Red Hat Enterprise MRG 2.0

Messaging Installation Guide

Installation information for the Messaging component of Red Hat Enterprise MRG

Lana Brindley

Alison Young

Red Hat Enterprise MRG 2.0 Messaging Installation Guide
Installation information for the Messaging component of Red Hat Enterprise MRG
Edition 1

Author	Lana Brindley	lbrindle@redhat.com
Author	Alison Young	alyoung@redhat.com

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

This book will show you how to download and install the MRG Messaging component of the Red Hat Enterprise MRG distributed computing platform. To learn how to program MRG Messaging applications, see *Programming in Apache Qpid*.

Preface

Red Hat Enterprise MRG

This book contains basic installation information for the MRG Messaging component of Red Hat Enterprise MRG. Red Hat Enterprise MRG is a high performance distributed computing platform consisting of three components:

1. *M*essaging — Cross platform, high performance, reliable messaging using the Advanced Message Queuing Protocol (AMQP) standard.
2. *R*ealtime — Consistent low-latency and predictable response times for applications that require microsecond latency.
3. *G*rid — Distributed High Throughput (HTC) and High Performance Computing (HPC).

All three components of Red Hat Enterprise MRG are designed to be used as part of the platform, but can also be used separately.

MRG Messaging

MRG Messaging is an open source, high performance, reliable messaging distribution that implements the Advanced Message Queuing Protocol (AMQP) standard. MRG Messaging is based on *Apache Qpid*[1].

This guide shows you how to install MRG Messaging and start the broker, and explains the basic options available. For more in depth explanation of MRG Messaging configuration you should refer to the *MRG Messaging User Guide*. If you want to write your own applications for use with MRG Messaging, you should also look at the *Programming in Apache Qpid* guide.

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[2] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

[1] http://cwiki.apache.org/qpid/
[2] https://fedorahosted.org/liberation-fonts/

> To see the contents of the file **my_next_bestselling_novel** in your current working directory, enter the **cat my_next_bestselling_novel** command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F2** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F1** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **mono-spaced bold**. For example:

> File-related classes include **filesystem** for file systems, **file** for files, and **dir** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.

> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at *http://access.redhat.com*. Through the customer portal, you can:

- search or browse through a knowledgebase of technical support articles about Red Hat products.
- submit a support case to Red Hat Global Support Services (GSS).
- access other product documentation.

Red Hat also hosts a large number of electronic mailing lists for discussion of Red Hat software and technology. You can find a list of publicly available mailing lists at *https://www.redhat.com/mailman/listinfo*. Click on the name of any mailing list to subscribe to that list or to access the list archives.

2.2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/* against the product **Red Hat Enterprise MRG.**

When submitting a bug report, be sure to mention the manual's identifier: *Messaging_Installation_Guide*

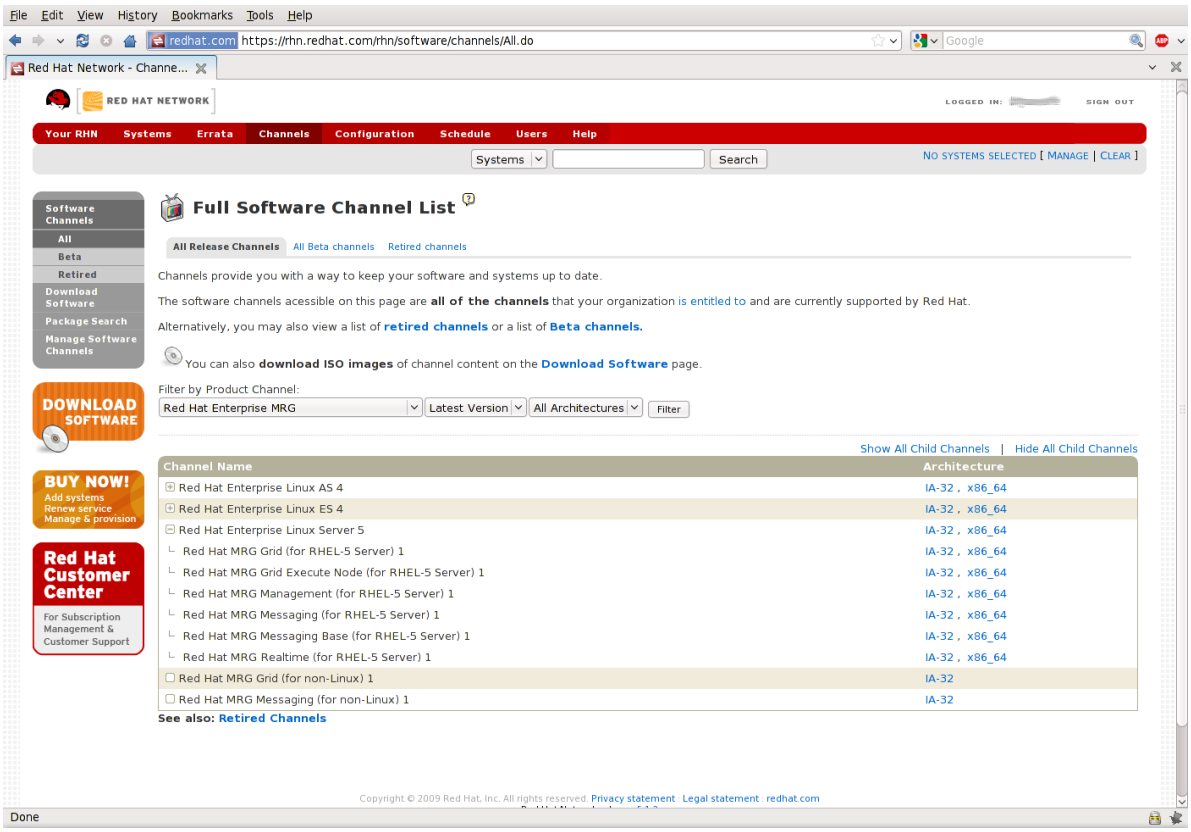
If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Installing MRG Messaging

In order to install MRG Messaging you will need to have registered your system with *Red Hat Network*[1] (RHN).

To see the available Red Hat Enterprise MRG channels available for MRG Messaging, log in to the Red Hat Network by going to *Red Hat Network login screen*[2] and entering your username and password. If you do not have an RHN account, you can find out how to get one by visiting the *RHN Login info screen*[3].

Red Hat Enterprise MRG channels are child channels of Red Hat Enterprise Linux. Select the appropriate Red Hat Enterprise Linux product to see the child channels.



Important

Before you install Red Hat Enterprise MRG check that your hardware and platform is supported. A complete list is available on the *Red Hat Enterprise MRG Supported Hardware Page*[4].

1.1. Installing MRG Messaging on Red Hat Enterprise Linux 5.6

1. Install the MRG Messaging group using the **yum** command.

[1] https://rhn.redhat.com/help/about.pxt
[2] https://www.redhat.com/wapps/sso/rhn/login.html
[3] https://rhn.redhat.com/rhn/sales/LoginInfo.do
[4] http://www.redhat.com/mrg/hardware/

```
# yum groupinstall "MRG Messaging"
```

Note

If you find that yum is not installing all the dependencies you require, make sure that you have registered your system with *Red Hat Network*[5].

1.2. Installing MRG Messaging on Client Machines

MRG Messaging can be run on machines that are client-only machines, that is, they do not run the broker. The packages required for client only machines are:

`qpid-cpp-client`
For C++ clients, **`qpid-cpp-client`** is required for runtime.

`qpid-cpp-client-devel`
For C++ clients that will be used for development, **`qpid-cpp-client-devel`** is required.

These packages might also have dependencies that will need to be resolved so that the packages can be successfully installed. Use **`yum deplist`** with the package name to discover the dependencies:

```
# yum deplist qpid-cpp-client
```

1.3. Available Packages — RPM

This section lists the RPM packages available for MRG Messaging..

Table 1.1. MRG Messaging Packages - i686 (AMD or Intel 32-bit)

RPM Package Name	Description	Language
`qpid-cpp-server`	MRG Messaging broker (Apache Qpid binaries for i386).	C++
`qpid-cpp-server-store`	MRG Messaging libraries, providing guaranteed message delivery.	C++
`qpid-cpp-client`	MRG Messaging client libraries. Required to run the broker.	C++
`qpid-cpp-client-devel`	C++ client libraries, including header files, developer documentation, and symbolic links to shared libraries.	C++

[5] https://rhn.redhat.com/help/about.pxt

Table 1.2. MRG Messaging Packages - x86_64 (AMD64 or Intel 64)

RPM Package Name	Description	Language
qpid-cpp-server	MRG Messaging broker (Apache Qpid binaries for x86).	C++
qpid-cpp-server-store	MRG Messaging libraries, providing guaranteed message delivery.	C++
qpid-cpp-client	MRG Messaging client libraries. Required to run the broker.	C++
qpid-cpp-client-devel	C++ client libraries, including header files, developer documentation, and symbolic links to shared libraries.	C++
qpid-cpp-client-devel-docs	C++ developer documentation.	C++

Table 1.3. MRG Messaging Packages - Java

RPM Package Name	Description	Language	Architecture
qpid-java-client	Java client library including JMS implementation	Java	Architecture Independent
qpid-java-common	Java client and broker (to be released) common library.	Java	Architecture Independent
qpid-java-example	Java examples	Java	Architecture Independent

Table 1.4. MRG Messaging Packages - Python

RPM Package Name	Description	Language	Architecture
python-qpid	Python client libraries and examples.	Python	Architecture Independent
qpid-tools	Python command line tools.	Python	Architecture Independent
python-qpid-qmf	Python libraries for the Qpid Management Framework.	Python	Architecture Independent

Table 1.5. MRG Messaging Packages - Plugins

RPM Package Name	Description
qpid-cpp-server-cluster	Clustering plugin
qpid-cpp-server-store	Persistence plugin
qpid-cpp-server-xml	XML exchange plugin
qpid-cpp-server-ssl	SSL Server plugin
qpid-cpp-client-ssl	SSL Client plugin
qpid-cpp-server-rdma	RDMA/Infiniband Server plugin
qpid-cpp-client-rdma	RDMA/Infiniband Client plugin

Programming Examples

Programming examples are installed by default with MRG Messaging. To look for specific files, use the **rpm** command:

```
$ rpm -qal python-qpid
```

- Python programming examples are installed with the **python-qpid** package. They are installed to **/usr/share/doc/python-qpid-0.10/examples**.
- C++ programming examples are installed with the **qpid-cpp-client-devel** package. They are installed to **/usr/share/qpidc/examples/messaging**.
- Java programming examples are installed with the **qpid-java-example** package. They are installed to **/usr/share/doc/qpid-java-0.10/examples**.

Starting the Broker

Qpid Initialisation Script

These actions are supported by *Qpid* initialisation scripts.

Table 2.1. Qpid Script Options

Qpid supported action	
start	Starts the service.
stop	Stops the service.
restart	If the service is already running, stop and restart the service. If the service isn't running, starts the service.
condrestart (and try-restart)	If the service is running, restarts the service. If the service isn't running, this action does nothing.
force-reload	Reloads the service configuration then restarts to service to ensure the configuration takes effect.
status	Prints the current status of the service.
usage	If this is run without an action, by default it will display a usage message listing all actions (intended for use).

Starting the C++ Broker

1. By default, the broker is installed in **/usr/sbin/**. If this is not on your path, you will need to type the whole path to start the broker:

```
$ /usr/sbin/qpidd -t
[date] [time] info Loaded Module: libbdbstore.so.0
[date] [time] info Locked data directory: /var/lib/qpidd
[date] [time] info Management enabled
[date] [time] info Listening on port 5672
```

   The **-t** or **--trace** option enables debug tracing, printing messages to the terminal.

Important

When starting the broker, it will inform you that it has locked a data directory. This data directory is used for persistence, which is enabled by default in MRG Messaging. For more information about persistence see *Chapter 4, Persistence*

2. To stop the broker, type **CTRL**+**C** at the shell prompt

```
[date] [time] notice Shutting down.
[date] [time] info Unlocked data directory: /var/lib/qpidd
```

3. For production use, MRG Messaging is usually run as a service. To start the broker as a service, run the following command as the root user:

```
# service qpidd start
Starting Qpid AMQP daemon:                                 [  OK  ]
```

4. You can check on the status of the service using the **`service status`** command and stop the broker with the **`service stop`**.

```
# service qpidd status
qpidd (pid PID) is running...

# service qpidd stop
Stopping Qpid AMQP daemon:                                 [  OK  ]
```

Running multiple brokers on a single machine

To run more than one broker on a single machine, they must run on different ports and use different directories for the journals.

1. Select two available ports, for example *5555* and *5556*.

2. Start each new broker, using the **`--data-dir`** command to specify a new data directory for each:

```
$ qpidd -p 5555 --data-dir /tmp/qpid/store/1
```

```
$ qpidd -p 5556 --data-dir /tmp/qpid/store/2
```

Options for Running the Broker

The broker can be run with a number of options. An overview of the most common options are given here.

3.1. Setting Broker Options

Setting Options Using the Configuration File

1. For options that persist across sessions, you can put the options in the configuration file. Become the root user, and open the **/etc/qpidd.conf** file in your preferred text editor to make the necessary changes.

2. This example uses the configuration file to enable debug tracing. Changes will take effect from the next time the broker is started and will be used in every subsequent session.

```
# Configuration file for qpidd
trace=1
```

3. If you are running the broker as a service, you will need to restart the service once you have made the changes.

```
# service qpidd restart
Stopping qpidd daemon:                [  OK  ]
Starting qpidd daemon:                [  OK  ]
```

4. If you are *not* running the broker as a service, you can now start the broker as normal.

```
# /usr/sbin/qpidd -t
[date] [time] info Locked data directory: /var/lib/qpidd
[date] [time] info Management enabled
[date] [time] info Listening on port 5672
```

Setting Options Using the Command Line

To set options for a single instance, add the option to the command line when you start the broker.

1. This example uses command line options to start the broker with debug tracing. These options will need to be explicitly stated every time the broker is run.

```
$ /usr/sbin/qpidd -t
```

Common Options

For more options, type **man qpidd** or **/usr/sbin/qpidd --help** at the shell prompt.

Table 3.1. General Broker Options

General options for running the broker	
-t	This option enables verbose log messages, for debugging only.

General options for running the broker	
-p <Port_Number>	Instructs the broker to use the specified port. Defaults to port 5672. It is possible to run multiple brokers simultaneously by using different port numbers.
-v	Displays the installed version.
-h	Displays the help message.

Table 3.2. Options for running the broker as a service (daemon)

Options for running the broker as a service	
-d	This option instructs MRG Messaging to run in the background as a daemon. Log lessages from the broker are sent to **syslog** (**/var/log/messages**) by default.
-q	This command shuts down the broker that is currently running.
-c	This command checks if the daemon is already running. If it is running, it returns the process ID number.
-d --wait=<seconds>	This sets the maximum wait time (in seconds) for the daemon to initialize. If the daemon has not successfully completed initialization within this time, an error is returned. This option must be used in conjunction with the **-d** option, or it will be ignored.

3.2. Using Modules with the Broker

MRG Messaging installs several modules by default, which are automatically loaded when the broker is started. The module directory for the client is located at **/usr/lib/qpid/client** (or **/usr/lib64/qpid/client** on 64-bit installations) and for the daemon at **/usr/lib/qpid/daemon** (or **/usr/lib64/qpid/daemon** on 64-bit installations). The default modules are:

- SSL
- Authorization (ACL enforcement)
- RDMA (Infiniband)
- XML exchange type
- Persistence
- Clustering

All these modules are server side only, with the exception of the SSL and RDMA modules, which have both client and server side plugins. More information on working with these modules can be found in the *MRG Messaging User Guide*.

Table 3.3. Options for using modules with the broker

Options for using modules with the broker	
--load-module *MODULENAME*	Instructs the broker to use the specified module as a plug-in.
--module-dir <DIRECTORY>	Causes the broker to use a different module directory.
--no-module-dir	Causes the broker to ignore module directories.

Getting Help with Modules

To see the help text for modules, use the **--help** command:

```
# /usr/sbin/qpidd --help
```

3.3. Logging Broker Errors

By default, log output is sent to **stderr** if the broker is run on the command line, or to **syslog** (**/var/log/messages/**), if the broker is run as a service. You can also choose to send them to a file.

If the broker is started with the **-d** or **--daemon** options, it will log to syslog by default. If the broker is started as a service, it will log to **STDERR** by default instead.

Table 3.4. Logging Options

Options for logging with syslog	
--log-to-stderr yes|no	Send logging output to **stderr**. Enabled by default when run from command line.
--log-to-stdout yes|no	Send logging output to **stdout**.
--log-to-file *FILE*	Send log output to the specified filename. *FILE*.
--log-to-syslog yes|no	Send logging output to **syslog**. Enabled by default when run as a service.
--syslog-name *NAME*	Specify the name to use in **syslog** messages. The default is **qpidd**.
--syslog-facility *LOG_XXX*	Specify the facility to use in **syslog** messages. The default is **LOG_DAEMON**.

3.4. Running the JMS client with Realtime Java

To achieve more deterministic behavior, the JMS Client can be run in a Realtime Java environment.

1. The client must be run on a realtime operating system, and supported by your realtime java vendor. Red Hat supports only Sun and IBM implementations.
2. Place the realtime .jar files provided by your vendor in the classpath.
3. Set the following JVM argument:

```
-Dqpid.thread_factory="org.apache.qpid.thread.RealtimeThreadFactory"
```

This ensures that the JMS Client will use **`javax.realtime.RealtimeThread`**s instead of **`java.lang.Thread`**s.

Optionally, the priority of the Threads can be set using:

```
-Dqpid.rt_thread_priority=30
```

By default, the priority is set at 20.

4. Based on your workload, the JVM will need to be tuned to achieve the best results. Refer to your vendor's JVM tuning guide for more information.

Persistence

A persistence library allows MRG Messaging to store messages and queue configuration, ready to be reloaded in the event of machine or network failure. When the persistence store module is loaded, it allows messages and other persistent state information to be recovered when a broker is restarted.

In order for messages to be stored the persistence store must be loaded. The **`--store-dir`** option specifies the directory used for for the persistence store and any configuration information. The default directory is **`/var/lib/qpidd`**. See *Table 4.1, "Persistence Options"* for options on how to change this behaviour.

In addition to loading the persistence store, queues and messages also need to be identified as *durable*. This can be done in the client application or by using the **`qpid-config`** command line tool. See the *Programming in Apache Qpid* guide for more information about creating client applications.

Important

If the persistence module is not loaded, messages and the broker state will not be stored to disk, even if the queue and messages sent to it are marked persistent.

Table 4.1. Persistence Options

Persistence Options	
`--data-dir` ***`DIRECTORY`***	Specifies the directory for data storage and log files generated by the broker. The default is **`/var/lib/qpidd`**.
`--no-data-dir`	Disables storage of configuration information and other data. If the default directory at **`/var/lib/qpidd`** exists, it will be ignored.
`--num-jfiles` ***`NUMBER`***	Set the number of files for each instance of the persistence journal. The default is 8.
`--jfile-size-pgs` ***`NUMBER`***	Set the size of each journal file in multiples of 64KB. The default is 24.
`--wcache-page-size` ***`NUMBER`***	The size (in KB) of the pages in the write page cache. Allowable values must be powers of 2 (1, 2, 4, ... 128). Lower values will decrease latency but also decrease throughput. The default is 32.

Note

Persistence is dealt with in more depth in the *MRG Messaging User Guide*

Command line utilities

MRG Messaging contains a number of command line utilities for monitoring and configuring messaging brokers.

Note

A graphical interface is also available for download from the Red Hat Enterprise MRG yum repository, see the *MRG Management Console Installation Guide* for more information on this tool.

`qpid-config`
Display and configure exchanges, queues, and bindings in the broker

`qpid-route`
Display and configure broker federation, including routing and links between brokers

`qpid-tool`
Access configuration, statistics, and control within the broker

`qpid-queue-stats`
Monitor the size and enqueue/dequeue rates of queues in a broker

`qpid-stat`
Display details and statistics for various broker objects.

`qpid-printevents`
Subscribes to events from a broker and prints details of events raised to the console window.

`qpid-cluster`
Display information about cluster membership, stop one or all members in a clean fashion. See *MRG Messaging User Guide* for more details on clustered brokers.

`qpid-cluster-store`
Used in recovering persistent data after a non-clean cluster shutdown. See Messaging User Guide for more details.

To install the command line utilities follow the installation instructions in *Chapter 1, Installing MRG Messaging* and install the **`qpid-tools`** package.

Note

For more information on the command line utilities see the *MRG Messaging User Guide*

Clustering and federation

Messaging Clusters

A *Messaging Cluster* is a group of brokers that act as a single broker. Changes on any broker are replicated to all other brokers in the same Messaging Cluster, so if one broker fails, its clients can fail-over to another broker without loss of state. The brokers in a Messaging Cluster may run on the same host or on different hosts. Two brokers are in the same cluster if

1. They use the same OpenAIS **mcastaddr**, **mcastport**, and **bindnetaddr**, and
2. They use the same cluster name.

Messaging Clusters are implemented using using OpenAIS, which provides a reliable multicast protocol, tools, and infrastructure for implementing replicated services. You must install and configure OpenAIS to use MRG broker groups. Once you have installed OpenAIS, configure MRG Messaging to run in a cluster as follows.

1. Set the binding address for openais in /etc/ais/openais.conf. Use **ifconfig** to find the inet addr and the netmask for the interface you want:

```
# ifconfig
eth0  Link encap:Ethernet  HWaddr 00:E0:81:76:B6:C6
      inet addr:10.16.44.222  Bcast:10.16.47.255  Mask:255.255.248.0
      inet6 addr: fe80::2e0:81ff:fe76:b6c6/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:35914541 errors:6 dropped:0 overruns:0 frame:6
      TX packets:6529841 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:20294124383 (18.9 GiB)  TX bytes:12925473031 (12.0 GiB)
      Interrupt:98 Base address:0x8000
```

   The binding address in **/etc/ais/openais.conf** should be the network address for the interface, which you can find by doing a bitwise **AND** of the inet addr (in this case, 10.16.44.222) and the network mask (in this case, 255.255.248.0). The result is 10.16.40.0. As a sanity check, you can use **route** and make sure the address you computed is associated with the interface:

```
$ /sbin/route
Kernel IP routing table
Destination   Gateway       Genmask        Flags Metric Ref Use Iface
20.0.10.0     *             255.255.255.0  U     0      0     0 eth1
192.168.122.0 *             255.255.255.0  U     0      0     0 virbr0
10.16.40.0    *             255.255.248.0  U     0      0     0 eth0
169.254.0.0   *             255.255.0.0    U     0      0     0 eth1
default       10.16.47.254  0.0.0.0        UG    0      0     0 eth0
```

   To use **eth0** as the interface for the cluster, find the setting for bindnetaddr in **/etc/ais/openais.conf**, and set it to 10.16.40.0:

```
bindnetaddr: 10.16.40.0
```

2. Make sure that the primary group for the user running qpidd is “ais”. For instance, if you are running qpidd as a daemon, the user is named **qpidd**. You can make **ais** the primary group for **qpidd** as follows:

```
# usermod -g ais qpidd
```

3. Set the name of the cluster in qpidd.conf.

```
cluster-name="Mick"
```

4. You can use **qpid-cluster** to list the membership of the cluster or to shutdown cleanly.

```
[conway@mrg32 ~]$ qpid-cluster mrg33
  Cluster Name: conway-test-cluster
Cluster Status: ACTIVE
  Cluster Size: 3
       Members: ID=20.0.100.33:22689
URL=amqp:tcp:20.0.10.33:5672,tcp:10.16.44.238:5672,\
tcp:20.0.100.33:5672,tcp:192.168.122.1:5672
               : ID=20.0.100.34:20810
URL=amqp:tcp:20.0.10.34:5672,tcp:10.16.44.239:5672,\
tcp:20.0.100.34:5672,tcp:192.168.122.1:5672
               : ID=20.0.100.35:20139
URL=amqp:tcp:20.0.10.35:5672,tcp:20.0.20.35:5672,tc\
p:10.16.44.240:5672,tcp:20.0.100.35:5672,tcp:192.168.122.1:5672
```

You can also run **qpid-tool** against any cluster node to view details of the cluster. The cluster is one of the objects shown by the **list** command.

```
qpid: list
Management Object Types:
ObjectType                           Active  Deleted
=====================================================
com.redhat.rhm.store:journal         1       0
com.redhat.rhm.store:store           1       0
org.apache.qpid.broker:binding       5       0
org.apache.qpid.broker:broker        1       0
org.apache.qpid.broker:connection    1       0
org.apache.qpid.broker:exchange      7       0
org.apache.qpid.broker:queue         2       0
org.apache.qpid.broker:session       1       0
org.apache.qpid.broker:system        1       0
org.apache.qpid.broker:vhost         1       0
org.apache.qpid.cluster:cluster      1       0
```

To see the properties of the cluster, use **show cluster**:

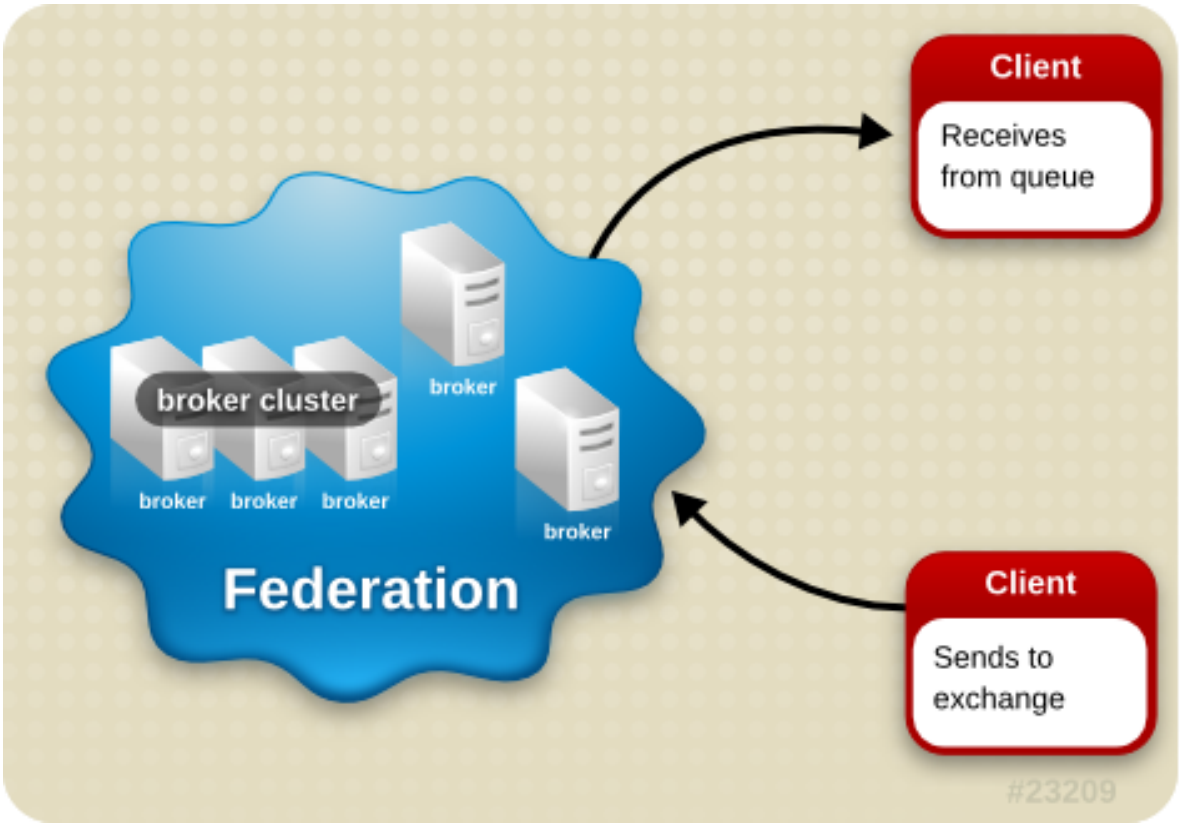
```
qpid: show cluster
Object of type org.apache.qpid.cluster:cluster: (last sample time: 13:56:40)
Type      Element       112
=============================================================================================
property  brokerRef     102
property  clusterName   foo
property  clusterID     da821ff9-2a88-4002-b976-f18680556290
property  publishedURL
 amqp:tcp:10.16.44.222:52265,tcp:20.0.10.15:52265,tcp:192.168.122.1:52265
property  clusterSize   1
property  status        ACTIVE
```

```
property  members
 amqp:tcp:10.16.44.222:52265,tcp:20.0.10.15:52265,tcp:192.168.122.1:52265
```

Messaging Clusters can be used together with Red Hat Clustering Services (RHCS) by starting brokers with the **`--cluster-cman`** option.

Federation

Federation is used to provide geographical distribution of brokers. A number of individual brokers, or clusters of brokers, can be federated together. This allows client machines to see and interact with the federation as though it were a single broker. Federation can also be used where client machines need to remain on a local network, even though their messages have to be routed out.



Federation is used primarily for connecting disparate locations across a wide area network. Full connectivity across an enterprise can be achieved while keeping local message traffic isolated to a single location. Departmental brokers can be specified with individual policies that control inter-departmental message traffic flow.

Some applications can benefit from having a broker co-resident with the client. This is good for situations where the client produces data that must be delivered reliably but connectivity can not be guaranteed. In this case, a co-resident broker provides queueing and durability that is not available in the client on its own.

Federation bridges disjointed IP networks. Message brokers can be configured to allow message connectivity between networks where there is no IP connectivity. For example, an isolated, private IP network can have messaging connectivity to brokers in other outside IP networks.

Links and routes

Federation is configured through a series of links and routes.

A *link* is a connection between two brokers that allows messages to be passed between them. A link is a transport level connection (using a protocol such as TCP, RDMA, or SSL) that is initiated by a broker and accepted by another broker. The broker that initiates the link is considered the client in the connection. The broker that receives that connection will not treat it any differently from any other client connection, other than annotating it as being for federation.

Routes are the paths that messages take from one broker to another, and can run along one or more links to the final destination. A route is associated with an AMQP session established over the link connection. A route controls the flow of messages across the link between brokers, and multiple routes can share the same link. Messages will flow over a single route in only one direction. For bi-directional connectivity a pair of routes must be created, one for each direction of message flow. Routes always consist of a session and a subscription for consuming messages. Depending on the configuration, a route can have a private queue on the source broker with a binding to an exchange on that broker.

Note

Clustering and federation are dealt with in more depth in the *MRG Messaging User Guide*

Authentication and Authorization

MRG Messaging uses Simple Authentication and Security Layer (SASL) for identifying and authorizing incoming connections to the broker, as mandated in the AMQP specification. SASL provides a variety of authentication methods. MRG Messaging clients (with the exception of the JMS client) and the broker use the *Cyrus SASL library*[1] to allow for a full SASL implementation.

Important

The **PLAIN** authentication method sends passwords in cleartext. If using this mechanism, for complete security using Security Services Library (SSL) is recommended. See the *MRG Messaging User Guide* for information on setting SSL on client machines.

Note

To use SSL in python Qpid clients using a version earlier than Python 2.6, you need to install the *python-ssl* package from the Extra Packages for Enterprise Linux (EPEL) repository.

Enabling and Using SASL Plain Authentication

To use the default SASL PLAIN authentication mechanism implemented by the MRG Messaging client libraries, either use the default username and password of *guest*, which are included in the database at **/var/lib/qpidd/qpidd.sasldb** on installation, or add your own accounts.

1. Add new users to the database by using the **saslpasswd2** command. The User ID for authentication and ACL authorization uses the form ***user-id*@*domain*.**.

   Ensure that the correct realm has been set for the broker. This can be done by editing the configuration file or using the **-u** option. The default realm for the broker is *QPID*.

   ```
   # saslpasswd2 -f /var/lib/qpidd/qpidd.sasldb -u QPID new_user_name
   ```

2. Existing user accounts can be listed by using the **-f** option:

   ```
   # sasldblistusers2 -f /var/lib/qpidd/qpidd.sasldb
   ```

   Note

   The user database at **/var/lib/qpidd/qpidd.sasldb** is readable only by the qpidd user. If you start the broker from a user other than the qpidd user, you will need to either modify the configuration file, or turn authentication off.

3. To switch authentication on or off, use the **auth yes|no** option when you start the broker:

[1] http://cyrusimap.web.cmu.edu/

```
# /usr/sbin/qpidd --auth yes

# /usr/sbin/qpidd --auth no
```

You can also set authentication to be on or off by adding the appropriate line to to the **/etc/qpidd.conf** configuration file:

```
auth=no

auth=yes
```

4. The SASL configuration file is in **/etc/sasl2/qpidd.conf** for Red Hat Enterprise Linux 5.6.

   For information on using a different configuration, use your web browser to view the Cyrus SASL documentation at **/usr/share/doc/cyrus-sasl-lib-2.1.22/index.html** for Red Hat Enterprise Linux 5.6.

Using ACL

1. The ACL module is loaded by default. You can check that it is loaded by running the **qpidd --help** command and checking the output for ACL options:

```
$ qpidd --help
...[output truncated]...
ACL Options:
--acl-file FILE (policy.acl)  The policy file to load from, loaded from data dir
```

2. To start using the ACL, you will need to specify the file to use. This is done by using the **--acl-file** command with a path and filename. The filename should have a **.acl** extension:

```
$ qpidd --acl-file ./aclfilename.acl
```

You can now view the file with the **cat** command and edit it in your preferred text editor. If the path and filename is not found, **qpidd** will fail to start.

Note

For more information on authentication and authorization see the *MRG Messaging User Guide*

Infiniband

MRG Messaging connections can use Infiniband, which provides high speed point-to-point bidirectional serial links that can be faster and have much lower latency than TCP connections.

The machines running the server and client must each have Infiniband properly installed. In particular:

- The kernel driver and the user space driver for your Infiniband hardware must both be installed.

  See the *Knowledge Base article*[1] for a list of hardware supported on RHEL 4 and RHEL 5, and the kernel and user space drivers associated with each.

- Allocate lockable memory for Infiniband.

  By default, the operating system can swap out all user memory. Infiniband requires lockable memory, which can not be swapped out. Each connection requires 8 Megabytes (8192 bytes) of lockable memory.

  To allocate lockable memory, edit **`/etc/security/limits.conf`** to set the limit, which is the maximum amount of lockable memory that a given process can allocate.

- The Infiniband interface must be configured to allow IP over Infiniband. This is not used for message transport, but is nevertheless needed for RDMA connection management.

To enable Infiniband on the Qpid server:

- Make sure that the package **`qpid-cpp-server-rdma`** has been installed for Qpid to use RDMA. Then make sure that the RDMA plugin, **`rdma.so`**, is present in the plugins directory.

- Allocate lockable memory for Infiniband.

  For example, if the user running the server is qpidd, and you wish to support 64 connections (64*8192=524288), add these entries:

```
qpidd soft memlock 524288
qpidd hard memlock 524288
```

To use Infiniband in a Qpid messaging client:

- Make sure that the package **`qpid-cpp-client-rdma`** has been installed. If it has, the file **`rdmaconnector.so`** will be present in the plugins directory.

- Allocate lockable memory for Infiniband.

  To set a limit for all users, for example supporting 16 connections (16*8192=32768), add this entry:

```
* soft memlock 32768
```

  If you want to set a limit for a particular user, use the UID for that user when setting the limits:

```
andrew soft memlock 32768
```

Windows Software Development Kit

The MRG Messaging Windows software development kit (WinSDK) is used for developing MRG Messaging applications on a Microsoft Windows system, and contains a set of libraries, header files, example code, and documentation. It supports native C++ (unmanaged) code and .NET (managed) code in any supported .NET language.

The WinSDK kit is distributed as a set of directories, as follows:

- **`\bin`**

  Precompiled binary (**`.dll`** and **`.exe`**) files and the associated debug program database (**`.pdb`**) files

  Boost library files

  Microsoft Visual Studio 2008 **`MSVC90`** runtime library files

- **`\include`**

  A directory tree of **`.h`** files

- **`\lib`**

  The linker **`.lib`** files that correspond to files in **`/bin`**

- **`\docs`**

  Apache Qpid C++ API Reference

- **`\examples`**

  A Visual Studio solution file and associated project files to demonstrate using the WinSDK in unmanaged C++

- **`\dotnet_examples`**

  A Visual Studio solution file and associated project files to demonstrate using the WinSDK in C#

- **`\management`**

  A python scripting code set for generating QMF data structures

Note

For more information about Qpid QMF visit *The Apache Qpid Wiki*[1]

9.1. WinSDK Installation

Installing the WinSDK on 32-bit Windows systems

Important

Before you install the WinSDK check that your hardware and platform is supported. A complete list is available on the *Red Hat Enterprise MRG Supported Hardware Page*[2].

The WinSDK kits are distributed as **.zip** files. The kit contents are copied to your system using file copy operations.

1. Unzip the **qpid-cpp-winsdk-1.3.0.24-x86.zip** file into your preferred directory.
2. The **qpid-cpp-winsdk-1.3.0.24-x86.zip** kit contains a copy of the Microsoft C++ 2008 Redistributable Package (x86). This package is located in the **\bin** directory after the kit has been unzipped. No further installation is necessary.

Installing the WinSDK on 64-bit Windows systems

Important

Before you install the WinSDK check that your hardware and platform is supported. A complete list is available on the *Red Hat Enterprise MRG Supported Hardware Page*[3].

The WinSDK kits are distributed as **.zip** files. The kit contents are copied to your system using file copy operations.

1. Unzip the **qpid-cpp-winsdk-1.3.0.24-x64.zip** file into your preferred directory.
2. The **qpid-cpp-winsdk-1.3.0.24-x64.zip** file does not contain the Microsoft C++ 2008 Redistributable Package (x64). On 64-bit systems this package must be installed using a formal installation process. See *the Microsoft Download Center*[4] for information about getting and installing this package.

Note

Microsoft provides *Release* versions of redistributable packages of the C++ runtime libraries for all platform architectures, but do not allow redistribution of *Debug* versions of the C++ runtime libraries.

If you require *Debug* versions of the C++ runtime libraries, you will need to get them by installing an end-user development platform, such as Microsoft Visual Studio or Microsoft Visual Studio Express.

[2] http://www.redhat.com/mrg/hardware/
[3] http://www.redhat.com/mrg/hardware/

Important

The MRG Messaging WinSDK has been built using Microsoft Visual Studio 2008. All native C ++ component libraries have been linked to the specific C++ runtime libraries that come with Microsoft Visual Studio 2008. You must use Microsoft Visual Studio 2008 when compiling example projects or when developing your own code. Executable programs created using Microsoft Visual Studio 2010 that link to this version of the MRG Messaging WinSDK could experience problems due to multiple conflicting versions of the C++ runtime libraries.

9.2. WinSDK Usage

Building the CPP Examples

1. Open the **`examples\examples.sln`** solution file in Visual Studio.
2. When prompted, select the platform architecture the match the development target.
3. Build the solution.

Table 9.1. CPP Examples

Example	Description
Server	Creates a receiver and listens for messages. Upon message receipt the message content is converted to upper case and forwarded to the received message's *`ReplyTo`* address.
Client	Sends a series of messages to the server and prints the original message content and the received message reply content.
Map_receiver	Creates a receiver and listens for a map message. Upon message receipt the message is decoded and displayed on the console.
Map_sender	Creates a map message and sends it to the map_receiver. The map message contains string, integer, floating point, list, and UUID values.
Spout	Spout is a more complex example of code that generates a series of messages and sends them to the peer program Drain. Flexible command line arguments allow the user to specify a variety of message and program options.
Drain	Drain is a more complex example of code that receives a series of messages and displays their contents on the console.

Building the C# Examples

1. Open the **`dotnet_examples\winsdk_dotnet_examples.sln`** solution file in Visual Studio.
2. When prompted, select the platform architecture the match the development target.
3. Build the solution.

Note

The WinSDK MRG Messaging Managed Callback Library is built specifically for .NET Framework v2.0 and can be included in any project using v2.0 or higher. No other MRG Messaging libraries have a dependency on a .NET Framework version number.

Table 9.2. C# Examples

Example	Description
csharp.example.server	Creates a receiver and listens for messages. Upon message receipt the message content is converted to upper case and forwarded to the received message's *`ReplyTo`* address.
csharp.example.client	Sends a series of messages to the server and prints the original message content and the received message reply content.
csharp.map.receiver	Creates a receiver and listens for a map message. Upon message receipt the message is decoded and displayed on the console.
csharp.map.sender	Creates a map message and sends it to the map_receiver. The map message contains values for every supported MRG Messaging data type.
csharp.example.spout	Spout is a more complex example of code that generates a series of messages and sends them to the peer program Drain. Flexible command line arguments allow the user to specify a variety of message and program options.
csharp.example.drain	Drain is a more complex example of code that receives a series of messages and displays their contents on the console.
csharp.map.callback.receiver	Creates a receiver and listens for a map message. Upon message reception the message is decoded and displayed on the console. This example illustrates the use of the C# managed code callback mechanism provided by the MRG Messaging Managed Callback Library
csharp.map.callback.sender	Creates a map message and sends it to the map_receiver. The map message contains values for every supported MRG Messaging data type.
csharp.example.declare_queues	A program to illustrate creating objects on a broker. This program creates a queue used by spout and drain.
csharp.direct.receiver	Creates a receiver and listens for a messages. Upon message receipt the message is decoded and displayed on the console.
csharp.direct.sender	Creates a series of messages and sends them to **`csharp.direct.receiver`**.

Example	Description
csharp.example.helloworld	A program to send a message and to receive the same message from a broker.

The C++ and .NET C# example programs are designed to demonstrate how the two language environments interoperate. In test cases, running the C++ and .NET C# examples will produce the same results at the test console.

This table describes which code functions are supported by equivalent code examples:

Table 9.3. Equivalent Examples

Program Function	C++ Code Example	.NET C# Code Example
client	C++ client	csharp.example.client
server	C++ server	csharp.example.server
map_receiver	C++ map_receiver	csharp.map.receiver
map_sender	C++ map_sender	csharp.map.sender
spout	C++ spout	csharp.example.spout
drain	C++ drain	csharp.example.drain

More Information

Reporting a Bug

If you have found a bug in MRG Messaging, follow these instructions to enter a bug report:

1. You will need a *Bugzilla*[1] account. You can create one at *Create Bugzilla Account*[2].
2. Once you have a Bugzilla account, log in and click on *Enter A New Bug Report*[3].
3. When submitting a bug report, you will need to identify the product (Red Hat Enterprise MRG), the version (2.0), and whether the bug occurs in the software (component = messaging) or in the documentation (component = Messaging_Installation_Guide).

Further Reading

Red Hat Enterprise MRG and MRG Messaging Product Information

http://www.redhat.com/mrg

MRG Messaging and other Red Hat Enterprise MRG manuals

http://docs.redhat.com/docs/en-US/index.html

http://www.redhat.com/mrg/resources/

Red Hat Knowledgebase

https://access.redhat.com/knowledge/search

Appendix A. Revision History

Revision 1-0 Thu Jun 23 2011 **Alison Young** *alyoung@redhat.com*

Prepared for publishing

Revision 0.1-6 Mon Jun 20 2011 **Alison Young** *alyoung@redhat.com*

Rebuilt for docs stage

Revision 0.1-5 Fri Jun 17 2011 **Alison Young** *alyoung@redhat.com*

Rebuilt for docs stage

Revision 0.1-4 Thu Jun 16 2011 **Alison Young** *alyoung@redhat.com*

BZ#704546 - tech review updates

Revision 0.1-3 Fri May 27 2011 **Alison Young** *alyoung@redhat.com*

Technical Review fixes

Revision 0.1-2 Wed May 18 2011 **Alison Young** *alyoung@redhat.com*

BZ#704546 - Technical review fixes

Revision 0.1-1 Tue Mar 01 2011 **Alison Young** *alyoung@redhat.com*

BZ#628516 - Chapter 2

Revision 0.1-0 Tue Feb 22 2011 **Alison Young** *alyoung@redhat.com*

Fork from 1.3