and recreating it at the lower version. The current cluster version is initially set by the first node defined in the cluster. Subsequent nodes added to the cluster must be equal to the current cluster version or the next level version, otherwise they cannot be added to the cluster.

If you are upgrading a node to a new release, you must ensure that the node has the appropriate cluster version. Clusters only supports a one version difference. If all the nodes in the cluster are at the same release, you should upgrade to the new release, before changing the cluster version. This ensures that all function associated with the new release will be available. See the topic, Cluster version for detailed actions for upgrading to a new release.

Use the following instructions to both verify and change the cluster version for a node.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To verify or change the cluster version of a cluster, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Right-click the cluster, and select **Properties**.
4. Verify the cluster version setting or change the version to the desired setting.

Using Cluster commands and APIs

You can also use the following to adjust the cluster version of a cluster:

- Change Cluster Version (CHGCLUVER) command
- Adjust Cluster Version (QcstAdjustClusterVersion) API

Change the recovery domain for a cluster resource group

You can change the roles of nodes in a "Recovery domain" on page 8 for a cluster resource group, as well as add or remove nodes from a recovery domain. For a device cluster resource group, you can also change site name and data port IP addresses for a node in the recovery domain.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To change the role of nodes in the recovery domain for a cluster resource group (switchable hardware, switchable software, or switchable data), or to add or remove nodes to the recovery domain, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Expand the cluster containing the switchable hardware, software, or data for which you would like to change the recovery domain.
4. Expand the switchable hardware, software, or data.
5. Right-click the switchable hardware, software, or data, and select **Properties**.
6. Select the **Recovery Domain** page.

Click Help on the Recovery Domain page for instructions on how to change roles or add or remove nodes.

Using CL commands and APIs

To change the role of nodes in the recovery domain, or add or remove nodes, use the following CL commands and APIs:

Function	CL command	API
Add node to recovery domain	Add Cluster Resource Group Node Entry (ADDCRGNODE)	QcstAddNodeToRcvyDomain
Remove node from recovery domain	Remove Cluster Resource Group Node Entry (RMVCRGNODE)	QcstRemoveNodeFromRcvyDomain
Change cluster resource group	Change Cluster Resource Group (CHGCRG)	QcstChangeClusterResourceGroup

Perform a switchover

Performing a manual "Switchover" on page 17 causes the current primary node to switch over to the backup node, as defined in the cluster resource group's "Recovery domain" on page 8. When this happens, the current roles of the nodes in the recovery domain of a cluster resource group change such that:

- The current primary node is assigned the role of last active backup.
- The current first backup is assigned the role of primary.
- Subsequent backups are moved up one in the order of backups.

A switchover is only allowed on CRGs that have a status of ACTIVE.

Note: If you are performing a switchover on a switchable hardware group (also known as a device CRG), you should Synchronize user profile name, UID, and GID for performance reasons.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To switch a resource - a switchable hardware group, switchable software product, or switchable data group - from the primary node to the backup node in the recovery domain, the resource must have a status of **Started**.

To perform a switchover on a resource, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Expand the cluster containing the desired resource.
4. Click **Switchable Hardware**, **Switchable Software**, or **Switchable Data**.
5. Right-click the desired resource, and select **Switch...**

Using Cluster APIs

You can also use the following to perform a switchover:

- Change Cluster Resource Group Primary (CHGCRGPRI) command
- Initiate Switchover (QcstInitiateSwitchOver) API

Add a node to a device domain

A "Device domains" on page 12 is a subset of nodes in a cluster that share device resources. Before a node can be added to the recovery domain for a device cluster resource group (CRG), the node must be first defined as a member of a device domain. All nodes that will be in the recovery domain for a device CRG must be in the same device domain. A cluster node can belong to at most one device domain.

To create and manage device domains, you must have Option 41 (OS/400 - HA Switchable Resources) installed and a valid license key must exist on all cluster nodes that will be in the device domain.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To add a node to a device domain in iSeries Navigator, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Expand the cluster that contains the node you would like to add to the device domain.
4. Click **Nodes**.
5. Right-click the node you would like to add to the device domain, and select **Properties**.
6. On the **Clustering** page, specify the name of the device domain you would like to add the node to in the **Device domain** field.

Using CL commands and APIs

You can also use the following to add a node to a device domain:

- Add Device Domain Entry (ADDDEVDMNE) command
- Add Device Domain Entry (QcstAddDeviceDomainEntry) API

Remove a node from a device domain

A "Device domains" on page 12 is a subset of nodes in a cluster that share device resources.

Important

Be cautious when removing a node from a device domain. If you remove a node from a device domain, and that node is the current primary point of access for any independent disk pools, those independent disk pools remain with the node being removed. This means that those independent disk pools will no longer be accessible from the remaining nodes in the device domain.

Once a node is removed from a device domain, it cannot be added back to the same device domain if one or more of the existing cluster nodes still belong to that same device domain. In order to add the node back to the device domain, you must:

1. Delete the independent disk pools currently owned by the node being added to the device domain.
2. Perform a system restart (IPL) on the node.
3. Add the node to the device domain. See "Add a node to a device domain" on page 84.
4. Recreate the independent disk pools deleted in Step 1. See Adding a disk unit or disk pool.

Using iSeries(TM) Navigator

This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To remove a node from a device domain in iSeries Navigator, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Expand the cluster that contains the node you would like to remove from the device domain.
4. Click **Nodes**.
5. Right-click the node you would like to remove from the device domain, and select **Properties**.
6. On the Clustering page, remove the entry in the **Device domain** field.

Using CL commands and APIs

You can also use the following to remove a node from a device domain:

- Remove Device Domain Entry (RMVDEVDMNE) command
- Remove Device Domain Entry (QcstRemoveDeviceDomainEntry) API

Monitor cluster status

Cluster resource services performs basic monitoring of a cluster and its components using the "Reliable message function" on page 21 and "Heartbeat monitoring" on page 19, taking appropriate actions when necessary.

You can also manually monitor the status of a cluster and its components.

Using iSeries(TM) Navigator

This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

To monitor the status of a cluster in iSeries Navigator:

1. In iSeries Navigator, expand Management Central.
2. Expand **Clusters**.
3. Navigate within the iSeries Navigator folders for the desired cluster to view the status of the cluster, its nodes, and resources using the Status column in the iSeries Navigator list. The online help contains

descriptions of the possible values for the Status column. You can also right-click on components of the cluster and select **Properties** to view information about the cluster.

Using CL commands and APIs

You can use the following commands and APIs to monitor cluster status:

Cluster Information
Retrieves information about a cluster, such as the nodes in the cluster, which adapter IP addresses are being used on each node, and the status of each node in the cluster.

- Display Cluster Information (DSPCLUINF) command
- List Cluster Information (QcstListClusterInfo) API
- List Device Domain Info (QcstListDeviceDomainInfo) API
- Retrieve Cluster Resource Services Information (QcstRetrieveCRSInfo) API
- Retrieve Cluster Information (QcstRetrieveClusterInfo) API

Cluster Resource Group information
Generates a list of cluster resource groups and information about the cluster resource group in the cluster, such as the name of the primary node of each CRG in the cluster.

- Display Cluster Resource Group Information (DSPCRGINF) command
- List Cluster Resource Groups (QcstListClusterResourceGroups) API
- List Cluster Resource Group Information (QcstListClusterResourceGroupInf) API

Cluster performance

When changes are made to a cluster, the overhead necessary to manage the cluster can be affected. The only resources that clustering requires are those necessary to perform "Heartbeat monitoring" on page 19, to manage the cluster resource groups and the cluster nodes, and to handle any messaging taking place between cluster resource groups and cluster nodes. Once your clustering environment is operational, the only increase in overhead will be if you make changes to the cluster.

During a normal operating environment, there should be minimal impact to your clustered systems due to clustering activity.

To get the optimal performance out of your clustered servers review these topics:

- "Balance network load for clusters"
- "Tune cluster performance"

Balance network load for clusters

You can balance the network load by splitting your work between the communication lines that you use to connect the nodes in the cluster. The more you can balance the work to keep resource utilization low, the more smoothly your system will run.

See CPU load on backup nodes for more information on keeping your backup systems running smoothly.

Tune cluster performance

Since potentially significant differences exist in your communications environment, you have the capability to adjust variables that affect cluster communications to best match your environment. The default values should normally be acceptable to most common environments. If your particular environment is not well suited for these defaults, you can tune cluster communications to better match your environment. Two levels of tuning are available.

Base-level tuning allows you to set the tuning parameters to a predefined set of values identified for high, low, and normal timeout and messaging interval values. When the normal level is selected, the default values are used for cluster communications performance and configuration parameters. Selecting the low level causes clustering to increase the heartbeating interval and the various message timeout values. With fewer heartbeats and longer timeout values, the cluster will be less sensitive to communications failures. Selecting the high level causes clustering to decrease the heartbeating interval and the various message timeout values. With more frequent heartbeats and shorter timeout values, the cluster will be more sensitive to communications failures.

Advanced-level tuning is also available such that individual parameters may be tuned over predefined ranges of values. This allows more granular tuning to meet any special circumstances in your communications environment. If an advanced level of tuning is desired, it is recommended that you obtain help from IBM(R) support personnel or equivalent. Setting the individual parameters incorrectly could easily result in decreased performance.

See Tunable cluster communications parameters and the Change Cluster Resource Services (QcstChgClusterResourceServices) API for more information on the specific parameters available and the allowed values.

End cluster jobs

You should never try to end a cluster job directly. If you need to stop whatever may be running in a clustered environment, you should:

1. End the cluster node.
2. Fix the problem.
3. "Start a cluster node" on page 80.

Job structure and user queues

Cluster resource services job structure

Cluster resource services consists of a set of multi-threaded jobs. When clustering is active on a server, the following jobs run in the QSYSWRK subsystem under the QSYS user profile. The jobs run using the QDFTJOBD job description, but with the logging level set such that a job log will be produced.

- Cluster control consists of one job that is named QCSTCTL.
- Cluster resource group manager consists of one job that is named QCSTCRGM.
- Cluster resource groups consist of one job per cluster resource group object. The job name is the same as the cluster resource group name.
- When one or more device list entries in a resilient device CRG have been set to be brought online at switchover or failover, additional jobs will be submitted to perform the vary on function.

The QCSTCTL and QCSTCRGM job are cluster critical jobs. That is, the jobs must be running in order for the node to be active in the cluster.

Most cluster resource group APIs result in a separate job being submitted that use the user profile specified when the cluster resource group was created. The exit program defined in the cluster resource group is called in the submitted job. By default, the jobs are submitted to the QBATCH job queue. Generally, this job queue is used for production batch jobs and will delay or prevent completion of the exit programs. To allow the APIs to run effectively, create a separate user profile, job description, and job queue for use by cluster resource groups. Specify the new user profile for all cluster resource groups that you create. The same program is processed on all nodes within the recovery domain that is defined for the cluster resource group.

Cluster APIs use of user queues

Functions performed by an API that has a Results information parameter operate asynchronously and send their results to a user queue once the API is finished processing. The user queue must be created before calling the API. You can create a user queue by using the Create User Queue (QUSCRTUQ) API. The queue must be created as a keyed queue. The key for the user queue is described in the format of the user queue entry. The user queue name is passed to the API. For more information on user queues, see Cluster APIs Use of User Queues.

When the Distribute Information (QcstDistributeInformation) API is used, the information sent between nodes is deposited on the user queue specified when the CRG was created. This queue must be created by the user on all active nodes in the recovery domain prior to the use of the Distribute Information API. See the Create Cluster (CRTCLU) command and the Create Cluster Resource Group (QcstCreateClusterResourceGroup) API for details on when the distribute information queue must exist.

The failover message queue receives messages regarding failover activity. See Failover message queue for details.

Backup and recovery of clusters

If you implement clustering on your systems, it is still important that you create a backup and recovery strategy to protect your data. If you are not familiar with why you would want a strategy and how to go about creating one, see Planning a backup and recovery strategy.

If you are planning on using clustering as your backup strategy so that you have one system up and running while the other system is down when its being backed up, it is recommended that you have a minimum of three systems in the cluster. By having three systems in the cluster, you will always have one system to switch over to should a failure occur.

For more information on backup and recovery procedures, see “Restore a cluster from backup tapes” on page 98.

Saving and restoring cluster resource groups

You can save a cluster resource group whether the cluster is active or inactive. The following restrictions apply for restoring a cluster resource group:

- If the cluster is up and the cluster resource group is known to that cluster, you cannot restore the cluster resource group.
- If the node is not configured for a cluster, you cannot restore a cluster resource group.

You can restore a cluster resource group if the cluster is active, the cluster resource group is not known to that cluster, the node is in the recovery domain of that cluster resource group, and the cluster name matches that in the cluster resource group. You can restore a cluster resource group if the cluster is configured but is not active on that node and if that node is in the recovery domain of that cluster resource group.

Preparing for a disaster

In the case of a disaster, you will need to reconfigure your cluster. In order to prepare for such a scenario, it is recommended that you save your cluster configuration information and keep a hardcopy printout of that information.

1. Use the Save Configuration (SAVCFG) command or the Save System (SAVSYS) command after making cluster configuration changes so that the restored internal cluster information is current and consistent with other nodes in the cluster. See Saving configuration information for details on performing a SAVCFG or SAVSYS.

2. Print a copy of cluster configuration information every time you change it. See Printing system information for details. Keep a copy with your backup tapes to use in case of a disaster where you would need to reconfigure your entire cluster.

For recovery information, see:

- "Recover a cluster after a complete system loss" on page 97

Save cluster configurations

You can use the SAVSYS (Save System) Command which saves your entire system, not just your configured cluster. You can use the SAVCFG (Save Configuration) Command to save your configured system.

You can use the following commands to save your cluster resource group objects:

- SAVOBJ(QUSRSYS/*ALL) OBJTYPE (*CRG)

Note: Cluster resource group objects can be saved only for the current release.

See "Backup and recovery of clusters" on page 88 for save and restore considerations for cluster resource groups.

Examples: Cluster configurations

Use these example cluster configurations to understand the possibilities when planning and implementing clusters in your environment.

The following configurations are examples of some common cluster implementations:

- Example: A simple, two-node cluster
- Example: A four-node cluster
- Example: A switched-disk cluster using independent disk pools
- 

  Example: Independent disk pools with geographic mirroring

Troubleshoot clusters

At times, it may appear that the cluster is not working properly. This topic covers information about problems that you may encounter with clusters.

"Determine if a cluster problem exists" on page 90
Start here to diagnose your cluster problems.

"Common cluster problems" on page 91
This topic lists some of the most common problems that can occur in a cluster, as well as ways to avoid and recover from them.

"Partition errors" on page 93
Certain cluster conditions are easily corrected. If a cluster partition has occured, you can learn how to recover. This topic also tells you how to avoid a cluster partition and gives you an example of how to merge partitions back together.

"Cluster recovery" on page 96
Read about how to recover from other cluster failures that may occur.

"Frequently asked questions about iSeries Navigator cluster management" on page 98
Questions and answers about the iSeries(TM) Navigator graphical user interface for creating and managing clusters.

"Who to call for cluster suppport" on page 104
See this topic if you need to contact IBM(R) with your cluster questions.

Determine if a cluster problem exists

At times, it may seem that your cluster is not operating correctly. When you think a problem exists, you can use the following to help determine if a problem exists and the nature of the problem.

- **Determine if clustering is active on your system.**
  To determine if cluster resource services is active, look for the two jobs - QCSTCTL and QCSTCRGM - in the QSYSWRK subsystem. If these jobs are active, then cluster resource services is active. You can use the Work Management function in iSeries(TM) Navigator to View jobs in a subsystem or use the WRKACTJOB (Work with Active Jobs) CL command to do this. You can also use the DSPCLUINF (Display Cluster Information) command to view status information for the cluster.
  - Additional jobs for cluster resource services may also be active. See "Job structure and user queues" on page 87 for details.
- **Look for messages indicating a problem.**
  - Look for inquiry messages in QSYSOPR that are waiting for a response.
  - Look for error messages in QSYSOPR that indicate a cluster problem. Generally, these will be in the CPFBB00 to CPFBBFF range.
  - Display the history log (DSPLOG CL command) for messages that indicate a cluster problem. Generally, these will be in the CPFBB00 to CPFBBFF range.
- **Look at job logs for the "Job structure and user queues" on page 87 for severe errors.**
  These jobs are initially set with a logging level at (4 0 *SECLVL) so that you can see the necessary error messages. You should ensure that these jobs and the exit program jobs have the logging level set appropriately. If clustering is not active, you can still look for spool files for the cluster jobs and exit program jobs.
- **If you suspect some kind of hang condition, look at call stacks of cluster jobs.**
  Determine if there is any program in some kind of DEQW (dequeue wait). If so, check the call stack of each thread and see if any of them have getSpecialMsg in the call stack.
- **Check for cluster vertical licensed internal code (VLIC) logs entries.**
  These log entries would have a 4800 major code.
- **Use NETSTAT command to determine if there are any abnormalities in your communications environment.**
  NETSTAT returns information about the status of TCP/IP network routes, interfaces, TCP connections and UDP ports on your system.
  - Use Netstat Option 1 (Work with TCP/IP interface status) to ensure that the IP addresses chosen to be used for clustering show an 'Active' status. Also ensure that the LOOPBACK address (127.0.0.1) is also active.
  - Use Netstat Option 3 (Work with TCP/IP Connection Status) to display the port numbers (F14). Local port 5550 should be in a 'Listen' state. This port must be opened via the STRTCPSVR *INETD command evidenced by the existence of a QTOGINTD (User QTCP) job in the Active Jobs list. If clustering is started on a node, local port 5551 must be opened and be in a '*UDP' state. If clustering is not started, port 5551 must not be opened or it will, in fact, prevent the successful start of clustering on the subject node.
- **Use the CLUSTERINFO macro to show cluster resource services' view of nodes in the cluster, nodes in the various cluster resource groups, and cluster IP addresses being currently used.**

Discrepencies found here could help pinpoint trouble areas if the cluster is not performing as expected. The CLUSTERINFO macro can be invoked from System Service Tools (SST) via the STRSST command as follows:

– SST Option 1 Start a service tool
– Start Option 4 Display/Alter/Dump
– Display/Alter Option 1 Display/Alter storage
– Select Data Option 2 Licensed Internal Code Data
– Select LIC Data Option 14 Advanced Analysis
– Select CLUSTERINFO macro (-h option for parameters and more information)

Common cluster problems

The following common problems are easily avoidable or easily correctable.

You cannot start or restart a cluster node.

This situation is typically due to some problem with your communications environment. To avoid this situation, ensure that your network attributes are set correctly, including the loopback address, INETD settings, ALWADDCLU attibute, and the IP addresses for cluster communications.

- The ALWADDCLU network attribute must be appropriately set on the target node if trying to start a remote node. See "Enable a node to be added to a cluster" on page 34 for details on setting this attribute. This should be set to either *ANY or *RQSAUT depending on your environment.
- The IP addresses chosen to be used for clustering locally and on the target node must show an 'Active' status.
- The LOOPBACK address (127.0.0.1) locally and on the target node must also be active.
- The local and any remote nodes must be able to PING using the IP addresses to be used for clustering to insure network routing is active.
- INETD must be active on the target node. When INETD is active, port 5550 on the target node should be in a 'Listen' state. See INETD server for information on starting the INETD server.
- Prior to attempting to start a node, port 5551 on the node to be started must not be opened or it will, in fact, prevent the successful start of clustering on the subject node.

You end up with several, disjointed one-node clusters.

This can occur when the node being started cannot communicate with the rest of the cluster nodes. Check the communications paths.

The response from exit programs is slow.

A common cause for this situation is incorrect setting for the job description used by the exit program. The MAXACT parameter may be set too low so that, for example, only one instance of the exit program can be active at any point in time. It is recommended that this be set to *NOMAX.

Performance in general seems to be slow.

There are several common causes for this symptom.

- The most likely cause is heavy communications traffic over a shared communications line. See "Cluster performance" on page 86 for more information.
- Another likely cause is an inconsistency between the communications environment and the cluster message tuning parameters. You can use the Retrieve Cluster Resource Services Information (QcstRetrieveCRSInfo) API to view the current settings of the tuning parameters and the Change Cluster Resource Services (QcstChgClusterResourceServices) API to change the

settings. Cluster performance may be degraded under default cluster tuning parameter settings if using old adapter hardware. The adapter hardware types included in the definition of 'old' are 2617, 2618, 2619, 2626, and 2665. In this case, setting of the 'Performance class' tuning parameter to 'Normal' is desired.

- Another common cause of this condition is problems with the IP multicast groups. If the primary cluster addresses (first address entered for a given node when creating a cluster or adding a node) for several nodes reside on a common LAN, the cluster will utilize IP multicast capability. Using the NETSTAT command, insure the primary cluster addresses show a multicast host group of '226.5.5.5'. This can be seen using option 14 'Display multicast group' for the subject address. If the multicast group does not exist, verify the default setting of TRUE is still set for the 'Enable multicast' cluster tuning parameter by using the Retrieve Cluster Resource Services Information (QcstRetrieveCRSInfo) API.
- If all the nodes of a cluster are on a local LAN or have routing capabilities which can handle Maximum Transmission Unit (MTU) packet sizes of greater than 1,464 bytes throughout the network routes, large cluster message transfers (greater than 1,536K bytes) can be greatly speeded up by increasing the cluster tuning parameter value for 'Message fragment size' to better match the route MTUs.

You cannot use any of the function of the new release.

If you attempt to use new release function and you see error message CPFBB70, then your current "Cluster version" on page 10 is still set at the prior version level. You must upgrade all cluster nodes to the new release level and then use the adjust cluster version interface to set the current cluster version to the new level. See "Adjust the cluster version of a cluster" on page 82 for more information.

You cannot add a node to a device domain or access the iSeries(TM) Navigator cluster management interface.

To access the "iSeries Navigator cluster management" on page 24 interface, or to use switchable devices, you must have OS/400(R) Option 41, HA Switchable Resources installed on your system. You must also have a valid license key for this option.

You applied a cluster PTF and it does not seem to be working.

Did you signoff/signon? The old program is still active in the activation group until the activation group is destroyed. All of the cluster code (even the cluster APIs) run in the default activation group.

CEE0200 appears in the exit program joblog.

On this error message, the from module is QLEPM and the from procedure is Q_LE_leBdyPeilog. Any program that the exit program invokes must run in either *CALLER or a named activation group. You must change your exit program or the program in error to correct this condition.

CPD000D followed by CPF0001 appears in the cluster resource services joblog.

When you receive this error message, make sure the system value for QMLTTHDACN is set to either 1 or 2.

Cluster appears hung.

Make sure cluster resource group exit programs are outstanding. To check the exit program, use the WRKACTJOB (Work with Active Jobs) command, then look in the Function column for the presence of PGM-QCSTCRGEXT.

Partition errors

A "Cluster partition" on page 22 occurs in a cluster whenever contact is lost between one or more nodes in the cluster and a failure of the lost nodes cannot be confirmed. This is not to be confused with a partition in a logical partition (LPAR) environment.

If you receive error message CPFBB20 in either the history log (QHST) or the QCSTCTL joblog, a cluster partition has occurred and you need to know how to recover. The following example shows a cluster partition that involves a cluster made up of four nodes: A, B, C, and D. The example shows a loss of communication between cluster nodes B and C has occurred, which results in the cluster dividing into two cluster partitions. Before the cluster partition occurred, there were four cluster resource groups, which could be of any type, called CRG A, CRG B, CRG C, and CRG D. The example shows the recovery domain of each cluster resource group.

<table>
<tr><th>Node A</th><th>Node B</th><th></th><th>Node C</th><th>Node D</th></tr>
<tr><td>CRG A (backup1)</td><td>CRG A (primary)</td><td rowspan="4">x</td><td></td><td></td></tr>
<tr><td></td><td>CRG B (primary)</td><td>CRG B (backup1)</td><td></td></tr>
<tr><td></td><td>CRG C (primary)</td><td>CRG C (backup1)</td><td>CRG C (backup2)</td></tr>
<tr><td>CRG D (backup2)</td><td>CRG D (primary)</td><td>CRG D (backup1)</td><td></td></tr>
<tr><td colspan="2">Partition 1</td><td></td><td colspan="2">Partition 2</td></tr>
</table>

Using this example, read how to "Determine primary and secondary cluster partitions" to see what kinds of cluster resource group actions you can take.

A cluster may partition if the maximum transmission unit (MTU) at any point in the communication path is less than the cluster communications tuneable parameter, message fragment size. MTU for a cluster IP address can be verified using the Work with TCP/IP Network Status (WRKTCPSTS) command on the subject node. The MTU must also be verified at each step along the entire communication path. If the MTU is less than the message fragment size, either raise the MTU of the path or lower the message fragment size. You can use the Retrieve Cluster Resource Services Information (QcstRetrieveCRSInfo) API to view the current settings of the tuning parameters and the Change Cluster Resource Services (QcstChgClusterResourceServices) API to change the settings.

Once the cause of the cluster partition condition has been corrected, the cluster will detect the re-established communication link and issue the message CPFBB21 in either the history log (QHST) or the QCSTCTL joblog. This informs the operator that the cluster has recovered from the cluster partition. Be aware that once the cluster partition condition has been corrected, it may be a few minutes before the cluster merges back together.

If the partition condition reported is really a failure condition of one or more nodes, see "Change partitioned nodes to failed" on page 94.

For more information on troubleshooting a cluster partition, see:

- "Avoid a cluster partition" on page 29
- "Tips: Cluster partitions" on page 95
- "Merge" on page 19
- "Example: Failure" on page 15

Determine primary and secondary cluster partitions

In order to determine the types of cluster resource group actions that you can take within a cluster partition, you need to know whether the partition is a primary or a secondary cluster partition. The

cluster partition that contains the current primary node in the recovery domain of a cluster resource group is considered the primary partition of the cluster resource group. All other partitions are secondary partitions. The primary partitions may not be the same for all cluster resource groups. The restrictions for each Cluster Resource Group API are:

Table 1. Cluster Resource Group API Partition Restrictions

Cluster Resource Group API	Allowed in primary partition	Allowed in secondary partitions
Add Node to Recovery Domain	X	
Add CRG Device Entry		
Change Cluster Resource Group	X	
Change CRG Device Entry	X	X
Create Cluster Resource Group		
Delete Cluster Resource Group	X	X
Distribute Information	X	X
End Cluster Resource Group	X	
Initiate Switchover	X	
List Cluster Resource Groups	X	X
List Cluster Resource Group Information	X	X
Remove Node from Recovery Domain	X	
Remove CRG Device Entry	X	
Start Cluster Resource Group	X	

By applying these restrictions, cluster resource groups can be resynchronized when the cluster is no longer partitioned. As nodes rejoin the cluster from a partitioned status, the version of the cluster resource group in the primary partition is copied to nodes from a secondary partition.

When a partition is detected, the Add Cluster Node Entry, Adjust Cluster Version, and the Create Cluster API cannot be run in any of the partitions. The Add Device Domain Entry API can only be run if none of the nodes in the device domain are partitioned. All of the other Cluster Control APIs may be run in any partition. However, the action performed by the API takes affect only in the partition running the API.

Change partitioned nodes to failed

Sometimes, a partitioned condition is reported when there really was a node outage. This can occur when cluster resource services loses communications with one or more nodes, but cannot detect if the nodes are still operational. When this condition occurs, a simple mechanism exists for you to indicate that the node has failed.

Attention: When you tell cluster resource services that a node has failed, it makes recovery from the partition state simpler. However, changing the node status to failed when, in fact, the node is still active and a true partition has occurred should not be done. Doing so could cause a node in more than one partition to assume the primary role for a cluster resource group. When two nodes think they are the primary node, data such as files or databases could become disjoint or corrupted if multiple nodes are each independently making changes to their copies of files. In addition, the two partitions cannot be “Merge” on page 19 back together when a node in each partition has been assigned the primary role.

When the status of a node is changed to Failed, the role of nodes in the recovery domain for each cluster resource group in the partition may be reordered. The node being set to Failed will be assigned as the last backup. If multiple nodes have failed and their status needs to be changed, the order in which the

nodes are changed will affect the final order of the recovery domain's backup nodes. If the failed node was the primary node for a CRG, the first active backup will be reassigned as the new primary node.

Using iSeries(TM) Navigator
This requires Option 41 (OS/400 - HA Switchable Resources) to be installed and licensed.

When cluster resource services has lost communications with a node but cannot detect if the node is still operational, a cluster node will have a status of **Not communicating** in the Nodes container in iSeries Navigator. You may need to change the status of the node from **Not communicating** to **Failed**. You will then be able to restart the node.

To change the status of a node from **Not communicating** to **Failed**, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Expand the cluster that contains the node for which you would like to change the status.
4. Click **Nodes**.
5. Right-click the node for which you would like to change the status, and select **Cluster** > **Change Status...**

To restart the node, follow these steps:

1. Right-click the node, and select **Cluster** > **Start...**

Using CL commands and APIs

To change the status of a node from **Not communicating** to **Failed**, follow these steps:

1. Use the CHGCLUNODE command or the Change Cluster Node Entry API (QcstChangeClusterNodeEntry) to change the status of a node from partitioned to failed. This should be done for all nodes that have actually failed.
2. Use the STRCLUNOD command or the Start Cluster Node (QcstStartClusterNode) API to start the cluster node, allowing the node to "Rejoin" on page 17 the cluster.

Tips: Cluster partitions

1. The rules for restricting operations within a partition are designed to make merging the partitions feasible. Without these restrictions, reconstructing the cluster would require extensive work by you.
2. If the nodes in the primary partition have been destroyed, special processing may be necessary in a secondary partition. The most common scenario that causes this condition would be the loss of the site that made up the primary partition. Use the example in "Partition errors" on page 93 and assume that Partition 1 was destroyed. In this case, the primary node for Cluster Resource Groups B, C, and D must be located in Partition 2. The simplest recovery is to use Change Cluster Node Entry to set both Node A and Node B to failed. See "Change partitioned nodes to failed" on page 94 for more information on how to do this.
   Recovery can also be achieved manually. In order to do this, perform these operations:
   a. Remove Nodes A and B from the cluster in Partition 2. Partition 2 is now the cluster.
   b. Establish any replication environments needed in the new cluster. IE. Start Cluster Resource Group API/CL command, etc.

   Since nodes have been removed from the cluster definition in Partition 2, an attempt to merge Partition 1 and Partition 2 will fail. In order to correct the mismatch in cluster definitions, run the Delete Cluster (QcstDeleteCluster) API on each node in Partition 1. Then add the nodes from Partition 1 to the cluster, and reestablish all the cluster resource group definitions, recovery domains, and replication. This requires a great deal of work and is also prone to errors. It is very important that you do this procedure only in a site loss situation.
3. Processing a start node operation is dependent on the status of the node that is being started:

The node either failed or an End Node operation ended the node:

a. Cluster resource services is started on the node that is being added
b. Cluster definition is copied from an active node in the cluster to the node that is being started.
c. Any cluster resource group that has the node being started in the recovery domain is copied from an active node in the cluster to the node being started. No cluster resource groups are copied from the node that is being started to an active node in the cluster.

The node is a partitioned node:

a. The cluster definition of an active node is compared to the cluster definition of the node that is being started. If the definitions are the same, the start will continue as a merge operation. If the definitions do not match, the merge will stop, and the user will need to intervene.
b. If the merge continues, the node that is being started is set to an active status.
c. Any cluster resource group that has the node being started in the recovery domain is copied from the primary partition of the cluster resource group to the secondary partition of the cluster resource group. Cluster resource groups may be copied from the node that is being started to nodes that are already active in the cluster.

Cluster recovery

See the following topics to help you recover from failures within your cluster, or even a server failure:

- "Recover from cluster job failures"
- "Recover a damaged cluster object"
- "Recover a cluster after a complete system loss" on page 97
- "Recover a cluster after a disaster" on page 98
- "Restore a cluster from backup tapes" on page 98

Recover from cluster job failures

Failure of a cluster resource services job is usually indicative of some other problem. You should look for the job log associated with the failed job and look for messages that describe why it failed. Correct any error situations. Then, to recover from a failure of a cluster resource services job:

1. End clustering on the node where the job failure occurred. See End a cluster node.
2. Restart clustering on the node. See "Start a cluster node" on page 80.

For more information on cluster jobs, see "Job structure and user queues" on page 87. If you are using a business partner cluster management product, refer to the documentation that came with the product.

Recover a damaged cluster object

While it is unlikely you will ever experience a damaged object, it may be possible for cluster resource services objects to become damaged. The system, if it is an active node, will attempt to recover from another active node in the cluster. The system will perform the following recovery steps:

For a damaged internal object:

1. The node that has the damage ends.
2. If there is at least one other active node within the cluster, the damaged node will automatically restart itself and rejoin the cluster. The process of rejoining will correct the damaged situation.

For a damaged cluster resource group:

1. The node that has a damaged CRG will fail any operation currently in process that is associated with that CRG. The system will then attempt to automatically recover the CRG from another active node.
2. If there is at least one active member in the recovery domain, the CRG recovery will work. Otherwise, the CRG job ends.

If the system cannot identify or reach any other active node, you will need to perform these recovery steps.

For a damaged internal object:
You receive an internal clustering error (CPFBB46, CPFBB47, or CPFBB48).

1. End clustering for the node that contains the damage.
2. Restart clustering for the node that contains the damage. Do this from another active node in the cluster.
3. If Steps 1 and 2 do not solve the problem, remove the damaged node from the cluster.
4. Add the system back into the cluster and into the recovery domain for the appropriate cluster resource groups.

For a damaged cluster resource group:
You receive an error stating that an object is damaged (CPF9804).

1. End clustering on the node that contains the damaged cluster resource group.
2. Delete the CRG (using the DLTCRG command).
3. If there is no other node active in the cluster that contains the CRG object, restore from media.
4. Start clustering on the node that contains the damaged cluster resource group. This can be done from any active node.
5. When you start clustering, the system resynchronizes all of the cluster resource groups. You may need to recreate the CRG if no other node in the cluster contains the CRG.

Recover a cluster after a complete system loss

Use this information in conjunction with the appropriate checklist in the Backup and Recovery

manual for recovering your entire system after a complete system loss when your server loses power unexpectedly.

Scenario 1: Restoring to the same system

1. In order to prevent inconsistencies in the device domain information between the Licensed Internal Code and OS/400(R), it is recommended that you install the Licensed Internal Code using option 3 (Install Licensed Internal Code and Recover Configuration).

Note: For the Install Licensed Internal Code and Recover Configuration operation to succeed, you must have the same disk units — with exception of the load source disk unit if it has failed. You must also be recovering the same release.

2. After you have installed the Licensed Internal Code, follow the *How to Recover Your Disk Configuration* procedure in chapter 5 of the Backup and Recovery manual. These steps will help you avoid having to reconfigure the ASPs.
3. After you have recovered your system information and are ready to start clustering on the node you just recovered, you must start clustering from the active node. This will propagate the most current configuration information to the recovered node.

Scenario 2: Restoring to a different system

After you have recovered your system information and checked the job log to make sure that all objects have restored, you must perform the following steps to obtain the correct cluster device domain configuration.

1. From the node you just restored, delete the cluster.
2. From the active node, perform these steps:

a. Remove the recovered node from the cluster.
b. Add the recovered node back into the cluster.
c. Add the recovered node to the device domain.
d. Create the cluster resource group or add the node to the recovery domain.

Recover a cluster after a disaster

In the case of a disaster where all your nodes are lost, you will need to reconfigure your cluster. In order to prepare for such a scenario, it is recommended that you save your cluster configuration information and keep a hardcopy printout of that information.

See “Backup and recovery of clusters” on page 88 for details.

Restore a cluster from backup tapes

During normal operations, you should never have to restore from a backup tape. The only time that you need to do this would be if a disaster happened, and you lost all of the nodes in your cluster. If a disaster should occur, you would recover by following your normal recovery procedures that you have put in place after you created your backup and recovery strategy. For more information, see the Backup and Recovery

manual.

Frequently asked questions about iSeries Navigator cluster management

The IBM(R) graphical user interface for creating and managing clusters is available in iSeries™ Navigator and accessible through Option 41 (OS/400 - HA Switchable Resources). See “iSeries Navigator cluster management” on page 24 for details on the interface.

Here is a list of iSeries Navigator cluster management questions and answers.

General

1. Is there a checklist that outlines the prerequisites to creating a cluster? (page 99)

iSeries Navigator cluster management

1. Where is the Clusters function located in the iSeries Navigator interface? (page 99)
2. How do I create a cluster? (page 99)
3. What is the relationship between the Clusters folder and the Management Central system group? (page 100)
4. I already have a cluster defined on some iSeries systems in the network. How do I add it so I can view and manage it through iSeries Navigator? (page 100)
5. None of the nodes in my cluster have a status of ″Started″. Which node should I start first? (page 100)
6. Why should I care which node is started first? (page 100)
7. What does the ″Current Primary Node″ column mean in the switchable hardware group and switchable software product folders? (page 101)
8. How do I find a device cluster resource group (CRG) in iSeries Navigator? (page 101)
9. How do I find an application cluster resource group (CRG) in iSeries Navigator? (page 101)
10. How do I find a data cluster resource group (CRG) in iSeries Navigator? (page 101)
11. I want to be able to see the switchable hardware group (device CRG) status without having to go back up to the Switchable Hardware folder to see it. How can I do this? (page 101)

Communications

1. What IP address does the Clusters function in iSeries Navigator use to communicate with the nodes in the cluster? Doesn't it use the IP address of the node name? (page 101)

Security

1. Why are most of the context menus in the Clusters folder in iSeries Navigator disabled or disappeared? (page 102)
2. Does the Clusters function in iSeries Navigator use Application Administration values? (page 102)
3. Why does the Clusters function in iSeries Navigator show a signon window to my nodes in the cluster? (page 102)

Troubleshooting

1. Why isn't the Clusters folder showing up under Management Central? (page 103)
2. I already have a cluster, but it doesn't show up in the Clusters folder. Why? (page 103)
3. Why doesn't the latest status show up in the Clusters folder? (page 103)
4. Why didn't a failover of my switchable hardware group or switchable software product occur? (page 103)
5. I received a message for a damaged object. What can I do about it? (page 103)
6. I'm using the ″Browse″ button in the wizards for the nodes to browse for IP addresses. Why aren't all of the TCP/IP addresses that I expect showing up in the browse window? (page 104)
7. Why are most of the context menus in the Clusters folder in iSeries Navigator disabled or disappeared? (page 102)
8. I was using the ″New Cluster″ wizard and I got a panel titled: ″New Cluster - No Switchable Software Found″. Is this bad? (page 104)
9. One of my nodes has a status of ″Not communicating″. How do I correct this? (page 104)

General

Is there a checklist that outlines the prerequisites to creating a cluster?

Yes. Use the "Cluster configuration checklist" on page 35 to make sure that you are prepared to configure clusters in your environment.

Back to questions (page 98)

iSeries Navigator cluster management: Where is the Clusters function located in the iSeries Navigator interface?

The iSeries Navigator cluster management interface is available as a part of the software package IBM iSeries Access. The Clusters function is located in the Management Central folder in iSeries Navigator. See "iSeries Navigator cluster management" on page 24 for details.

Back to questions (page 98)

How do I create a cluster?

To create a simple cluster using the New Cluster wizard in iSeries Navigator, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Right-click **Clusters**, and select **New Cluster.**
3. Follow the wizard's instructions to create a cluster.

For complete details on creating and configuring a cluster, see "Create a cluster" on page 78.

Back to questions (page 98)

What is the relationship between the Clusters folder and the Management Central system group?

When you use iSeries Navigator to create a cluster, a system group is also created on the Management Central server. The system group is given the same name as the cluster name and the endpoint systems in the system group are the nodes in the cluster. The system group also has its own special type so that iSeries Navigator knows it is a special system group that represents a cluster.

Important: The Management Central system contains the system groups. If you choose to change your current Management Central system in iSeries Navigator, the new management central system will not have the special cluster system groups, and therefore those clusters will not show up in the Clusters folder.

Back to questions (page 98)

I already have a cluster defined on some iSeries systems in the network. How do I add it so I can view and manage it through iSeries Navigator?

To add an existing cluster to appear through iSeries Navigator, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Right-click **Clusters**, and select **Add Existing Cluster.**
3. On the **Add Existing Cluster** window, specify one of the servers in the cluster.
4. Click OK.

Back to questions (page 98)

None of the nodes in my cluster have a status of ″Started″. Which node should I start first?

You should start the node that most recently had a status of ″Started″. For example, say you have two nodes in your cluster: A and B. Node A is currently not started and node B is currently not started. However, node B was the last node to be running with a status of ″Started″. You should start node B first because it will have the most recent information about the cluster.

Back to questions (page 98)

Why should I care which node is started first?

You should care because the node that most recently had a status of ″Started″ is the node that contains the latest information about the cluster. This is important because if you started the other node that had been down the longest, then it may contain outdated information about the cluster. The danger is that the outdated information could get propagated to the other nodes in the cluster when those other nodes are started. For example, say have a two node cluster with nodes A and B. If node B was the most recently active node with a status of ″Started″, then it will contain the latest cluster information. If you choose to start node A first, then it might contain some outdated information, but will still be started. When you then start node B, it will join with a currently active node in the cluster (it joins with node A). The outdated cluster information from node A will get propagated to node B and the result is that both nodes will contain outdated information about the cluster. This is why it is important to start node B first. The outdated cluster information can have an effect on the configuration of the switchable hardware groups. If you find that you have some problems starting up switchable hardware groups because of disk units reporting in on the backup node when the switchable hardware group is showing a different current node, then you will need to change the role of nodes in the recovery domain, making the node which owns the disk units the primary node.

Back to questions (page 98)

What does the ″Current Primary Node″ column mean in the Switchable Hardware, Switchable Software, and Switchable Data folders?

The ″Current Primary Node″ column indicates that the node that is currently serving as the primary node for the switchable hardware group or switchable software product. Or, in cluster API terminology, it means that it is the node with the current role in the recovery domain of primary.

Back to questions (page 98)

How do I find a device cluster resource group (CRG) in iSeries Navigator?

Device CRGs (cluster resource groups) are referred to as Switchable Hardware Groups and found in the **Switchable Hardware** folder in the Clusters folder.

Back to questions (page 98)

How do I find an application cluster resource group (CRG) in iSeries Navigator?

Application CRGs (cluster resource groups) are referred to as Switchable Software Products and found in the **Switchable Software** folder in the Clusters folder.

Back to questions (page 98)

How do I find a data cluster resource group (CRG) in iSeries Navigator?

Data CRGs (cluster resource groups) are referred to as Switchable Data Groups and found in the **Switchable Data** folder in the Clusters folder.

Back to questions (page 98)

I want to be able to see the Switchable Hardware Group (device CRG) status without having to go back up to the Switchable Hardware folder to see it. How can I do this?

As an alternative to navigating to the Switchable Hardware folder every time you want to view status, you can also open a new window with the Switchable Hardware view by right-clicking on the **Switchable Hardware** folder and selecting **Open**. The separate window will show the Switchable Hardware Groups (device CRGs) and their associated status information. This also applies for **Switchable Software** and **Switchable Data**.

Back to questions (page 98)

Communications: What IP address does the Clusters function in iSeries Navigator use to communicate with the nodes in the cluster? Doesn't it use the IP address of the node name?

There is a ″Server″ column in the main Clusters folder that displays information about your configured clusters. The server name is also on the properties panel for each cluster. The server listed in the "Server" column is the node in the cluster that the iSeries Navigator interface uses to communicate with the cluster. It only applies to how iSeries Navigator communicates with the cluster object on the server, not how the nodes in the cluster communicate with one another. The server used by iSeries Navigator cluster management has nothing to do with the current Management Central server.

If the node that iSeries Navigator is using to communicate with the cluster goes down, you can change the communications vehicle to a different node in the cluster to perform cluster actions.

To change the server that will be used by the iSeries Navigator interface to communicate to the cluster, follow these steps:

1. In iSeries Navigator, expand **Management Central**.
2. Expand **Clusters**.
3. Right-click the cluster, and select **Change Server.**

Back to questions (page 98)

Security: Why are most of the context menus in the Clusters folder in iSeries Navigator disabled or disappeared?

Some operations are only available depending on the state of the current configuration of your cluster. For example, you cannot stop a node that is already stopped, you cannot add a node to a cluster that already has the maximum amount of nodes, four, configured. The online help for particlular tasks has explanations of why some of these items are disabled or not available.

Some operations are not available if you don't have enough authority. If you are using iSeries Navigator and you have *SECOFR userclass authority, you will have access to all cluster operations and administration. iSeries Navigator uses Application Administration authority from the current Management Central system to determine if you have Application Administration authority for the various iSeries Navigator cluster management operations.

See Application Administration for details on working with Application Administration.

Back to questions (page 98)

Does the clusters function in iSeries Navigator use Application Administration values?

Yes. iSeries Navigator cluster management uses the Application Administration authority values from the current Management Central system to determine if you have Application Administration authority for various cluster operations.

iSeries Navigator has two types of authority settings for access: **Cluster Operation** and **Cluster Administration**

With the **Cluster Operation** authority, you can:

- View the status of the cluster
- Start and stop nodes
- Start and stop switchable hardware and switchable software
- Perform manual switching of switchable hardware and switchable software

With the **Cluster Administration** authority, you can:

- Create/Delete clusters
- Add and remove nodes
- Add and delete switchable hardware, switchable software, and disk pools
- Change the properties of switchable hardware and switchable software

Back to questions (page 98)

Why does the Clusters function in iSeries Navigator show a signon window to my nodes in the cluster?

In some cases, iSeries Navigator will try to communicate with all of the nodes in the cluster. This depends on the state of your cluster. When iSeries Navigator needs to communicate with a node, it will first search the existing signon cache in iSeries Navigator to try to find an existing open connection. If it does not find an existing connection, it will then ask the user to sign on. If you cancel the signon window, iSeries Navigator will make an attempt to allow the user to do cluster operations. Some operations may not be possible if iSeries Navigator can't communicate with the nodes.

Back to questions (page 98)

Troubleshooting: Why isn't the Clusters folder showing up under Management Central?

It is possible that you didn't do a full install of iSeries(TM) Access for Windows on your PC. You may have performed a basic install or chosen some custom options. See iSeries Access for details on installation.

Back to questions (page 98)

I already have a cluster, but it doesn't show up in the Clusters folder. Why?

The short answer is this: It isn't showing up because there is not a system group on your Management Central system that represents the cluster. That system group representing the cluster is created by iSeries Navigator cluster management when either the cluster is created or the cluster is added to the Clusters folder by using the ″add existing cluster″ action. You can expand the **System Groups** folder in Management Central to see the system groups. The cluster system groups will show up as ″third party″ system groups, but don't assume all ″third party″ system groups are clusters.

Back to questions (page 98)

Why doesn't the latest status show up in the Clusters folder?

iSeries Navigator displays information about configured clusters as a shapshot by going out to the cluster nodes and getting the latest information about the cluster and then displaying it in the iSeries Navigator window. It does not automatically perform regular updates of the information. The best way to get the latest snapshot of information is to do a manual refresh. You can use the ″View″ menu in iSeries Navigator and then choose the ″Refresh″ option. The alternative is to set up iSeries Navigator to perform automatic refreshes.

Back to questions (page 98)

Why didn't a failover of my switchable hardware group, switchable software product, or switchable data group occur?

The most likely scenario is that you didn't have the switchable resource (cluster resource group) started in the cluster. In other words, before the automatic failover was to occur, the status of the switchable resource was not ″Started″. Your switchable resources must be started for a failover to occur.

Back to questions (page 98)

I received a message for a damaged object. What can I do about it?

You may have received a message like this: `CPF811C User queue QUGCLUSRQ in QCLUMGT damaged`

Option 1: One option is to delete the object and restore it. This is only possible if you previously saved the object.

Option 2: Delete the damaged object. For example, if QUGCLUSRQ in library QCLUMGT is damaged, then delete the object. Then add the existing cluster in iSeries Navigator. By adding the cluster, the cluster GUI will check if the objects exist and re-create them if they don't already exist. See How do I add an existing cluster so I can view and manage it through iSeries Navigator? (page 100) for details on adding the existing cluster.

Back to questions (page 98)

I'm using the ″Browse″ button in the wizards for the nodes to browse for IP addresses. Why aren't all of the TCP/IP addresses that I expect showing up in the browse window?

The list is only a candidate list of possible IP addresses. You are not restricted to the list of possible addresses shown in the window. You can enter any cluster interface address you want. Be aware, however, that you will receive errors later if iSeries Navigator can't connect using the IP address you specify as the primary IP address. iSeries Navigator uses the primary IP address to connect to the nodes in the cluster.

Back to questions (page 98)

I was using the ″New Cluster″ wizard and I got a panel titled: ″New Cluster - No Switchable Software Found″. Is this bad?

No, this is not bad and it is not an error. It means exactly what it says; the iSeries Navigator interface could not find any switchable software that could be automatically installed using the wizard. iSeries Navigator requires that the any auto-installable switchable software conform to the "OS/400 architecture for cluster-enabled applications" on page 38. Additionally, iSeries Navigator only supports a subset of this architecture, not all of it.

Back to questions (page 98)

One of my nodes has a status of ″Not communicating″. How do I correct this?

A cluster partition happens if you lose contact between one or more nodes in the cluster and a failure of the lost nodes cannot be confirmed. See "Partition errors" on page 93 for more information.

Sometimes a partitioned condition is reported when there really was a node outage. This can occur when cluster resource services loses communications with one or more nodes, but cannot detect if the nodes are still operational. When this condition occurs, a simple mechanism exists for you to indicate that the node has failed. See "Change partitioned nodes to failed" on page 94 for details.

Back to questions (page 98)

If you have a question that is not on this page, please contact us.

Who to call for cluster suppport

If you need help in deciding whether clustering can benefit your business, or if you run into problems after you have implemented clustering, contact these sources:

- For additional technical marketing assistance or if you would like to hire IBM(R) consultation services, contact the Continuous Availability Center in the iSeries(TM) Technology Center by sending an e-mail to rchclst@us.ibm.com.
- For other problems, contact either the business partner you purchased your clustering software package from or call 1-800-IBM-4YOU (1-800-426-4968).

Related information

Listed below are the Web sites and IBM(R) Redbooks(TM) (in PDF format) that relate to the Clusters topic.

Web sites

High Availability and Clusters

(www.ibm.com/servers/eserver/iseries/ha)
IBM site for High Availability and Clusters

Redbooks

Clustering and IASPs for Higher Availability

(about 6.4 MB)
This redbook presents an overview of cluster and switched disk technology available for iSeries(TM) servers.

iSeries Independent ASPs: A Guide to Moving Applications to IASPs

(about 3.4 MB)
This redbook presents a step-by-step approach to independent ASPs on iSeries servers.

Roadmap to Availability on the iSeries 400(R)

(about 626 KB)
This redpaper presents a step-by-step approach to independent ASPs on iSeries servers.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the link above).
2. Click **Save Target As...** if you are using Internet Explorer. Click **Save Link As...** if you are using Netscape Communicator.

3. Navigate to the directory in which you would like to save the PDF.
4. Click **Save**.

Downloading Adobe Acrobat Reader

You need Adobe Acrobat Reader to view or print these PDFs. You can download a copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html)

.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

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

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

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

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

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
```

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

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

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:
Application System/400
AS/400
e (logo)
IBM
iSeries
Operating System/400
OS/400
400

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Terms and conditions for downloading and printing information

Permissions for the use of the information you have selected for download are granted subject to the following terms and conditions and your indication of acceptance thereof.

Personal Use: You may reproduce this information for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of this information, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display this information solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of this information, or reproduce, distribute or display this information or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the information or any data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the information is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations. IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THIS INFORMATION. THE INFORMATION IS PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

By downloading or printing information from this site, you have indicated your agreement with these terms and conditions.

Code disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Printed in USA