High Availability Solution for WebSphere Message Queue in a Single Instance Queue Manager Configuration Using Linux HA and a Shared DASD Device
Table of Contents

Table of Contents .......................................................................................................................................................... 2
Abstract ................................................................................................................................................................. 3
Architecture at a glance ............................................................................................................................................. 3
Solution Implementation ............................................................................................................................................... 5
Defining the Shared Disk........................................................................................................................................... 5
 MDISK 0301 3390 13410 10016 DT0064 MWV .................................................................................................. 6
 LINK LAC0025 0301 0301 MW .......................................................................................................................... 6
 Rdevice xxxx Type DASD Shared Yes MDC OFF .................................................................................................. 6
 Formatting the DASD device: ................................................................................................................................... 7
 Partitioning the DASD device: ................................................................................................................................ 7
 Making a file system on the DASD device: ............................................................................................................ 8
 Create a mount point for the DASD device: ........................................................................................................... 9
 Make the bin directory on the DASD: ..................................................................................................................... 9
 Installing the SupportPacs to Obtain the HACMP Scripts ...................................................................................... 10
 A caveat regarding SupportPacs: .......................................................................................................................... 10
 Untar the SupportPac tar files: ............................................................................................................................. 10
 Creating a Queue Manager Instance on the Shared Disk ...................................................................................... 11
 Working with the HACMP Scripts .......................................................................................................................... 12
 Defining Heartbeat resources .................................................................................................................................. 16
 The layout of the cib.xml file: ................................................................................................................................. 18
 Defining the WebSphere MQ Queue Manager instance (mqm) resource ............................................................... 22
 Defining the IPaddr resource ................................................................................................................................... 24
 Configuring the pingd resource ............................................................................................................................. 24
 Configuring Constraints ......................................................................................................................................... 27
 Using cibadmin to populate the CIB ...................................................................................................................... 29
 Loading the resource definitions ............................................................................................................................ 29
 Screen shot of the hb_gui showing the full configuration: .................................................................................... 35
 Test Experience ........................................................................................................................................................ 36
 Test Summary ........................................................................................................................................................... 37
Abstract
This paper describes the design and implementation of a high availability (HA) solution for a WebSphere® Message Queue Single Instance (SI) Queue Manager (QM) configuration on a pair of virtual Linux servers running on an IBM System z10®. The solution tested during the creation of this paper used SUSE Linux Enterprise Server (SLES) 10 SP3, Linux® HA (Heartbeat version 2.1.4), and WebSphere MQ version 6.0.2. Although the specifics listed in this paper have been tested by the authors only with the software versions described, the same techniques are applicable to other Linux distributions and other versions of WebSphere MQ with some modifications.

This document assumes a basic working understanding of Linux on IBM System z® including IBM networking technologies, and WebSphere MQ operation and installation.

Architecture at a glance
The WebSphere MQ HA configuration consists of three resources, which will be managed by Heartbeat:

1. WebSphere MQ Queue Manager - a WebSphere MQ system service that provides a logical container for the message queue, and is responsible for transferring data to other queue managers using message channels.
2. A DASD device that contains the WMQ queue and log files, which is used as the persistent state shared between the two cluster nodes.
3. A virtual IP address, which serves as the public interface to the WMQ QM.

The Linux HA (High Availability Linux) project is an open source project that provides a high-availability clustering solution for Linux and other operating systems. Linux HA promotes reliability, availability, and serviceability (RAS).

Linux HA will be referred to as Heartbeat in this document.

Heartbeat sometimes experiences a condition, called split brain, when only two systems are used. This happens when both machines assume that the other system has failed. Having a third system prevents this condition by providing quorum of three votes rather than two votes.

While the split brain decision issue is worthy of concern, it was not encountered in the testing of this solution.
Heartbeat provides mechanisms for monitoring and providing fail-over for the resources required for this configuration. Some additional scripts used in this solution are available as part of WebSphere MQ SupportPac™ which are available on the web at no charge from their respective links below:

- **IC91 (has a prereq of SupportPac MC91)**

This table shows the script that is needed from the IC91 SupportPac. The script is located in the Heartbeat/resource.d directory in the tree that is created when the ic91.tar.z tar file is expanded.

<table>
<thead>
<tr>
<th>Name of Script</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>hacrtmqm</td>
<td>Create a WebSphere MQ Queue Manager instance that will be amenable to High Availability.</td>
</tr>
<tr>
<td>Hamqm_appmon_su</td>
<td>Used by the High Availability product to monitor the WebSphere MQ Queue Manager.</td>
</tr>
<tr>
<td>Hamqm_start</td>
<td>Wrapper script for hamqm_start_su.</td>
</tr>
<tr>
<td>Hamqm_stop_su</td>
<td>Used by the High Availability product to stop the WebSphere MQ Queue Manager.</td>
</tr>
<tr>
<td>hadltmqm</td>
<td>Delete a WebSphere MQ Queue Manager instance on the owning node.</td>
</tr>
<tr>
<td>Hamqm_qm_directory</td>
<td>Displays the fully qualified path of the WebSphere MQ Queue Manager directory.</td>
</tr>
<tr>
<td>Hamqm_start_su</td>
<td>Used by the High Availability to start the WebSphere MQ Queue Manager.</td>
</tr>
<tr>
<td>hamqproc</td>
<td>Used internally by other scripts.</td>
</tr>
<tr>
<td>halinkmqm</td>
<td>Make WebSphere MQ aware of a Queue Manager instance, so that it works in a High Availability environment</td>
</tr>
<tr>
<td>Hamqm_running</td>
<td>Used by the High Availability product to test whether the WebSphere MQ Queue Manager is running or not.</td>
</tr>
<tr>
<td>Hamqm_stop</td>
<td>Wrapper script for hamqm_stop_su.</td>
</tr>
</tbody>
</table>

- **MC91 (marked as withdrawn but still available). MC91 was recently updated for WMQ V7.**
This table shows the scripts that are needed from the MC91 SupportPac. All are located in the HACMP directory in the tree that is created when the mc91.tar.z tar ball is expanded.

The solution as documented was designed and tested to insure the integrity of the WMQ message queue and correct failover actions for the following three use cases:

<table>
<thead>
<tr>
<th>Script Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>mqm</td>
<td>provides ability to stop, start, and monitor a WebSphere MQ Queue Manager instance by the High Availability Product</td>
</tr>
</tbody>
</table>

- A full system failure, such as a kernel panic or any planned or unplanned system halt.
- A network failure on one system at a time, such as loss of network connectivity hardware or virtual networking devices such as IBM VSWITCH. It is therefore vital that the networking architecture ensures redundant and orthogonal networking devices for each system in the HA solution.
- A WebSphere MQ Queue Manager failure.

In the unlikely event that a failure listed above should occur, the Heartbeat software stack on the surviving system detects that failure and performs necessary operations to failover connectivity and all three resources to the surviving system.

**Solution Implementation**

After installing SUSE Linux Enterprise Server along with the appropriate packages from associated High Availability pattern for that release, it is necessary to install the version of WebSphere MQ that is suitable for the SUSE release you have selected.

The basic Heartbeat configuration can be done using YaST. Consult the documentation for the Linux distribution you have installed. For those following the recommended solution software levels as tested in this document, the SUSE provided HA documentation can be found here: [www.novell.com/documentation/sles10/pdfdoc/heartbeat/heartbeat.pdf](http://www.novell.com/documentation/sles10/pdfdoc/heartbeat/heartbeat.pdf)

**Defining the Shared Disk**

A shared disk device is needed for use by the WebSphere MQ Queue Manager. This device will also contain the scripts from the MC91 SupportPac mentioned previously.

**Note:** It is possible to use a SCSI device in place of a DASD device for this solution.

The DASD device must be configured to allow multiple systems to perform both read and write operations to the device. The integrity of the device will be maintained by Heartbeat. Heartbeat will insure that the device is mounted on only one system at a time.
In the solution presented in this document two systems are used, named: lac0025 and lac0030.

If both systems are on the same LPAR, that is they are guests of the same z/VM system, then one system will require a MDISK (mini disk) directory control statement in it’s z/VM system directory entry, while the other system will require a LINK directory statement in it’s z/VM system directory entry.

The system directory entry for lac0025 would contain this MDISK directory control statement:

```
MDISK 0301 3390 13410 10016 DT0064 MWV
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0301</td>
<td>The virtual address of the device.</td>
</tr>
<tr>
<td>3390</td>
<td>The device type.</td>
</tr>
<tr>
<td>13410</td>
<td>The starting cylinder for the minidisk that you are defining on the real device it is being mapped on.</td>
</tr>
<tr>
<td>10016</td>
<td>The number of cylinders allocated to the minidisk.</td>
</tr>
<tr>
<td>DT0064</td>
<td>The volume serial number of the real DASD volume on which the minidisk resides.</td>
</tr>
<tr>
<td>MW</td>
<td>The mode of the minidisk. MW allows Multiple-write access.</td>
</tr>
<tr>
<td>V</td>
<td>Tells CP to use its virtual reserve/release support in the I/O operations for the minidisk. MWV means the minidisk functions with write linkage using CP’s virtual reserve/release.</td>
</tr>
</tbody>
</table>

The system directory entry for lac0030 would contain this LINK directory control statement:

```
LINK LAC0025 0301 0301 MW
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAC0025</td>
<td>The user ID in the system directory, whose entry is to be searched for the device.</td>
</tr>
<tr>
<td>0301</td>
<td>The virtual address that was assigned to the device on lac0025.</td>
</tr>
<tr>
<td>0301</td>
<td>The virtual device number that is to be assigned to the device on lac0030.</td>
</tr>
<tr>
<td>MW</td>
<td>The mode of the device. MW allows Multiple-write access.</td>
</tr>
</tbody>
</table>

In order to update the z/VM system directory, a user ID with at least Class B privileges is needed.

It is best to dedicate real DASD device(s) that will be shared by the individual Linux guests when the guests reside on multiple LPARs. The shared disk device must be configured using an RDEVICE directory control statement in the system directory entries of both systems.

```
Rdevice xxxx Type DASD Shared Yes MDC OFF
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx</td>
<td>This is real device address of the shared disk device.</td>
</tr>
</tbody>
</table>
With the disk online, it is then necessary to format the disk, partition it, and then make a file system. For our testing we used the ext3 file system.

**Formatting the DASD device:**

```bash
# dasdfmt -b 4096 -p -f /dev/dasdc
```

Drive Geometry: 10016 Cylinders * 15 Heads = 150240 Tracks

I am going to format the device `/dev/dasdc` in the following way:

Device number of device: 0x301
  Labelling device : yes
  Disk label : VOL1
  Disk identifier : 0X0301
  Extent start (trk no) : 0
  Extent end (trk no) : 150239
  Compatible Disk Layout : yes
  Blocksize : 4096

--- ATTENTION! ---

All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).
cyl 10016 of 10016 #---------------------------------------------------------------| 100%

Finished formatting the device.
Rereading the partition table... ok

**Partitioning the DASD device:**

```bash
# fdasd /dev/dasdc –a
```

reading volume label ..: VOL1
reading vtoc ..........: ok

auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...

Making a file system on the DASD device:

# mkfs.ext3 -b 4096 /dev/dasdc1

mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
903168 inodes, 1802856 blocks
90142 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1849688064
56 block groups
32768 blocks per group, 32768 fragments per group
16128 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 35 mounts or 180 days, whichever comes
first. Use tune2fs -c or -i to override.
Create a mount point for the DASD device:

# mkdir -m 755 /MQHA

Several of the scripts that are provided by SupportPac MC91 expect the shared disk to be mounted at /MQHA, which was used in this solution. To use a different mount point, you will need to modify the affected scripts. They are:

Hacrtmqm
hadltmqm
halinkmqm
hamqm_applmon.csq1
hamqm_applmon.ha.csq1
hamqm_applmon_su
hamqm_qm_directory
hamqm_running
hamqm_start
hamqm_start_su
hamqm_stop

Make the bin directory on the DASD:

Mount the DASD device at /MQHA then make a bin directory. This is where the scripts from the SupportPacs will be placed.

# mkdir -m 755 /MQHA/bin
Installing the SupportPacs to Obtain the HACMP Scripts

A caveat regarding SupportPacs:
The SupportPacs are provided as is and are not supported. In the text that follows, some errors are described that were found during our testing. These errors will need to be corrected manually. Also one or possibly both of the SupportPacs referenced in this document are marked Withdrawn. This is not an indication that you cannot use the SupportPac. It merely indicates that the SupportPac has been replaced by some other medium for later releases of WMQ.

Untar the SupportPac tar files:
After you have completed the set up for the shared disk device, the MC91 and IC91 SupportPacs can be installed. Note that even though MC91 does not provide support for Heartbeat directly, it does provide support for HACMP as well as other high availability products. HACMP is the High Availability Cluster Multi-Processing product, which is available for AIX® systems. The scripts can be used with Heartbeat with some minor modifications.

Heartbeat provides a frame work for creating a high availability cluster and defining the policy required to implement this solution. The SupportPacs require the scripts that execute the commands necessary to migrate resources from a system experiencing a failure to a waiting back up system.

The MC91 SupportPac is packaged as a tar file. Download the mc91.tar.Z file and expand it to /tmp on the system where the shared disk device is currently mounted.

Commands to expand the tar file:

```
# cd /tmp
# mkdir mc91
# cd mc91
# tar -xzvf /root/mc91.tar.Z
```

When the archive is expanded, copy the hacmp scripts from the support pac /hacmp directory into /MQHA/bin, and set the execute permissions on. An example command to do this is:

```
# cp /tmp/mc91/hacmp/* /MQHA/bin/
# chmod +x /MQHA/bin/*
```
Creating a Queue Manager Instance on the Shared Disk

With the previous steps completed, it is now an appropriate time to prepare the shared disk for use by the WebSphere MQ Queue Manager. To begin, create the log and data directories as shown below:

```
# mkdir -m 755 /MQHA/csq1/log
# mkdir -m 755 /MQHA/csq1/data
```

Next, run the hacrtmqm script from /MQHA/bin to create the Queue Manager instance on the shared disk. This must be done under the mqm userid, which is nominally created during the WebSphere MQ installation. Note that the owner of the shared disk mount point should be mqm, while the group should be specified as mqm.

By default the hacrtmqm script uses a Queue Manager name of csq1 with the data path set to /MQHQ/csq1/data and the log path set to /MQHA/csq1/log. While the default settings may be used, you will probably want to modify the Queue Manager name (qmgr1) to a name that fits the standards for your organization.

This is a section from the hacrtmqm script:

```
# qmgr1=csq1
# MQHAFSDATA="/MQHA/$qmgr1/data"
# MQHAFSLOG="/MQHA/$qmgr1/log"
```

Example of running the hacrtmqm command using the mqm user ID:

```
# su mqm hacrtmqm
```

On the second system, run the halinkmqm script from the mqm user ID. If you have additional systems in your HA cluster, you will need to run halinkmqm on the additional systems as well.

You also need to make one other modification on the second system and any additional systems you may add to the HA cluster. add the following text to /var/mqm/mqs.ini:

```
QueueManager:
    Name=csq1
    Prefix=/MQHA/csq1/data
    Directory=csq1
```

This defines the Queue Manager instance to the other systems in the cluster. It provides the Queue Manager name as well as information describing the path name for the files required by the Queue Manager that have been placed on the shared disk device,
Based on the previous example settings in hacrtmq, the syntax for halinkmqm would be:

```
# su mqm – c halinkmqm csq1 csq1 /MQHA/csq1/data
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>csq1</td>
<td>Queue Manager instance name</td>
</tr>
<tr>
<td>csq1</td>
<td>Mangled Queue Manager directory name</td>
</tr>
<tr>
<td>/MQHA/csq1/data</td>
<td>Location of the QM’s data and logh files</td>
</tr>
</tbody>
</table>

**Working with the HACMP Scripts**

Next, each of the four HACMP scripts need to be tested and debugged as required. The modifications shown were needed for SLES 10 SP3. The remainder of this section describes the scripts needed for this HA architecture, as well as the customizations performed during the course of our testing.

- The hamqm_start_su script is used to start the Queue Manager. Example syntax:
  ```
  # su mqm -c /MQHA/bin/hamqm_start_su
  ```
  No modification to the hamqm_start_su script is necessary.

- The hamqm_stop_su script is used to stop the Queue Manager. Example syntax:
  ```
  # su mqm -c /MQHA/bin/hamqm_stop_su
  ```
  Problems may be encountered with the hamqm_stop_su script that will need to be resolved. Example output:

  ```
  hamqm_stop_su is running: "csq1" 5
  /MQHA/bin/hamqm_qm_directory: line 25: nawk: not found
  /MQHA/bin/hamqm_running: line 34: /usr/bin/cat: not found
  /MQHA/bin/hamqm_running[65]: [: nawk:: unknown operator
  ```

  For the first error shown in this example, nawk may not be available the Linux distribution used. However, awk is a good substitute. The easiest solution is to create a symlink for nawk pointing to awk.

  ```
  # ln -s /usr/bin/awk /usr/bin/nawk
  ```

  For the second error, the path in the script did not contain the proper path for cat. Run the `which` command to find the location of the cat executable, then set up a symbolic link to the true location of the file.

  ```
  # which cat
  /bin/cat
  # ln -s /bin/cat /usr/bin/cat
  ```
The `hamqm_running` script is used to determine the status of the Queue Manager. The syntax used to execute `hamqm_running` is:

```
# hamqm_running
```

This script also had references to `nawk`, which was fixed with the symlink mentioned earlier. This script also needed to have the path to the `cat` command fixed.

The final script is named `hamqm_applmon.csq1`. This script is used to determine the status of the Queue Manager. This script is created when you run `hacrtmqm` or `halinkmqm`.

Example syntax:

```
# /MQHA/bin/hamqm_applmon.csq1
```

This script worked with no modifications.

Only one file is needed from the expanded IC91 SupportPAC tar file (ic91.tar.Z) download. Expanding the download into a directory of your choice, for example `/tmp/ic91`.

Commands to expand the tar file:

```
# cd /tmp
# mkdir ic91
# cd ic91
# tar -xzvf /root/ic91.tar.Z
```

The only file we need from this SupportPac is the `mqm` script, which is located in `/tmp/ic91/Heartbeat/resource.d` based on the commands shown.

The comments in the script indicate the intended location within the file system for the file. In our testing, the comment indicated that the script was designed to reside at the following path:

```
/etc/ha.d/resource.d/mqm
```

Though the intent of the original author was probably ideal for his or her deployment, when using this script on SUSE Linux Enterprise Server, the script should be placed in `/etc/init.d/`. Heartbeat will not recognize the script as a Linux Standards Base (LSB) type script unless it is placed in this location. It is recommended to change the commented path at the top of the file to match that of the final location in the file system for future reference.

Example of command to copy the `mqm` script:

```
# cp /tmp/ic91/Heartbeat/resource.d/mqm /etc/init.d/mqm
```

The `mqm` script performs four functions. Each function calls a script provided by the MC91 SupportPac.
This table describes the functions provided by the mqm script and shows the script that is executed to perform that function and the parameters that are passed to the called script.

<table>
<thead>
<tr>
<th>Function</th>
<th>Script</th>
<th>Parameters Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Hamqm_start_su</td>
<td>Queue Manager name</td>
</tr>
<tr>
<td>stop</td>
<td>Hamqm_stop_su</td>
<td>Queue Manager name, Time Out - seconds (set to 30)</td>
</tr>
<tr>
<td>status</td>
<td>Hamqm_running</td>
<td>Queue Manager name</td>
</tr>
<tr>
<td>monitor</td>
<td>Hamqm_running</td>
<td>Queue Manager name</td>
</tr>
</tbody>
</table>

There was an error in the mqm script (part of the IC91 SupportPac) as provided. This is a section from the original script:

```bash
STRMQM=/usr/bin/strmqm
test -x $STRMQM || exit 5

ENDMQM=/usr/bin/strmqm
test -x $SENDMQM || exit 5
QM=$1
```

The variable ENDMQM is set to /usr/bin/strmqm. It should have been set to /usr/bin/endmqm.

Our testing revealed that while mqm worked when run manually, it did not work when it was invoked by Heartbeat. Root cause analysis determined that the Heartbeat software was not able to correctly pass parameters to the called script. To get around this problem, the “QM” variable needed to be set to the name of the Queue Manager (csq1 from our ongoing example).

A section of the script containing the requisite changes is shown below. Note the last two lines with the commented original QM value and the newly assigned QM variable set to “csq1”:

```bash
STRMQM=/usr/bin/strmqm
test -x $STRMQM || exit 5

#ENDMQM=/usr/bin/strmqm
#test -x $SENDMQM || exit 5
ENDMQM=/usr/bin/endmqm
test -x $ENDMQM || exit 5

#QM=$1
QM="csq1"
```

The mqm script also had a problem with the status function.

# ./mqm csq1 status
/mqm: line 58: [: status: integer expression expected
Queue Manager csq1 is running

status)
  status=`/MQHA/bin/hamqm_running $QM`

  if [ status -lt 1 ]
    then
      echo "Queue Manager $QM is not operational"
      exit 1
    fi
  echo "Queue Manager $QM is running"
  exit 0
check

For this script to run properly, change 'status -lt 1' to '$status -lt 1'.
Defining Heartbeat resources

A resource can be any type of a service that a computer provides. Heartbeat can manage only resources that can be controlled by a Heartbeat Resource Agent. In this solution, two types of Resource Agents are used. Both types are scripts that can be used to start, stop, and monitor the status of a resource.

One type of Resource Agent is the Linux Standards Base (LSB) style script. The other type of Resource Agent is the Open Cluster Framework (OCF) style script. OCF scripts are provided by the heartbeat RPM. LSB scripts are often provided by the vendor of a resource, but can also be user written. While the coding style of the two types of Resource Agents differs, the functionality they provide is similar. Each provides Heartbeat with functions to manage a resource (such as: start, stop, and monitor the status of that resource).

Resources are configured in the Cluster Information Base (CIB). The contents of the CIB are stored as XML code in a file (/var/lib/heartbeat/crm/cib.xml).

This solution requires that Heartbeat manage three resources. They are the shared disk, the Queue Manager (QM), and the IP address. To make management of these three resources easier, the resources are configured into a resource group. The purpose of the resource group is to allow all three resources to be managed as though they were one single resource when doing things like stopping, starting, or migrating the group from one system to another. Resources that are part of a group can still be managed individually.

In order for Heartbeat to be aware of dependencies between one resource and another or where a resource can be run, policy rules called constraints are needed. Order Constraints insure, for example, that resource “a” is started before resource “b” when the order of events is critical to proper operation. Location Constraints provide information that tells Heartbeat which systems individual resources can be started on.

One way resources can be defined to Heartbeat is to code the appropriate XML and use the cibadmin program to place the code into the CIB. The cibadmin utility is provided by Heartbeat.

Resources can also be defined using the hb_gui program, which is also provided by Heartbeat. This program presents an x11 based GUI interface. Information on how to use the GUI to define resources can be found here: www.novell.com/documentation/sles10/heartbeat/?page=/documentation/sles10/heartbeat/data/man_crmresource.html
The remainder of this section details the procedure for creating the XML code that describes this solution to Heartbeat, and the mechanism for implementing the actual Heartbeat configuration by loading the XML into the CIB.

XML is stored in structures called elements. Elements are constructed using tag pairs. The first tag is the opening tag and consists of the element name enclosed in angle brackets. The second tag in the pair is the closing tag and consists of the element name preceded by a slash enclosed in angle brackets. For example to code an element called “sample” the opening tag would be coded <sample>. The end tag would be </sample>. Any data to be associated with the element would be placed between the tags.

Ex.: <sample> sample data </sample>

The first element in an XML file is called the root element. In the cib.xml file the root element is “cib”. The entire file is delimited by the tags <cib> and </cib>. Elements coded within the root element are called sub elements. The root element is considered the parent element and it’s sub elements are considered children. Elements may have other elements nested with in them.

Ex.;

```
<parent>
  <child1>
    <subelement1>
      </subelement1>
  </child1>
  <child2>
    </child2>
</parent>
```

It might be helpful to look at the structure of the cib.xml file. The layout provided shows the structure of the file. The blue text describes what would be contained in various fields.
The layout of the cib.xml file:

```xml
<cib ...
  <configuration>
    <crm_config>
      <cluster_property_set id="cib-bootstrap-options">
        <attributes>...
        </attributes>
      </cluster_property_set>
    </crm_config>
    <nodes>
      <node id="first node">
        ...
      </node>
      <node id="next node">
        ...
      </node>
    </nodes>
    <resources>
      <group id="group name">
        <meta_attributes id="group name_meta_attrs">
          <attributes>
            <nvpair id="Heartbeat generated id" name="variable name" value="data"/>
          </attributes>
        </meta_attributes>
        <primitive class="ocf or lsb" type="script name" provider="heartbeat" is_managed="true" id="resource name">
          <instance_attributes id="resource name_instance_attrs">
            <attributes>
              <nvpair id="Heartbeat generated id" name="variable" value="data"/>
            </attributes>
          </instance_attributes>
          <operations>
            <op id="resource name" name="monitor" interval="seconds" timeout="seconds" start_delay="seconds" disabled="false" role="Started"/>
          </operations>
        </primitive>
      </group>
      Other resources may be defined here...
    </resources>
    <constraints>
      Various policy settings in the form of constraints are defined here...
    </constraints>
  </configuration>
</cib>
```
In the CIB the resources element consists of some number of primitive tags, which describe individual resources. A group is formed by a set of primitive tags that are nested within a group tag.

The structure is similar to this:

```xml
<cib>
  <resources>
    <group>
      <primitive/>
      <primitive/>
    </group>
  </resources>
</cib>
```

Primitive tags contain information that is required by both Heartbeat and the resource agent that manages that resource. Information that is required by Heartbeat is provided in the meta_attributes tag. Information required by the resource agent is provided in the instance_attributes tag and the operations tag.

Nested within the instance_attributes tag is an attributes tag. In XML elements may contain attributes which are coded in a name/value pair format. Name/value pairs consist of an attribute name followed by the equals sign and the attribute value in quotes.

Ex.:

```xml
<weekday day="Monday"/>
```

When configuring the attributes tag a special name value pair tag called nvpair is used to provide information required by the resource agent. An nvpair tag is enclosed within angle brackets. It starts with the word “nvpair” and ends with a “/”. The nvpair is formed by providing an “id” parameter that is used to refer to the name associated with the attribute being provided, and a “value” parameter that contains the value being attributed to attribute.

Ex.:

```xml
<nvpair id="sample_id" value="sample_data"/>
```

Primitive tags may also contain an operations tag that provides information for special operations performed by the resource agent such as monitoring the resource. The operation tag
contains an op tag, which is enclosed in angle brackets and starts with the word “op”. A slash “/” delimits the end of tag.

Ex.: <operations>
    <op id="op_name" parm1="data1" />
</operations>

Defining the Filesystem resource

Heartbeat needs to know what type of resource agent is being used to control the shared disk resource. Since the resource agent is actually a script, Heartbeat needs to know the name of the script and where it is located. This information is provided with the primitive tag.

Here is the primitive tag for the shared disk resource:

```
<primitive class="ocf" type="Filesystem" provider="heartbeat" is_managed="true" id="MQHA_Filesystem">
```

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>ocf</td>
<td>Script style used for resource agent</td>
</tr>
<tr>
<td>type</td>
<td>filesystem</td>
<td>Name of script used as resource agent</td>
</tr>
<tr>
<td>provider</td>
<td>heartbeat</td>
<td>The combination of class and provider tells Heartbeat where the resource agent script is located. In this case it is located in /etc/ha.d/resource.d.</td>
</tr>
<tr>
<td>is_managed</td>
<td>true</td>
<td>Is the resource managed by Heartbeat?; true or false</td>
</tr>
<tr>
<td>id</td>
<td>MQHA_Filesystem</td>
<td>The name Heartbeat will associate with this resource. This value can be changed by the user.</td>
</tr>
<tr>
<td>Location of the filesystem resource agent</td>
<td>/etc/ha.d/resource.d/filesystem</td>
<td></td>
</tr>
</tbody>
</table>

The resource agent needs to know about the resource it is controlling. In this case the resource is the shared disk device. There is a filesystem on the shared disk and the disk must be mounted at a specific mount point.
Here are the instance_attributes and attributes tags:

```
<instance_attributes id="MQHA_Filesystem_instance_attrs">
  <attributes>
    <nvpair name="device" value="/dev/disk/by-path/ccw-0.0.0301-part1"/>
    <nvpair name="directory" value="/MQHA"/>
    <nvpair name="fstype" value="ext3"/>
  </attributes>
</instance_attributes>
```

The Filesystem script requires three pieces of information, device, directory, and fstype.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>/dev/disk/by-path/ccw-0.0.0301-part1</td>
<td>This nvpair provides the by-path ID of the shared disk</td>
</tr>
<tr>
<td>directory</td>
<td>/MQHA</td>
<td>This nvpair describes the mount point of the shared disk device</td>
</tr>
<tr>
<td>fstype</td>
<td>ext3</td>
<td>this nvpair describes the filesystem type used on the shared disk</td>
</tr>
</tbody>
</table>

The Filesystem resource agent provides the ability to monitor the disk device to insure that it is operational at all times. The settings for the monitor operation are provided with the operations sub element.

Here is the operations tag:

```
<operations>
  <op id="MQHA_Fileystem" name="monitor" interval="120" timeout="60" start_delay="0" disabled="false" role="Started"/>
</operations>
```

The information provided consists of:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>MQHA_Filesystem</td>
<td>The ID associated with the operation. This value can be changed by the user.</td>
</tr>
<tr>
<td>name</td>
<td>monitor</td>
<td>The name of the operation</td>
</tr>
<tr>
<td>interval</td>
<td>120</td>
<td>Length of time in seconds between monitor operations</td>
</tr>
<tr>
<td>timeout</td>
<td>60</td>
<td>How many seconds to wait for the monitor operation to complete</td>
</tr>
<tr>
<td>start-delay</td>
<td>0</td>
<td>How many seconds to wait before beginning the monitor operation</td>
</tr>
<tr>
<td>disabled</td>
<td>false</td>
<td>Should the monitor operation be disabled?: true or false</td>
</tr>
<tr>
<td>role</td>
<td>started</td>
<td>Initial state of the monitor operation (started or stopped)</td>
</tr>
</tbody>
</table>
Defining the WebSphere MQ Queue Manager instance (mqm) resource

Heartbeat needs to know what type of resource agent is being used to control the Queue Manager (QM) instance resource. Since the resource agent is actually a script, Heartbeat needs to know the name of the script and where it is located. This information is provided with the primitive tag.

Here is the primitive tag:

```xml
<preimitive id="MQHA_mqm" class="lsb" type="mqm" provider="heartbeat" is_managed="true">...
```

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>lsb</td>
<td>Script style used for resource agent</td>
</tr>
<tr>
<td>Type</td>
<td>IPaddr</td>
<td>Name of script used as resource agent</td>
</tr>
<tr>
<td>provider</td>
<td>heartbeat</td>
<td>The combination of class and provider tells Heartbeat where the resource agent script is located. In this case it is located in /etc/rc.d</td>
</tr>
<tr>
<td>is_managed</td>
<td>true</td>
<td>Is the resource managed by Heartbeat?; true or false</td>
</tr>
<tr>
<td>id</td>
<td>MQHA_IP_mqm</td>
<td>The name Heartbeat will associate with this resource. This value can be changed by the user.</td>
</tr>
<tr>
<td>Location of the mqm resource agent</td>
<td>/etc/rc.d/mqm</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The mqm script is provided by the IC91 SupportPac and is placed in /etc/rc.d by the person implementing the solution.

The mqm script provides the ability to monitor the QM. In order to do so, mqm requires information describing how often to check the status of this resource (monitor), how long to wait for a status check to complete, and if there should be a time delay before starting to monitor the resource, or if the monitor function should be disabled for some reason. This information is provided on the op tag, which is encapsulated in the operations tag.

The XML code for the operations section follows:

```xml
<operations>
  <op id="MQHA_mqm_monitor" name="monitor" interval="60" timeout="15" start_delay="0" disabled="false"/>
</operations>
```
Here is a description of information that is coded on the op statement:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>MQHA_mqm_monitor</td>
<td>The ID of the operation. This value can be changed by the user.</td>
</tr>
<tr>
<td>name</td>
<td>monitor</td>
<td>The name of the operation</td>
</tr>
<tr>
<td>interval</td>
<td>60</td>
<td>Length of time in seconds between monitor operations</td>
</tr>
<tr>
<td>timeout</td>
<td>15</td>
<td>How many seconds to wait for the monitor operation to complete</td>
</tr>
<tr>
<td>start-delay</td>
<td>0</td>
<td>How many seconds to wait before beginning the monitor operation</td>
</tr>
<tr>
<td>disabled</td>
<td>false</td>
<td>Should the monitor operation be disabled?; true or false</td>
</tr>
</tbody>
</table>

The mqm resource agent also requires some additional information, which is provided in the meta_attributes section of the XML. Within that section there is an instance_attributes section that contains an empty attributes tag indicating that there are no instance_attributes for this resource.

```
<instance_attributes id="MQHA_mqm_instance_attrs">
  <attributes/>
</instance_attributes>
```

The meta_attributes tag has a nested attributes tag. The attributes tag contains two nvpair tags.

```
<meta_attributes id="MQHA_mqm_meta_attrs">
  <attributes>
    <nvpair id="MQHA_mqm_metaattr_is_managed" name="is_managed" value="true"/>
    <nvpair id="MQHA_mqm_metaattr_target_role" name="target_role" value="started"/>  
  </attributes>
</meta_attributes>
```

The first nvpair tells Heartbeat that the resource is managed:

```
<nvpair id="MQHA_mqm_metaattr_is_managed" name="is_managed" value="true"/>
```
The second nvpair tells Heartbeat that the resource should have a status of started:

```xml
<nvpair id="MQHA_mqm_metaattr_target_role" name="target_role" value="started"/>
```

**Defining the IPaddr resource**

Heartbeat needs to know what type of resource agent is being used to control the IPaddr resource. Since the resource agent is actually a script, Heartbeat needs to know the name of the script and where it is located. This information is provided with the primitive tag.

Here is the primitive tag for the IPaddr resource:

```xml
<primitive id="MQHA_IP_addr" class="ocf" type="IPaddr" provider="heartbeat">
```

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>ocf</td>
<td>Script style used for resource agent</td>
</tr>
<tr>
<td>type</td>
<td>IPaddr</td>
<td>Name of script used as resource agent</td>
</tr>
<tr>
<td>provider</td>
<td>heartbeat</td>
<td>The combination of class and provider tells Heartbeat where the resource agent script is located. In this case it is located in /etc/ha.d/resource.d.</td>
</tr>
<tr>
<td>is_managed</td>
<td>true</td>
<td>Is the resource managed by Heartbeat?; true or false</td>
</tr>
<tr>
<td>id</td>
<td>MQHA_IP_addr</td>
<td>The name Heartbeat will associate with this resource. This value can be changed by the user.</td>
</tr>
<tr>
<td>Location of the IPaddr resource agent</td>
<td>/etc/ha.d/resource.d/IPaddr</td>
<td></td>
</tr>
</tbody>
</table>

The resource agent needs to know about the resource it is controlling. In this case the resource is the public IP address used by the Queue Manager. An nvpair tag in the attributes tag of the instance_attributes tag provides the IP address.

```xml
<nvpair name="ip" value="192.168.70.16"/>
```

**Configuring the pingd resource**

The pingd resource agent monitors connectivity to specific hosts or IP addresses. In this solution, the pingd resource agent is used to monitor the IP address of the gateway. This provides a way to determine the health of the network. Pingd uses a clone resource to set up an instance of the pingd daemon on each node in the cluster.

Here is the XML code for the pingd clone resources:
The primitive tag for the pingd resource agent is nested in the clone tag. The clone tag contains a meta_attributes tag that contains an attributes tag. Three nvpair tags within the attributes tag tells Heartbeat that pingd should be started, how many instances of pingd should be started in the cluster, and how many instances of pingd should be started on each node.

Heartbeat needs to know what type of resource agent is being used to control the pingd resource. Since the resource agent is actually a script, Heartbeat needs to know the name of the script and where it is located. This information is provided with the primitive tag.

Here is the primitive for pingd:

```xml
<primitive id="MQHA_Pingd" class="ocf" type="pingd" provider="heartbeat">  
</primitive>
```
The meta_attributes tag provides information used by Heartbeat to clone the instances of pingd on the two nodes in the cluster.

Here is the meta_attributes tag for the pingd clone resource.

```xml
<meta_attributes id="MQHA_Pingd_Clone_Max_meta_attrs">
  <attributes>
    <nvpair name="target_role" id="MQHA_Pingd_Clone_Max_metaattr_target_role" value="started"/>
    <nvpair id="MQHA_Pingd_Clone_Max_metaattr_clone_max" name="clone_max" value="2"/>
    <nvpair id="MQHA_Pingd_Clone_Max_metaattr_clone_node_max" name="clone_node_max" value="1"/>
  </attributes>
</meta_attributes>
```

Pingd requires some information in order to perform the job of monitoring the gateway IP address. Three nvpairs provide the information required. They are contained in the attributes tag that is nested within the instanace_attributes tag.

```xml
<instance_attributes id="MQHA_Pingd_instance_attrs">
  <attributes>
    <nvpair name="host_list" value="192.168.70.1"/>
    <nvpair name="multiplier" value="10"/>
    <nvpair name="dampen" value="5s"/>
  </attributes>
</instance_attributes>
```
Name | Value | Description
--- | --- | ---
host_list | 192.168.70.1 | The list of IP addresses to be monitored. The gateway address is used in this solution. This should match your network’s gateway.
multiplier | 10 | A number used to help set a score that will trigger a failover when a problem is found.
dampen | 5 | The number of seconds to wait before updating the CIB after a ping.

**Configuring Constraints**

Constraints tell Heartbeat where and how resources are to be run. In this solution, two types of constraints are used; the order constraint and the location constraint. The order constraint controls the order in which actions are taken, for instance starting resource ‘a’ before starting resource ‘b’. The location constraint tells Heartbeat where a specific resource can be started.

An order constraint is used to ensure that resources are started in the proper order. Order constraints are configured using an rsc_order tag. The QM (MQHA_mqm) must not be started unless the Filesystem resource (MQHA_Filesystem) is started.

This is the order constraint:

```xml
<rsc_order id="order_MQHA" from="MQHA_mqm" to_action="start" to="MQHA_Filesystem" score="INFINITY" action="start" type="after"/>
```

How Heartbeat interprets the order constraint:

1. Start MQHA_mqm after starting MQHA_Filesystem
2. If MQHA_Filesystem cannot be started do not start MQHA_mqm
3. Stop MQHA_Filesystem after stopping MQHA_mqm
4. If MQHA_mqm cannot be stopped, do not stop MQHA_Filesystem

Location constraints are configured using the rsc_location tag. Nested within the rsc_location tag is a rule tag. The rsc_location tag has two name/value pairs. The first is “id” which provides a name that is associated with this constraint. The second is “rsc” which provides the ID of the resource this constraint applies to.
The rule tag has two name/value pairs and a nested expression tag. The first name/value pair is “id” which provides a name to be associated with this rule. The second name/value pair is “score” which provides a numeric value that is used by Heartbeat to make a decision on the status of the resource that this rule is associated with. In testing it was determined that a score of 100 caused Heartbeat to manage this resource in the proper manner.

The expression tag provides three name/value pairs. The first is “attribute”, which should be set to “#uname”. This setting indicates that the node name of the system is to be used. The second name/value pair is “operation”, which should be set to “eq”, indicating that the Boolean operator to be used is equal. The last name/value pair is “value”, which should be set to the node name of the system. In the example provided is the node name is “lac0030”.

The effect of this constraint is to tell Heartbeat that resource “MQHA_Filesystem” is allowed to run on node “lac0030” of the cluster.

In this solution there are two nodes, named lac0025 and lac0030. In order for all resource to be able to run on either system in the cluster, each resource needs an rsc_location tag for each node. In a cluster. With more than two nodes, an rsc_location tag would be needed for every node that a given resource could be run on.

In this solution resources MQHA_Filessystem, MQHA_mqm, and the group MQHA all require two location constraints to allow those resources to be started on the two nodes: lac0025 and lac0030.

The pingd resource runs as a clone resource. It requires a location constraint in order to run but it defines the resource it is associated with as MQHA, which is the group that all of the other resources belong to.
The pingd location constraint:

```xml
<rsc_location id="MQHA_Pingd-constraint" rsc="MQHA">
  <rule id="preferred_MQHA_Pingd-constraint" score="0">
    <expression attribute="MQHA_ping_set" id="pingd-defined"
      operation="defined"/>
  </rule>
</rsc_location>
```

The pingd resource is designed to run as a clone resource on multiple nodes, so it does not require that information to be provided by the location constraint.

**Using cibadmin to populate the CIB**

**Loading the resource definitions**

The cibadmin program reads a file containing XML code and loads the information contained in the code to CIB.

To load the resource configuration needed for this solution, copy and edit the XML code shown in the boxed example.

Note: When copying the text, select the entire contents of the boxed example. Footnotes will not be included in the copy. Be sure to copy all the text in the boxed example, even if the box spans a page break.
These are resource definitions for the MQHA group:

```xml
<group id="MQHA">
  <meta_attributes id="MQHA_meta_attrs">
    <attributes>
      <nvpair id="MQHA_metaattr_target_role" name="target_role" value="started"/>
    </attributes>
  </meta_attributes>
  <primitive class="ocf" type="Filesystem" provider="heartbeat" is_managed="true" id="MQHA_Filesystem">
    <instance_attributes id="MQHA_Filesystem_instance_attrs">
      <attributes>
        <nvpair name="device" value="/dev/disk/by-path/ccw-0.0.0301-part1"/>
        <nvpair name="directory" value="/MQHA"/>
        <nvpair name="fstype" value="ext3"/>
      </attributes>
    </instance_attributes>
    <operations>
      <op id="MQHA_Fileystem" name="monitor" interval="120" timeout="60" start_delay="0" disabled="false" role="Started"/>
    </operations>
  </primitive>
  <primitive id="MQHA_mqm" class="lsb" type="mqm" provider="heartbeat" is_managed="true">
    <operations>
      <op id="MQHA_mqm_monitor" name="monitor" interval="60" timeout="15" start_delay="0" disabled="false"/>
    </operations>
  </primitive>
  <primitive id="MQHA_IP_addr" class="ocf" type="IPaddr" provider="heartbeat">
    <instance_attributes id="MQHA_IP_addr_instance_attrs">
      <attributes>
        <nvpair name="ip" value="192.168.70.16"/>
      </attributes>
    </instance_attributes>
  </primitive>
</group>
```

1. Any of the id settings can be changed to a name that is preferred by the user.
2. The by-path-id of the shared disk device being used should be placed here.
3. The mount point can be changed to a name that is preferred by the user.
4. The IP address used for the user's WMQ QM should be used.
Copy the XML code shown, and paste it into a file named resource_configuration_file on one of your Linux systems in your Heartbeat cluster. The file should be modified as indicated in the footnotes. Then run the cibadmin command as shown below.

Syntax of cibadmin command being used to load resource definitions:
```
cibadmin -C -o resources -x resource_configuration_file
```
No messages are produced when cibadmin runs successfully.

To verify the integrity of the CIB after making an update, run the cmr_verify command.
```
cmr_verify -L
```
The –L flags indicates that crm_verify is to use the live CIB.

If no errors are found there will be no messages.

---

5 The IP address used for the user’s WMQ QM should be used.
Here is the XML code for the pingd clone resources:

```
<clone id="MQHA_Pingd_Clone_Max">
  <meta_attributes id="MQHA_Pingd_Clone_Max_meta_attrs">
    <attributes>
      <nvpair name="target_role" id="MQHA_Pingd_Clone_Max_metaattr_target_role" value="started"/>
      <nvpair id="MQHA_Pingd_Clone_Max_metaattr_clone_max" name="clone_max" value="2"/>
      <nvpair id="MQHA_Pingd_Clone_Max_metaattr_clone_node_max" name="clone_node_max" value="1"/>
    </attributes>
  </meta_attributes>
  <primitive id="MQHA_Pingd" class="ocf" type="pingd" provider="heartbeat">
    <instance_attributes id="MQHA_Pingd_instance_attrs">
      <attributes>
        <nvpair name="host_list" value="192.168.70.1"/>
        <nvpair name="multiplier" value="10"/>
        <nvpair name="dampen" value="5s"/>
      </attributes>
    </instance_attributes>
  </primitive>
</clone>
```

Cibadmin can add only one constraint at a time. If more than one constraint is placed in the xml files used as input to cibadmin, only the first one will be processed. This means that each constraint needed for this solution must be placed in a separate file. The table that follows shows a suggested file name for each constraint and the XML code that must be placed in the file. Copy and paste the test show into a file using the name indicated. The file can be placed on any Linux system in your Heartbeat cluster.

Any of the “id” values can be edited to use a name of the customer’s preference.

---

6 The host list should be modified to contain the IP address used for MQ at the customer’s location.

Copy the XML code shown and paste it into a file named pingd_configuration_file on one of your Linux systems in your Heartbeat cluster. The file should be modified as indicated in the footnotes. Then run the cibadmin command as shown below.

Syntax of cibadmin command being used to load resource definitions:

```
cibadmin -C -o resources -x pingd_configuration_file
```

No messages are produced when cibadmin runs successfully.

Run crm_verify to be sure that no configuration errors were made.
The node names provide in the “value” name/value pairs must be changed to match the nodes in the customer’s Heartbeat cluster. If more than two systems are configured in the cluster additional constraints may be needed.

<table>
<thead>
<tr>
<th>File to be created</th>
<th>XML code to be placed in file</th>
</tr>
</thead>
</table>
| fs1_constraint_file      | `<rsc_location id="cli-prefer-MQHA_Filesystem" rsc="MQHA_Filesystem">  
                             <rule id="prefered_cli-prefer-MQHA_Filesystem" score="100">  
                             <expression attribute="#uname" operation="eq" value="lac0025"/>  
                             </rule>  
                             </rsc_location>` |
| fs2_constraint_file      | `<rsc_location id="cli-standby-MQHA_Filesystem" rsc="MQHA_Filesystem">  
                             <rule id="prefered_cli-standby-MQHA_Filesystem" score="100">  
                             <expression attribute="#uname" operation="eq" value="lac0030"/>  
                             </rule>  
                             </rsc_location>` |
| mqha1_constraint_file    | `<rsc_location id="cli-standby-MQHA" rsc="MQHA">  
                             <rule id="prefered_cli-standby-MQHA" score="100">  
                             <expression attribute="#uname" operation="eq" value="lac0025"/>  
                             </rule>  
                             </rsc_location>` |
| mqha2_constraint_file    | `<rsc_location id="cli-standby-MQHA" rsc="MQHA">  
                             <rule id="prefered_cli-standby-MQHA" score="100">  
                             <expression attribute="#uname" operation="eq" value="lac0030"/>  
                             </rule>  
                             </rsc_location>` |
| mqm1_constraint_file     | `<rsc_location id="cli-prefer-resource_MQHA_mqm" rsc="MQHA_mqm">  
                             <rule id="prefered_cli-prefer-resource_MQHA_mqm" score="100">  
                             <expression attribute="#uname" operation="eq" value="lac0025"/>  
                             </rule>  
                             </rsc_location>` |
| mqm2_constraint_file     | `<rsc_location id="cli-standby-resource_MQHA_mqm" rsc="MQHA_mqm">  
                             <rule id="prefered_cli-standby-resource_MQHA_mqm" score="100">  
                             <expression attribute="#uname" operation="eq" value="lac0030"/>  
                             </rule>  
                             </rsc_location>` |
The following set of cibadmin commands can be used to process the eight constraint files.

```
cibadmin -C -o constraints -x fs1_constraint_file
cibadmin -C -o constraints -x fs2_constraint_file
cibadmin -C -o constraints -x mqha1_constraint_file
cibadmin -C -o constraints -x mqha2_constraint_file
cibadmin -C -o constraints -x mqm1_constraint_file
cibadmin -C -o constraints -x mqm2_constraint_file
cibadmin -C -o constraints -x pingd_constraint_file
cibadmin -C -o constraints -x order_constraint_file
```

When cibadmin runs successfully, it does not produce any messages.

Run crm_verify to be sure that no configuration errors were made.

At this point the solution should be operational.
Screen shot of the hb_gui showing the full configuration:
**Test Experience**

In our testing we found that in order for this solution to work, the message queue must be set to persistent.

Define a queue named TEMP.OUT.

An example of how to create a message queue:

```
mqm@LAC0030:/root/ih03> echo " define ql(TEMP.OUT)" | runmqsc csq1
```

Starting MQSC for queue manager csq1.

```
1 : define ql(TEMP.OUT)
```

AMQ8006: WebSphere MQ queue created.

One MQSC command read.

No commands have a syntax error.

All valid MQSC commands were processed.

Then make the message queue persistent:

```
mqm@LAC0030:/root/ih03> echo "alter ql(TEMP.OUT) defpsist(YES)" | runmqsc csq1
```

Starting MQSC for queue manager csq1.

```
1 : alter ql(TEMP.OUT) defpsist(YES)
```

AMQ8008: WebSphere MQ queue changed.

One MQSC command read.

No commands have a syntax error.

All valid MQSC commands were processed.
Test Summary

- 5,000 messages were inserted onto the queue.
- A capture program began processing the message in increments of 50.
- The system was crashed.

Linux-HA detected the failure and migrated the resource group.

- The whole process from crash to the resources being available again took less than one minute, as this log shows.

```plaintext
Feb 11 14:10:41 LAC0030 pengine: [4818]: notice: native_print: MQHA_Filesystem
(ocf::heartbeat:Filesysterm): Stopped
Feb 11 14:10:41 LAC0030 pengine: [4818]: notice: native_print: resource_MQHA_mqm
(heartbeat:mqm): Stopped
Feb 11 14:10:41 LAC0030 pengine: [4818]: notice: native_print: resource_MQHA_IP_addr
(ocf::heartbeat:IPaddr): Stopped
Feb 11 14:10:41 LAC0030 pengine: [4818]: debug: group_rsc_location: Processing rsc_location
prefered_cli-standby-MQHA for MQHA
Feb 11 14:10:41 LAC0030 pengine: [4818]: debug: group_rsc_location: Processing rsc_location
prefered_cli-prefer-MQHA for MQHA
Feb 11 14:10:41 LAC0030 pengine: [4818]: debug: native_assign_node: Assigning lac0030 to
MQHA_Filesystem
Feb 11 14:10:41 LAC0030 pengine: [4818]: debug: native_assign_node: Assigning lac0030 to
resource_MQHA_mqm
Feb 11 14:10:41 LAC0030 pengine: [4818]: debug: native_assign_node: Assigning lac0030 to
resource_MQHA_IP_addr
Feb 11 14:10:41 LAC0030 pengine: [4818]: notice: StartRsc: lac0030     Start MQHA_Filesystem
Feb 11 14:10:41 LAC0030 pengine: [4818]: notice: RecurringOp: Start recurring monitor (120s) for
MQHA_Filesystem on lac0030
Feb 11 14:10:41 LAC0030 pengine: [4818]: notice: StartRsc: lac0030     Start resource_MQHA_mqm
Feb 11 14:10:41 LAC0030 pengine: [4818]: notice: StartRsc: lac0030     Start resource_MQHA_IP_addr
```

- The message count at the crash was 1050. The capture program processed 3950 after the failover. No messages were lost.
- Similar testing was done with a simulated network failure with similar results.
- Testing was also done to prove that the queue manager is restarted after it was taken down manually. Again Heartbeat handled the failover properly.