GEMSTONE®

GEMFIRE®

ENTERPRISE

System Administrator's Guide

Version 6.5

September 2010

Send comments on this guide to docs@gemstone.com

INTELLECTUAL PROPERTY OWNERSHIP

This documentation is furnished for informational use only and is subject to change without notice. GemStone Systems, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this documentation.

This documentation, or any part of it, may not be reproduced, displayed, photocopied, transmitted, or otherwise copied in any form or by any means now known or later developed, such as electronic, optical, or mechanical means, without express written authorization from GemStone Systems, Inc.

Warning: This computer program and its documentation are protected by copyright law and international treaties. Any unauthorized copying or distribution of this program, its documentation, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted under the maximum extent possible under the law.

The software installed in accordance with this documentation is copyrighted and licensed by GemStone Systems, Inc. under separate license agreement. This software may only be used pursuant to the terms and conditions of such license agreement. Any other use may be a violation of law.

Use, duplication, or disclosure by the Government is subject to restrictions set forth in the Commercial Software - Restricted Rights clause at 52.227-19 of the Federal Acquisitions Regulations (48 CFR 52.227-19) except that the government agency shall not have the right to disclose this software to support service contractors or their subcontractors without the prior written consent of GemStone Systems, Inc.

This software is provided by GemStone Systems, Inc. and contributors “as is” and any expressed or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall GemStone Systems, Inc. or any contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

COPYRIGHTS

PATENTS

GemFire is protected by U.S. patent 6,360,219. Additional patents pending.

TRADEMARKS

GemStone Systems, Inc.
1260 NW Waterhouse Avenue, Suite 200
Beaverton, OR 97006

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

About This Guide

This guide describes the system administration functions required to install, deploy, and manage the GemStone® GemFire Enterprise® product, which is typically used for distributed caching and data distribution. The GemFire Java® API reference (Javadocs) lists all of the interfaces and methods supplied for building GemFire Enterprise applications. It is recommended that you use the reference pages for specific interface and method information.

The document assumes familiarity with basic Java terminology and programming practices.

How This Documentation Is Organized

This documentation contains the following sections:

- Chapter 1, *Product Installation*, on page 19 lists system requirements, tells how to install GemFire, and shows how to obtain and install the GemFire licenses.
- Chapter 2, *Overview of GemFire System Administration*, on page 29 tells how to set up your GemFire Enterprise system.
- Chapter 3, *Configuring the System*, on page 41 tells how to interpret and set the GemFire configuration attributes.
- Chapter 4, *Configuring Member Discovery and Communication*, on page 59 explains how to configure members of a GemFire Enterprise distributed system to find each other.
- Chapter 5, *Security*, on page 73 describes the configuration, authentication, and authorization of applications, clients, and servers and their operations in a distributed system.
- Chapter 6, *Managing Disk Stores*, on page 95 describes the mechanism used by regions and server and gateway queues for storing data to disk.
- Chapter 7, *Administering the Distributed System*, on page 123 details some operations required to administer a GemFire Enterprise distributed system.
- Chapter 8, *Monitoring and Tuning the Distributed System*, on page 139 describes the methods for analyzing and improving the performance of a GemFire system.
- Chapter 9, *Using JMX to Administer GemFire*, on page 167 tells how to use the JMX Agent to administer and manage a single GemFire Enterprise distributed system.
- Chapter 10, *GemFire System Logging*, on page 185 tells how to confirm system configuration

and debug problems in configuration and code.

- Chapter 11, *Troubleshooting and System Recovery*, on page 195 describes the types of faults that a distributed system may encounter, and suggests strategies for dealing with them.
- Appendix A, *The gemfire Command-line Utility*, on page 227 provides syntax and other reference information for the `gemfire` command-line utility.
- Appendix B, *System Statistics*, on page 235 provides information on GemFire Enterprise standard statistics for caching and distribution activities.

Typographical Conventions

This document uses the following typographical conventions:

- Methods, types, file names and paths, code listings and prompts are shown in `Courier New` typeface. For example:

  ```
  gfPut
  ```

- Parameters and variables are shown in *`italic`* font. For example,

  ```
  gfConnect(sysDir, connectionName, writeProtectAllowed)
  ```

- In examples showing both user input and system output, the lines you type are distinguished from system output by **`boldface`** type:

  ```
  prompt> gemfire
  ```

- If you are viewing this document online, the page, section, and chapter references are hyperlinks, like this reference to *Installing and Uninstalling GemFire Enterprise* on page 23 and this reference to Chapter 2, *Overview of GemFire System Administration*, on page 29. Blue text denotes a hyperlink.

Other Useful Documents

The *GemFire Enterprise online Java API documentation* can be accessed through the file `index.html` in the GemFire `docs` directory.

The *GemFire Enterprise Developer's Guide* describes the major services and functions in GemFire Enterprise.

The *Visual Statistics Display* guide describes how to use the Visual Statistics Display (VSD) tool, which is used to analyze archived historical data. Contact GemStone Technical Support for instructions about acquiring VSD.

Technical Support

GemStone provides several sources for product information and support. The *GemFire Enterprise Developer's Guide*, the *GemFire Enterprise System Administrator's Guide* and the GemFire Enterprise online Java API provide extensive documentation, and should always be your first source of information. GemStone Technical Support engineers will refer you to these documents when applicable. However, you may need to contact Technical Support for the following reasons:

- Your technical question is not answered in the documentation.
- You receive an error message that directs you to contact GemStone Technical Support.
- You want to report a bug.
- You want to submit a feature request.

Questions concerning product availability, pricing, license keyfiles, or future features should be directed to your GemStone account manager.

Preserving Artifacts for Technical Support

If you have a hung VM and you do not have to kill it, leave it while you contact Technical Support. If you cannot leave the VM running, and it is running under Unix, signal it twice with this command, letting five to ten seconds pass between the two signals:

```
kill -QUIT pid
```

This will send the VM's stack dumps into the log file for inspection. For Windows systems, call Technical Support for assistance in obtaining stack dumps.

Don't delete any files until you call Technical Support and find out exactly what data may be useful to Support or Engineering. Save all the artifacts, including:

- Log files. Send the full log to Technical Support, not just the stack. Even at the default logging level, the log contains data that may be important, such as the operating system and license.
- Statistics archive files.
- Core files.
- For Linux, you can use `gdb` to extract a stack from a core file. Call Technical Support if you need assistance.
- Crash dumps.
- For Windows, save the Dr. Watson output.

Contacting Technical Support

The GemStone support site at http://techsupport.gemstone.com provides all the information you need to contact our technical support team. If you are unable to access the website for any reason, you can e-mail technical support at techsupport@gemstone.com.

When contacting GemStone technical support, please be prepared to provide the following information:

- Your name, company name, and GemFire license number
- The GemFire product and version you are using
- The hardware platform and operating system you are using
- A description of the problem or request
- Exact error messages received, if any
- Any artifacts in the preceding list

Your GemStone support agreement may identify specific individuals who are responsible for submitting all support requests to GemStone. If so, please submit your information through those individuals. All responses will be sent to authorized contacts only.

24x7 Emergency Technical Support

GemStone offers, at an additional charge, 24x7 emergency technical support. This support entitles customers to contact us 24 hours a day, 7 days a week, 365 days a year, if they encounter problems that cause their production application to go down, or that have the potential to bring their production application down. Contact your GemStone account manager for more details.

Training and Consulting

Consulting and training for all GemStone products are available through GemStone's Professional Services organization.

- Training courses are offered periodically at GemStone's offices in Beaverton, Oregon, or you can arrange for onsite training at your desired location.
- Customized consulting services can help you make the best use of GemStone products in your business environment.

Contact your GemStone account representative for more details or to obtain consulting services.

Chapter 1

Product Installation

This chapter covers system requirements and licensing for GemFire Enterprise® systems, and describes the product tree. Systems that meet the requirements described here are suitable for installing GemFire and beginning development, but additional system resources may be necessary to support large applications.

In this chapter:

> *If you are upgrading an existing GemFire Enterprise installation, be sure to check the Release Notes for upgrade and migration instructions.*

To contact GemStone Technical Support:

- On the web: http://techsupport.gemstone.com
- By e-mail: support@gemstone.com
- By phone: 800/243-4772 or 503/533-3503

1.1 System Requirements

Supported Configurations

GemFire Enterprise® 6.5 runs with full capabilities and with Level A support on the following OS/Java platforms.

If you want to run GemFire on a platform other than those listed in the two tables, contact your GemStone sales representative. GemStone will evaluate whether it can support the platform, either as-is or under a special agreement.

GemFire Enterprise can also be installed to run in pure Java mode on any standard Java platform with some functional differences. See *Running GemFire in Pure Java Mode* on page 22.

Table 1.1 32-bit Platforms: Supported OS/Java Configurations

Operating System	Sun Java SE 1.6.0_17	BEA JRockit Java SE 1.5.0_19 B27.6.5	BEA JRockit Java SE 1.6.0_14 R27.6.5	IBM J9 1.5.0 build 2.3 20091104	IBM J9 1.6.0 build 2.4 20091214
Solaris 9	X				
Solaris 10	X				
Red Hat EL 4	X				
Red Hat EL 5	X	X	X	X	X
SLES 10	X	X	X	X	X
Windows XP SP3*	X	X	X	X	X
Windows 2003 Server SP2*	X	X	X	X	X

Table 1.2 64-bit Platforms: Supported OS/Java Configurations

Operating System	Sun Java SE 1.6.0_17	BEA JRockit Java SE 1.5.0_19 B27.6.5	BEA JRockit Java SE 1.6.0_14 R27.6.5	IBM J9 1.5.0 build 2.3 20091104	IBM J9 1.6.0 build 2.4 20091214
Solaris 10	X				
Red Hat EL 4	X	X	X	X	X
Red Hat EL 5	X	X	X	X	X
SLES 10	X	X	X	X	X
Windows 2003 Server SP2 (pure Java)*	X	X	X		

*The Microsoft Loopback Adapter is not supported.

GemFire Dependencies on Linux RPM Packages

This table lists the RPM package dependencies for the Linux EL 5 distributions. The i386 or i686 after the package names indicates that you must install the package for that particular architecture, regardless of the native operating system architecture. The packages listed are available on the default media for each distribution.

Linux Version	glibc	libgcc
Red Hat Enterprise Linux (EL) Server release 5 (i686)	glibc	libgcc
Red Hat Enterprise Linux (EL) Server release 5 (x86_64)	glibc (i686)	libgcc (i386)

For versions of Linux not listed in the table, you can verify that you meet the GemFire dependencies at the library level with this ldd command:

```
prompt> ldd <path to GemFire product dir>/lib/libgemfire.so
```

These libraries are external dependencies of the native library (libgemfire.so or libgemfire64.so). Check that the output of ldd includes all of these:

```
libdl.so.2
libm.so.6
libpthread.so.0
libc.so.6
libgcc_s.so.1
```

For details on the `ldd` tool, see the online Linux man page for `ldd`.

Running GemFire in Pure Java Mode

GemFire Enterprise can run on platforms not listed in *Supported Configurations* on page 20, with some functional differences. This is called running in pure Java mode, meaning GemFire runs without the GemFire native code.

In this mode, distributed system members still have access to GemFire's caching and distribution capabilities, but the following features may be disabled:

- Operating system statistics. Platform-specific machine and process statistics such as CPU usage and memory size.
- Access to the process ID. Only affects log messages about the application. The process ID is set to "0" (zero) in pure Java mode.

In addition, in pure Java mode, the cache server startup and shutdown are handled in a different way than when running with the native code. If the cache server is shut down in an abnormal way, the next startup may require manual intervention. In pure Java mode, the cache server startup script checks for the presence of a `.cacheserver.ser` file, and the server will not start if the file exists. The `.cacheserver.ser` file is automatically generated by the cache server when it starts, and it is automatically deleted when the server closes properly. An abnormal server termination may prevent `.cacheserver.ser` from being deleted, so subsequent attempts to start a cache server fail because the file exists. If the server terminates abnormally, delete the `.cacheserver.ser` file for the abnormally terminated server so another cache server can start.

Host Machine Requirements

Requirements for each host:

- File system that supports long file names.
- Adequate per-user quota of file handles (ulimit for Solaris and Linux)
- TCP/IP.
- System clock set to the correct time.
- For each Unix and Linux host, hostname and hosts files that are properly configured; see the system manpages for hostname and hosts.
- Time synchronization service such as NTP.

  For troubleshooting, you must run a time synchronization service on all hosts. Synchronized time stamps allow you to merge log messages from different hosts, for an accurate chronological history of a distributed run.

Documentation Requirements

- To view the online Java documentation, your web browser must support style sheets and frames.
- To view the HTML version of the manuals, your web browser must have JavaScript enabled.
- To view the PDF version of the manuals, you must have Adobe Acrobat Reader. You can download a free copy at http://www.adobe.com/products/acrobat/readstep.html.

1.2 Installing and Uninstalling GemFire Enterprise

The GemFire Enterprise installation is provided in two files: an installer and an installation instructions file named `INSTALL.txt`. A single installer installs GemFire on all platforms. After installing the product, `INSTALL.txt` is also included in the `docs` directory.

To install the product, obtain the installer file and the instructions file from your GemStone salesperson or from the GemStone website at http://www.gemstone.com/download. To access the website, you need to enter your login and password. If you are new to GemStone products, you will need to register and create a login and password to get into the download center.

GemFire comes bundled with an evaluation license that you can use to run a distributed system with up to three members and with three clients for any server you run. This license never expires. For greater functionality, you can get development and production licenses from GemStone.

You can uninstall GemFire Enterprise by deleting the entire product tree.

After you install GemFire Enterprise, see *Running Your GemFire Applications* on page 50 of the *GemFire Enterprise Developer's Guide*.

1.3 GemFire Licenses

GemFire has three types of licenses:

- Evaluation licenses are platform-independent and never expire. GemFire comes bundled with an evaluation license that you can use to run a distributed system with up to three members and with three clients for any server you run.
- Development licenses are used only for development and testing.
- Production licenses are usually node-locked and limited to a fixed number of CPUs. Other licensing models can be negotiated.

When you purchase GemFire, you may receive both production and development licenses. When GemStone generates a GemFire license for development or production, it is valid for a specific machine or a subnet (a specified list of machines) on which users can run GemFire. The license is further restricted to a maximum number of CPUs that you specify when purchasing the product.

License Type Configuration

The initial installation of GemFire has the `gemfire.properties` `license-type` configuration attribute set to `evaluation`. To use a development or production license, you modify the `license-type` attribute to `development` or `production` for all members of the distributed system. This helps avoid inadvertently starting a development process inside a production system. An application attempting to join a distributed system with a license type not equal to the type already in use fails to connect, with an error like this:

```
Could not connect using a "development" license because the existing
distributed system node "host/port" is using a "production" license.
```

Obtaining and Installing Production and Development Licenses

Follow these instructions to buy new licenses and update invalid licenses.

- You need to buy licenses for all machines where you will run GemFire for development and production.
- You may need to replace an existing license if CPUs are added to a host machine or if the machine's identity changes. If a license becomes invalid, running GemFire processes are unaffected, but new GemFire processes cannot run on the machine. You can request a change to the current license by following these instructions just for the machine whose information has changed.

Before you begin, install GemFire on every machine where you want to run GemFire processes.

- On each machine where you will run GemFire:
  - Run the `gemfire license` command and pipe or cut-and-paste the output to a file.
  - Edit the file, filling in the information requested. For descriptions of the license attributes, see Table 1.3 on page 25. This is sample output from the command:

Example 1.1 gemfire license Command Output

```
% gemfire license
#Data Needed to Obtain License for current machine.
#Mon Aug 09 12:20:02 PDT 2010
license-version=3.0
platform=Linux
customer-id=<determined by GemStone>
customer-name=<Please supply a customer name here>
```

```
node=hoom.gemstone.com 192.80.250.60
date=2010/08/09 12 oclock PM, PDT
native-node=00-1h-a0-27-cd-58
cpus=4
product=GemFire
purchased-cpus=<determined by GemStone>
group-id=<determined by GemStone>
Java version:   6.5 build 29218 08/06/2010 17:18:30 PDT javac 1.5.0_17
Native version: 6.5  06/02/2010 11:16:48 PDT optimized i386 Linux 2.4.21-
47.EL
Running on: doom/10.80.250.60, 4 cpu(s), i386 Linux 2.6.18-8.el5xen

-------------------- Contents of
jar:file:/export/doom2/users/build/gf65sancout/product/lib/gemfire.jar!/gemf
ireLicense.zip --------------------
product=GemFire
  customer-name=Embedded Evaluation
    customer-id=20100801
      group-id=1
        platform=ANY
          license-type=evaluation
            ========== embedded.evaluation.license ==========
            License signature is valid.
            License never expires.
            License is valid on any node.
            License had no native node limits.
            License limits cache servers to "3" clients.
            License limits distributed system to "3" members.
            License allows an unlimited number of cpus.
            NOTICE: use of GemFire with this license key is only permitted
in a non-production environment and for the period, if any, limited by the
license key.  Notwithstanding any other provision in the EULA, this
Evaluation License of GemFire is provided AS-IS without support or warranty
of any kind, expressed or implied.
```

- Send the files via e-mail to keyfiles@gemstone.com, with subject line “GemFire license request” and indicating whether this is a new request or request for upgrade. GemStone will respond by sending you a single GemFire license file, `gemfireLicense.zip`, which specifies all authorized uses.

  Do not unzip the GemFire license file.

- Place the license file on each machine in the top-level GemFire installation directory.
- Set the license type for your development and production work areas in the `gemfire.properties` setting, `license-type`.

License Attributes

This table lists the attributes whose values are displayed as part of the `gemfire license` output.

Table 1.3 License Attributes

Attribute	Description
`client-limit`	Number of clients that can connect to a GemFire cache server. If set to zero, no clients are allowed. If not set, an unlimited number of clients are allowed.

Attribute	Description
`customer-id`	Unique id, assigned by GemStone, that can be any string. All the customer's licenses should have the same value for this property. All members of a distributed GemFire system must have the same `customer-id`.
`customer-name`	Symbolic name to describe your organization. Can be any string, but should be consistent with past naming.
`cpus`	If specified, causes the license to only work on machines whose CPU count is less than or equal to this value. If unspecified then there is no CPU limit. If specified, the value must be a whole number greater than zero.
`date`	Absolute point in time after which the license is no longer valid.
`group-id`	An integer, initially `1`. This value is incremented whenever your organization changes an existing license or deletes a license. All members of a distributed GemFire system must have the same `group-id`.
`license-type`	Type of license: `evaluation`, `development`, or `production`. All members of a distributed GemFire system must have the same `license-type`. For details on `license-type`, see *License Attributes* on page 25.
`license-version`	License version supported by the product to be licensed. (This does not necessarily correspond to the product version.)
`member-limit`	Maximum number of system members allowed in any distributed system instance. This attribute is set to `unlimited` for non-evaluation licenses. Most evaluation licenses have a three-member limit, but you can contact GemStone to get an expanded evaluation license. If set to zero, there is no limit. If set to -1, members run in isolation, with no peers. The -1 setting is generally used for client licenses.
`native-node`	Locks the license to a single machine node. To obtain native node information for this platform, run the `gemfire license` command on the machine where you intend to lock the license. When running GemFire in pure Java mode, this attribute has the value `pureJavaMode`.
`node`	Locks the license to one or more nodes based on IP address or host name. For a list of all legal values, run the `gemfire license` command on the machine where you intend to lock the license. If the value of this property is a list of IP addresses or host names, the license is valid on any of the specified nodes. If an IP address or host name contains asterisk (`*`) characters, the license is valid with any IP address or host name that matches the pattern. For example: `10.80.10.*`
`platform`	Operating system on which GemFire is licensed to run: `Windows`, `Solaris`, `Linux`, or `ANY` (for a platform-independent license).
`product`	The actual product that is licensed. For this version, this is always GemFire.
`purchased-cpus`	Number of cpu licenses actually purchased for the machine GemFire is running on. This number may belower than the CPU limit specified in the `cpus` attribute (above).

1.4 GemFire Product Tree

Directory Name	Contents
`bin`	GemFire executables and setup scripts. **`agent`** – script for starting the GemFire JMX agent **`cacheserver`** – script for starting the cacheserver process **`gemfire`** – command-line administrative utility *There are two versions of these scripts that are compatible with the operating system where GemFire is installed. For example, shell scripts on Unix systems or batch files on Windows.*
`defaultConfigs`	Generic configuration files for use by application programmers
`docs`	See `docs/index.html` for descriptions and links to the GemFire examples and the documentation.
`dtd`	XML DTDs for GemFire configuration files. For details, see the discussion of cache configuration through the `cache.xml` file in *GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide*.
`lib`	GemFire `JAR` files and shared libraries
`templates`	Java programming templates for using product features.
`gemfireLicense.zip`	The GemFire license file. This file is not included automatically in the GemFire product. You must place it in a location where the product can find it, usually the GemFire product home directory. For details, see *GemFire Licenses* on page 24 and *GemFire Configuration and Deployment Files* on page 36. The GemFire license files are provided in a zip file. You do not need to unzip this file. *To avoid corrupting your GemFire license information, be certain not to change the contents of the zip file.*

GemFire Product Documentation

The documentation for your GemFire products is provided in PDF and HTML formats and can be viewed with any frames-capable web browser. These guides can be accessed through the index page in the GemFire `docs` directory:

Windows	*productDir*`\docs\index.html`
Linux	*productDir*`/docs/index.html`

- *Release Notes* describes differences between the current release and previous product releases.
- *GemFire Enterprise System Administrator's Guide* (this document) provides guidance for installing, configuring, monitoring, and tuning your GemFire Enterprise installation.
- *GemFire Enterprise Developer's Guide* describes the major services and functions for Java programs in the GemFire Enterprise product.
- *Early Access Features* describes functionality provided as part of early access programs, with specific, targeted customers in mind.

In addition, the GemFire *Java API reference documentation* provides information on the interfaces, classes, and methods in the GemFire Application Programmer's Interface.

Chapter 2

Overview of GemFire System Administration

This chapter introduces the basics of GemFire Enterprise® system administration, including the configuration files, how to start and stop your system members, tools for monitoring system operation, and system tuning. These topics are covered in detail in the rest of the book.

In this chapter:

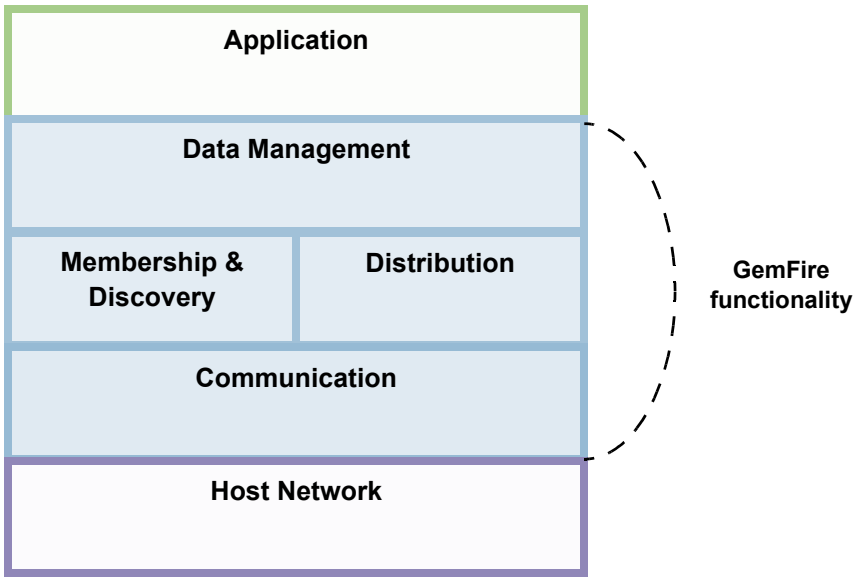
2.1 Architecture of a GemFire Distributed System

The GemFire Enterprise product is a library of classes that provide application developers with ready-made data distribution and data management capability.

Functional Overview

Conceptually, GemFire sits between the application-specific functionality and the underlying hosts and networks on which the applications run, providing the three middle layers shown here.

Figure 2.1 Functional Layers of the GemFire Enterprise Distributed System



Data Management

The GemFire data management layer determines how data is stored, retrieved, and made available to an application with a cache. This layer accesses and synchronizes data across multiple applications and application instances, and it provides scaling, data distribution, and data sharing.

Membership and Discovery

The distributed GemFire system defines how the application processes find each other. Application processes join to form a distributed system by specifying the same methods (protocol, address, and port) for discovery. The membership and discovery facility keeps track of its membership list and makes the members aware of the identities of the other members in the distributed system. For client/server installations, the discovery facility keeps track of servers and their current load status, providing clients with the locations of least-loaded servers.

Data Distribution and Notification

GemFire data distribution and notification provides applications with a facility for distributing data and consuming the distributed data pushed to the application. As data in a cache gets updated, notifications are sent to other applications that have registered interest in these updates. Different levels of cache consistency can be selected. Distribution operations can wait for acknowledgement from other caches before continuing, or they can return without waiting for a response. Caches can be replicated or not. Data can also be partitioned across many hosts. Applications can run without caching data, receiving

callbacks when data on the cache server changes and passing that information through to other receivers without the overhead of caching.

Communication

The communication layer facilitates distribution using connection and connectionless protocols, providing messaging in either mode. The choice of distribution protocol is independent from the discovery protocol. For discovery, IP multicast is the default protocol, and TCP/IP with a GemFire locator process is also an option. For distribution, the default protocol is TCP, and IP unicast is also an option. In addition, you can choose IP multicast for targeted data operations distribution.

Table 2.1 Network Protocol Options for Discovery and Distribution

	TCP/IP	IP Unicast	IP Multicast
Discovery	Option (locator)	N/A	Default
Distribution	Default	Option	Option (for targeted data operations)

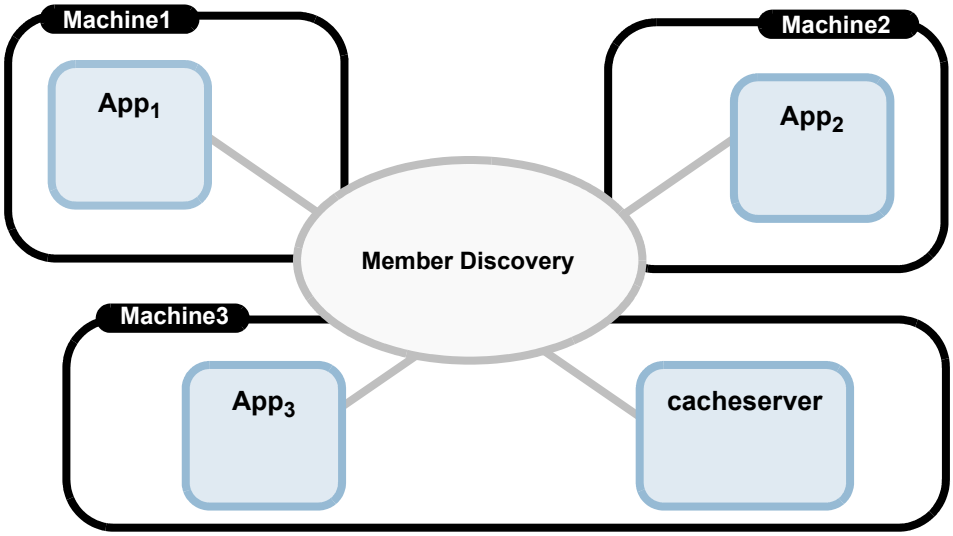
Operational Overview

Because GemStone customers compile the GemFire libraries into their applications, a running GemFire system mainly consists of application processes. In addition, GemFire provides these optional tools:

- `cacheserver`—Cache servers are long-lived, configurable members generally used to host long-lived data independent of the applications or to run servers in a hierarchical cache (see *Configuring and Running the GemFire Cache Server* on page 126).
- `locator`—At sites that use TCP for discovery, locators enable distributed GemFire processes to contact each other (see Chapter 4, *Configuring Member Discovery and Communication*, on page 59).

This is a high-level view of the processes forming a distributed system. In a running system, the membership list is dynamic. Applications can start up at specific times in the work flow and then shut down when they are done.

Figure 2.2 Members Joining a Distributed System



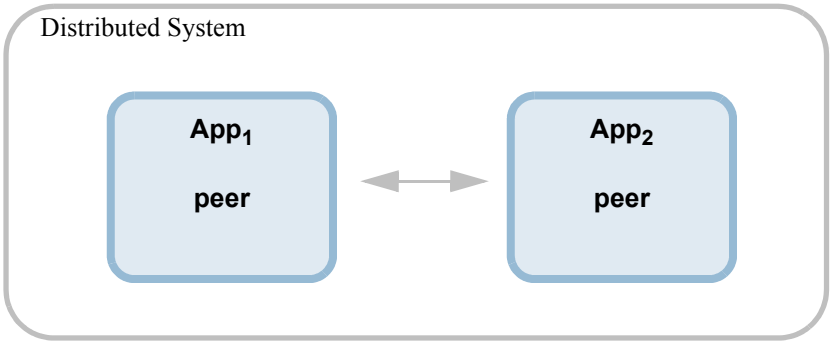
The Member Discovery *piece shown here is usually provided by GemFire locator processes.*

The operation of a running system varies, depending on the configuration. GemFire Enterprise supports peer-to-peer, client/server, and multi-site architectures.

Peer-to-peer— This figure shows a simple GemFire distributed system with two application processes that are peers to one another and can share data with each other. In peer-to-peer systems like this,

whenever a member joins or leaves the distributed system, all applications receive membership notifications.

Figure 2.3 GemFire Peer-to-Peer Data Distribution



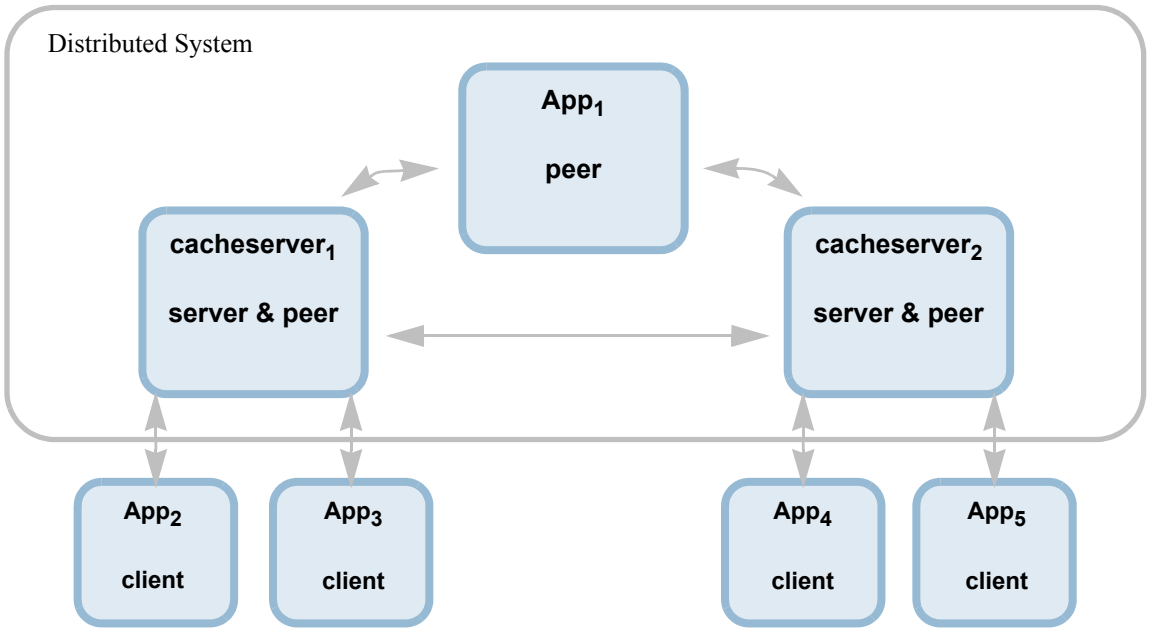
Client/server—In a client/server configuration, clients and servers are organized into separate distributed system. All the servers are in the same distributed system while the clients may belong to one distributed system or to separate systems.

Servers operate among themselves and with peer applications in standard peer-to-peer fashion. At the same time, servers provide services for their clients. Usually, cache servers are configured to operate as servers and their regions are configured to be replicas. If you are running the `cacheserver` executable shipped with the GemFire software, it always operates as a server.

Client applications connect to one or more servers in their server group and interact primarily with the servers. If a server shuts down, its clients fail over to another server.

The following figure shows a client/server system. The client applications App_2, App_3, App_4 and App_5 interact primarily with a cache server. In addition, the two cache server processes and the application process App_1 are peers to one another. Like any peers, these three processes share data and receive membership notification when one of the three joins or leaves the system.

Figure 2.4 GemFire Client/Server Data Distribution

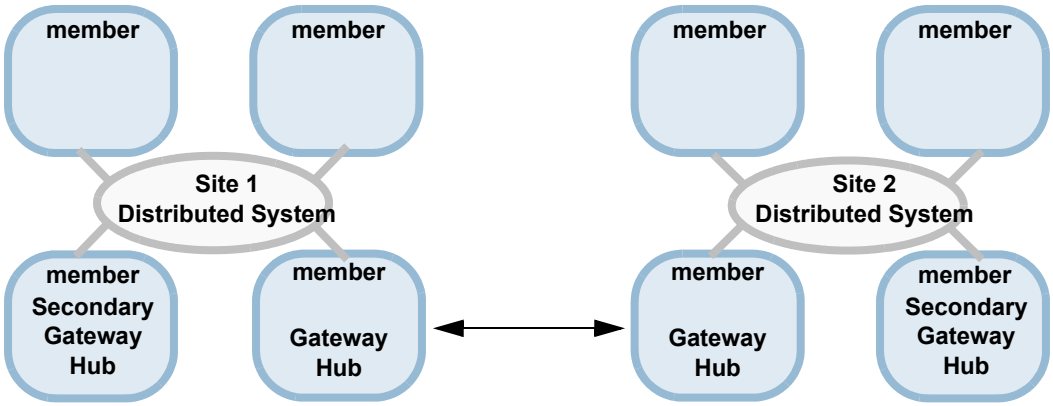


Multi-site—.In a multi-site configuration (generally run across a WAN) distributed systems are configured to communicate with one another through specially-configured gateways members. Each system is its own distinct distributed system and often each system also acts as a server system in a client/server configuration.

The members within each distributed system operate among themselves in standard peer-to-peer fashion. Additionally, the gateway members distribute cache operations to the remote distributed system sites and receive distributions from them.

This figure shows a distributed multi-site system composed of two distributed systems. Normally, there are multiple members configured as gateways to ensure uninterrupted communication between sites. In addition, the members of each single site are peers to one another. Like any peers, these processes share data and receive membership notification when one of the others joins or leaves the system.

Figure 2.5 GemFire Multi-site Data Distribution



2.2 GemFire Configuration and Deployment Files

These files are used for GemFire Enterprise deployment and system configuration. Only the license is required.

- **gemfire.properties**—This file contains the settings required to join a distributed system. Configuration includes system member discovery, communication parameters, security, logging, and statistics. Distributed system membership is described in *Configuring Member Discovery and Communication* on page 59. For a detailed list of the parameters in the `gemfire.properties` file, see *System Properties in the gemfire.properties File* on page 48.

  Each member has its own `gemfire.properties` file, which is usually placed in the working directory where the process runs. For other possible file locations, see *Specifying the Configuration File Locations* on page 42. The application software may include a set of `gemfire.properties` files that you can edit. If not, you create the files.

- **gemfireLicense.zip**—This is the license file provided to you by GemStone that allows you to use the product. Do not unzip this file. For more information, see *GemFire Licenses* on page 24.

- **cache.xml**—This is the declarative cache configuration file. This file contains XML declarations for cache, region, and region entry configuration, and it is chiefly of interest to application developers. Some of the parameters are needed for system administration, however, such as the settings for configuring disk store files. For a detailed list of the parameters in the `cache.xml` file, see *GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide*.

For more information, see Chapter 3, *Configuring the System*.

2.3 GemFire Output Files

GemFire Enterprise can create three kinds of output files: log files, statistics archive files, and data persistence files. All of these files are optional.

- **Log files**—GemFire Enterprise provides comprehensive logging messages to help you confirm system configuration and to debug problems in configuration and code. Log file behavior is configured for the system member in the `gemfire.properties` file. For details, see Chapter 10, *GemFire System Logging*.
- **Statistics archive files**—The GemFire Enterprise installation includes standard statistics for caching and distribution activities, which can be archived on disk. Statistics gathering is configured for the system member in the `gemfire.properties` file. For details, see Appendix B, *System Statistics*, on page 235 and the `archive-disk-space-limit` (page 48) and `archive-file-size-limit` (page 48) properties.
- **Disk store files**—Disk store files are used to hold persistent and overflow data from the cache. Regions can be configured to persist data to disk for backup purposes or overflow to disk to control memory usage. The subscription queues that servers use to send events to clients can be overflowed to disk. Gateway queues are overflowed to disk and can be persisted. The configuration uses the `cache.xml`. For details, see Chapter 6, *Managing Disk Stores*, on page 95.

2.4 Startup and Shutdown

The procedure for starting or stopping your GemFire distributed system depends on your system's configuration. The startup sequence is determined by dependencies between the processes to be started.

If you have locators, at least one locator must start first. For the application and cache server processes, you need to work out logical startup and shutdown sequences in terms of your distributed system's management structure. Generally, you start cache servers first and start dependent processes of all kinds last.

Use the gemfire command-line tool to shut down your system. It provides the most orderly shutdown and, if you persist data to disk, gives you the most efficient startup the next time you run your system.

For more information, see *Starting and Stopping the Distributed System* on page 124.

2.5 Management Tools

The `com.gemstone.gemfire.admin` Java API, the JMX Agent, and the `gemfire` command-line utility allow you to view and modify configuration attributes for distributed systems and individual system members. These tools store configuration attributes in files. You can copy and modify these configuration files as needed for individual system members.

For information about...	See...
Configuration attributes	Chapter 3, *Configuring the System*, on page 41
The JMX Agent	Chapter 9, *Using JMX to Administer GemFire*, on page 167
The `gemfire` Command	Appendix A, *The gemfire Command-line Utility*, on page 227
The `admin` API	*Developing System Administration Tools* on page 479 of the *GemFire Enterprise Developer's Guide*

2.6 Tools for Monitoring and Analyzing System Operation

In addition to your usual network monitoring software and OS-level commands, you use the following to monitor and analyze your system operations.

Log Files

A GemFire distributed system produces logs for applications, cache servers, and locators.

- **Applications and cache servers -** To create log files, you must set the `log-file` attribute in the process's `gemfire.properties` file. Otherwise, the messages go to `stdout`. These log files can be placed anywhere that is convenient for monitoring. For applications, these log files contain entries from GemFire operation only.
- **Locator -** The locator always creates a log file in its working directory. This logging is not configurable.

Reconstructing a sequence of events on the distributed system can mean synchronizing many log files on many machines. Be sure to run a time synchronization service on every machine.

For more information, see Chapter 10, *GemFire System Logging*. For suggested responses to conditions identified in the log files, see Chapter 11, *Troubleshooting and System Recovery*.

GemFire Statistics

GemFire provides statistics for analyzing system performance. Each application or cache server that joins the distributed system can collect and archive this statistical data: You set the configuration attributes that control statistics collection in the `gemfire.properties` configuration file. For more information, see Appendix B, *System Statistics*, on page 235.

You can also collect your own application defined statistics. This is described in *Statistics* on page 464 of the *GemFire Enterprise Developer's Guide*.

To view and analyze archived historical data, you can use Visual Statistics Display (VSD). Contact GemStone Technical Support for instructions about acquiring VSD. For documentation, see the *Visual Statistics Display* manual.

2.7 System Tuning

The performance of a distributed GemFire system depends on choosing the right data distribution protocol, either TCP or IP multicast, and setting up your network to support it optimally. For best results, weigh the benefits of high throughput provided by multicast against the reliable delivery of TCP, in terms of the specific needs of your application.

If you are running over multicast, expect the setup and tuning process to take more effort than average, especially if this is the first multicast application on your network.

> *Improperly configured multicast can affect production systems. If you intend to use multicast on a shared network, work with your network administrator and system administrator from the planning stage of the project.*

For more information, see Chapter 8, *Monitoring and Tuning the Distributed System*.

Chapter 3

Configuring the System

This chapter discusses the files used for GemFire Enterprise® deployment and system configuration, including their placement and specification. The chapter describes in detail the GemFire configuration attributes that you can modify through the `gemfire.properties` file. These properties govern basic system member communication, logging, and statistics gathering as well as more complex issues such as security, socket use, and message queue management.

In this chapter:

For details about using the `gemfire` command-line utility, see Appendix A.

> *This chapter does not include the cache-level configuration done through the cache configuration file,* `cache.xml`*, or the corresponding API for those attributes. For that information, see GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide.*

3.1 GemFire Configuration Files

This section lists the files used for GemFire Enterprise deployment and system configuration. Of these files, GemFire only requires the license. If the other files are not available, the system runs with default values.

- **gemfire.properties**—This file contains the settings required to join a distributed system. Configuration includes system member discovery, communication parameters, security, logging, and statistics. Distributed system membership is described in *Configuring Member Discovery and Communication* on page 59. For a detailed list of the parameters in the `gemfire.properties` file, see *System Properties in the gemfire.properties File* on page 48.

  Each member has its own `gemfire.properties` file, which is usually placed in the working directory where the process runs. Other possible file locations are listed below. The application software may include a set of `gemfire.properties` files that you can edit. If not, you create the files.

- **gemfireLicense.zip**—This is the license file provided to you by GemStone that allows you to use the product for development and production. Do not unzip this file. For more information, see *GemFire Licenses* on page 24.

- **cache.xml**—This is the declarative cache configuration file. This file contains XML declarations for cache, region, and region entry configuration. It is also used to configure things like disk stores, database login credentials, server and gateway location information, and socket configuration. For a detailed list of the parameters in the `cache.xml` file, see *GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide*.

The file names listed are defaults. You and the programmer can specify different names for these files.

Specifying the Configuration File Locations

Each of the three configuration files has a default name, a set of file search locations, and a system property that can be used to override the defaults. To use the default file specification, you must place the file at the top level of its directory or jar file. The system properties are standard file specifications that can have absolute or relative pathnames and filenames. If you do not specify an absolute file path and name, the search looks through all the search locations for the file.

Table 3.1 Configuration File Specifications and Search Locations

Default File Specification	Search Locations Used For Relative File Names	System Property
`gemfire.properties`	1. current directory 2. home directory 3. CLASSPATH	`gemfirePropertyFile`
`cache.xml`	1. current directory 2. CLASSPATH	`gemfire.cache-xml-file`
`gemfireLicense.zip`	1. current directory 2. product directory 3. CLASSPATH	`gemfire.license-file`

These are all acceptable values for the `gemfirePropertyFile` file:

```
/zippy/users/jpearson/gemfiretest/gemfire.properties
c:\gemfiretest\gemfire.prp
myGF.properties
test1/gfprops
```

For this last specification, if you launch your GemFire system member from `/testDir` in a Unix file system, GemFire first looks for `/testDir/test1/gfprops`, then for `yourHomeDir/test1/gfprops`, then under every location in your `CLASSPATH` for `test1/gfprops`.

Changing the Default File Specifications

Applications can pass `java.lang.System` *properties to the distributed system connection to change these file specifications. The Java system properties override command-line settings and* `gemfire.properties` *settings. You can verify an application's property settings in the configuration information logged at application startup. The configuration is listed for* `log-level` *set to* `config` *or lower. For more information on logging, see* Chapter 10, *GemFire System Logging*, on page 185.

If GemFire does not find your license file, it defaults to the evaluation file provided with the product, which limits your system to three members, and any server to three client connections.

You can change file specifications at the command line. This invocation of the application, `testApplication.TestApp1`, provides non-default specifications for the `cache.xml` and `gemfire.properties` files:

Example 3.1 Setting Non-default File Specifications at the Command Line for an Application

```
java -Dgemfire.cache-xml-file=
/gemfireSamples/examples/dist/cacheRunner/queryPortfolios.xml
-DgemfirePropertyFile=defaultConfigs/gemfire.properties
testApplication.TestApp1
```

Here is a `cacheserver` invocation using the same specifications:

Example 3.2 Setting Non-default File Specifications at the Command Line for the cacheserver

```
cacheserver start -J-Dgemfire.cache-xml-file=
/gemfireSamples/examples/dist/cacheRunner/queryPortfolios.xml
-J-DgemfirePropertyFile=defaultConfigs/gemfire.properties
```

You can also change the specifications for the `cache.xml` file and the license file inside the `gemfire.properties` file. You can modify the `defaultConfigs/gemfire.properties` file to specify the other files.

Specifications in `gemfire.properties` *files cannot use environment variables.*

Example 3.3 Setting Non-default File Specifications in a gemfire.properties File

```
#Tue May 09 17:53:54 PDT 2006
mcast-address=224.0.0.250
mcast-port=10333
locators=
license-type=evaluation
cache-xml-file=/gemfireSamples/examples/dist/cacheRunner/queryPortfolios.xml
```

Then these application and cacheserver invocations accomplish the same as the previous command line invocations:

Example 3.4 Setting Non-default File Specifications Through a gemfire.properties File: Application

```
java -DgemfirePropertyFile=defaultConfigs/gemfire.properties
testApplication.TestApp1
```

Example 3.5 Setting Non-default File Specifications Through a gemfire.properties File: cacheserver

```
cacheserver start -J-DgemfirePropertyFile=defaultConfigs/gemfire.properties
```

Jar File Deployment

The GemFire files can be deployed inside your application `jar` file.

To use this option, you need to:

- Jar the files.
- Set the GemFire system properties to point to the files as they reside in the `jar` file.
- Include the `jar` file in your `CLASSPATH`.
- Make sure the `jar` file copies are the only ones visible to the application at runtime. GemFire searches the `CLASSPATH` only after searching the current directory and possibly other directories (see the search order in Table 3.1 on page 42), so you must ensure the files are not present in the other search areas.

Example

This example includes the cache configuration file, `myCache.xml`, in `my.jar`. The contents of `my.jar` list as follows:

```
% jar -tf my.jar
META-INF
META-INF/MANIFEST.MF
myConfig/
myConfig/myCache.xml
```

To use this, set the system property, `gemfire.cache-xml-file`, to this file specification:

```
myConfig/myCache.xml
```

and make sure your `CLASSPATH` includes `my.jar`. Finally, verify that `myCache.xml` is not present in `./myConfig/myCache.xml`, (the current directory location of `myConfig/myCache.xml`). When you start your application, the configuration file is loaded from the `jar` file.

3.2 Configuring GemFire System Properties

Every member that joins a distributed GemFire system needs to be configured to find the other members and communicate with them. This information is defined in the GemFire distributed system property settings. Generally, you will store all your properties in the `gemfire.properties` file, but you may need to provide properties through other means. For example, to pass in security properties for username and password that you have received into your running application from keyboard input.

The product `defaultConfigs` *directory has a sample* `gemfire.properties` *file.*

For each distributed system property, the connection process searches through a number of sources until the value is determined or an exception is reached. The sources are listed here in the order of preference. The API configurations are available only to Java applications.

In addition to the properties settings themselves, you can set a non-default `gemfire.properties` *file name through a* `java.lang.System` *property or* `Properties` *object. For information on where the method searches for this file, see* *Specifying the Configuration File Locations* on page 42.

1. `java.lang.System` property setting. For applications, set these in your code or at the command line.

   Naming: Specify these properties in the format `gemfire.`*`property-name`*, where *`property-name`* matches the name in the `gemfire.properties` file. To set the gemfire property file name, use `gemfirePropertyFile` by itself.

   - In the API, set the `System` properties before the cache creation call.

     Example:

     ```
     System.setProperty("DgemfirePropertyFile", "gfTest");
     System.setProperty("Dgemfire.mcast-port", "10999");

     Cache cache = new CacheFactory().create();
     ```

   - At the `java` command line, pass in `System` properties using the `-D` switch.

     Example:

     ```
     java -DgemfirePropertyFile=gfTest -Dgemfire.mcast-port=10999
     test.Program
     ```

2. Entry in a `Properties` object.

   Naming: Specify these properties using the names in the `gemfire.properties` file. To set the gemfire property file name, use `gemfirePropertyFile`.

   - For the API, create a `Properties` object and pass it to the cache create method.

     Example:

     ```
     Properties properties= new Properties();
     properties.setProperty("log-level", "warning");
     properties.setProperty("name", "testMember2");
     ClientCache userCache =
         new ClientCacheFactory(properties).create();
     ```

   - For the `cacheserver`, pass the properties in at the command line, in `name=value` pairs.

     Example:

     ```
     cacheserver start mcast-port=10338 cache-xml-file=
     /serverConfig/cache.xml
     ```

For more information see *Configuring and Running the GemFire Cache Server* on page 126.

3. Entry in a `gemfire.properties` file. Set these attributes one to a line.

   Example:

   ```
   cache-xml-file=cache.xml
   conserve-sockets=true
   disable-tcp=false
   ```

   The product `defaultConfigs` directory has a sample `gemfire.properties` file.

4. Default value. The default property values are listed in the online Java documentation for `DistributedSystem`.

   You might want to provide your developers with a `gemfire.properties` *file with selected attributes pre-configured. This gives you some control over such things as network use and archive file location and size.*

3.3 Overview of System Properties

This table provides an overview of the properties you can set in the file:

Table 3.2 Overview of Distributed System Connection Properties

Distributed System Connection Properties by Group	
Connection name	The symbolic name to be used to identify the connection.
Distributed system	How system members locate one another. The user indicates the locators' host and port values and/or multicast port and address. For locators, this includes SSL security and network partitioning detection settings. See *Configuring Member Discovery and Communication* on page 59.
Communication	Settings governing the use of TCP/IP and UDP sockets and specifying timeouts for member communication and for cache message distribution. See *Peer-to-Peer Messaging and Distribution* on page 65.
Roles	The optional membership roles that a system member can play. Members of a distributed system can fill one or more roles. A role describes how the member relates to other members, or what purpose it fills. See *Managing Member Relationships* on page 453 of the *GemFire Enterprise Developer's Guide*.
Licensing	The type of license and where the license file may be found. All system members must provide the same licensing information. See *GemFire Licenses* on page 24.
Cache XML file	The optional, declarative XML file used for cache configuration. By default, the file specification is `cache.xml`. This file is discussed in *GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide*.
Logging	Where to save logging output and the granularity of logging to perform. The default is standard out. See Chapter 10, *GemFire System Logging*, on page 185.
Network Partitioning	How members in the distributed system are alerted in the event that they may be disconnected from the distributed system if they do not respond quickly enough, or how an alert is sent to signal that something might be wrong with an unresponsive system member. See *Handling Network Outages* on page 129.
Security	Specifies client and peer authentication and authorization as well as security message logging and the detail level for the logged security messages. See Chapter 5, *Security*, on page 73.
Statistics	Whether and how to collect and archive statistical information. By default, there is no archiving. See Appendix B, *System Statistics*, on page 235.

3.4 System Properties in the gemfire.properties File

This table lists the `gemfire.properties` used to join a distributed system in alphabetical order. Distributed system members include applications, the cacheserver, and other GemFire processes.

Table 3.3 Configuration Properties in gemfire.properties

gemfire property name	Description	Default
`ack-severe-alert-threshold`	Sends a severe warning alert to members in the distributed system, indicating that the member may be disconnected from the distributed system if they do not respond quickly enough. This time period begins after the `ack-wait-threshold` has elapsed.	`0`
`ack-wait-threshold`	The number of seconds a distributed message can wait for acknowledgment before it sends an alert to signal that something might be wrong with the system member that is unresponsive. After sending this alert the waiter continues to wait. The alerts are logged in the system member's log as warnings. Legal values are in the range `0..2147483647`.	`15`
`archive-disk-space-limit`	The maximum size (in megabytes) of all inactive statistic archive files combined. If this limit is exceeded, inactive archive files are deleted, oldest first, until the total size is within the limit. If set to zero, disk space usage is unlimited. For details about statistics archiving, see *Controlling the Size of Archive Files* on page 236.	`0`
`archive-file-size-limit`	The maximum size (in megabytes) of a single statistic archive file. Once this limit is exceeded, a new statistic archive file is created, and the current archive file becomes inactive. If set to zero, file size is unlimited. For details about statistics archiving, see *Controlling the Size of Archive Files* on page 236.	`0`
`async-distribution-timeout`	The number of milliseconds a process that is publishing to this process should attempt to distribute a cache operation before switching over to asynchronous messaging for this process. To enable asynchronous messaging, the value must be set above zero. If a thread that is publishing to the cache exceeds this value when attempting to distribute to this process, it will switch to asynchronous messaging until this process catches up, departs, or some specified limit is reached, such as `async-queue-timeout` or `async-max-queue-size`. Valid values are in the range `0..60000`. *This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.*	`0`

gemfire property name	Description	Default
async-max-queue-size	Limit on the size of asynchronous queues used by processes that are publishing to this process; see async-distribution-timeout on page 48. The maximum size in megabytes the queue can reach before the publisher asks this process to leave the distributed system. Only non-conflated queuing is affected by this value (see enable-async-conflation on page 106 of the *GemFire Enterprise Developer's Guide*). Valid values are in the range 0..1024. *This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.*	8
async-queue-timeout	Limit on asynchronous queues used by processes that are publishing to this process; see async-distribution-timeout on page 48. The maximum milliseconds the publisher should wait with no distribution to this process before it asks this process to leave the distributed system. If a queuing publisher has not been able to send this process any cache operations prior to the timeout, it sends the depart request and this process attempts to close its cache and disconnect from the distributed system. For details see *Slow Receivers with TCP/IP* on page 143. *This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.*	60000
bind-address	Specifies the network adapter card the cache binds to for peer-to-peer communication. Also specifies the default location for GemFire servers to listen on, used unless overridden by the server-bind-address (page 56). This is a machine-wide attribute used for system member and client/server communication. It has no effect on locator location, unless the locator is embedded in a member process. This is only relevant for multi-homed hosts—machines with multiple network cards. Specify the IP address, not the hostname, because each network card may not have a unique hostname. An empty string (the default) causes the member to listen on the default card for the machine. For more information on multi-homed hosts and bind addresses, see *Selecting a Network Adapter Through a Bind Address* on page 69.	""
cache-xml-file	The default file to use to initialize a GemFire cache. For details, see *Cache XML Declaration File Requirements* on page 77 of the *GemFire Enterprise Developer's Guide*.	cache.xml

gemfire property name	Description	Default
conflate-events	Client/server configuration setting. This is a client-side property that is passed to the server. Allowable values are server, true, and false. With the server setting, this client's servers use their own client queue conflation settings. With true or false, the servers disregard their own configuration and either enable (true setting) or disable conflation of events for all regions for the client.	server
conserve-sockets	Specifies whether sockets are shared by the system member's threads. If set to true, threads share, and a minimum number of sockets are used to connect to the distributed system. If false, every application thread has its own sockets for distribution purposes. Where possible, it is better to set conserve-sockets to true and enable the use of specific extra sockets in the application code if needed. For details on the API, see *Controlling Socket Use* on page 471 of the *GemFire Enterprise Developer's Guide*.	true
delta-propagation	Specifies whether to distribute the deltas for entry updates, instead of the full values, between clients and servers and between peers. For details, see *Delta Propagation* on page 327 of the *GemFire Enterprise Developer's Guide*.	true
departure-correlation-window	The number of seconds of process failure history kept by the system for correlating the loss of processes eligible to be the membership coordinator and the lead member. For more information, see *New Membership and Loss of Members* on page 134.	1800
disable-tcp	Disables the use of TCP/IP sockets for inter-cache communications, forcing the cache to use datagram (UDP) sockets for point-to-point messaging. Valid values: true or false. For more information, see *Peer-to-Peer Messaging and Distribution* on page 65.	false
durable-client-id	The ID used by a client to indicate that it is durable. When a durable client connects to a server, this ID is used by the server to identify it. For details, see *Durable Subscription Queues* on page 246 of the *GemFire Enterprise Developer's Guide*.	empty string (not durable)
durable-client-timeout	The number of seconds this disconnected durable client is kept alive and updates are accumulated for it by the server before it is terminated. For details, see *Durable Subscription Queues* on page 246 of the *GemFire Enterprise Developer's Guide*.	300
enable-network-partition-detection	If true, instructs the system to detect and handle splits in the distributed system, typically caused by a partitioning of the network where the distributed system is running. For information, see Handling Network Outages (page 129).	false

gemfire property name	Description	Default
`enable-time-statistics`	Enables time-based statistics for the distributed system and caching. For performance reasons, time-based statistics are disabled by default. For more information, see Appendix B, *System Statistics*, on page 235.	`false`
`license-file`	The name of the license file that contains the license for the distributed system member. If this does not match the file you are using, the system uses the GemFire evaluation license provided in the download.	`gemfireLicense.zip`
`license-type`	The type of license used by this distributed system member: `evaluation`, `development`, or `production`. All members of a distributed system must have the same type of license. For details, see *License Attributes* on page 25.	`evaluation`

`locators`	The list of locators used by system members. The list must be configured consistently for every member of the distributed system. If the list is empty, locators are not used.
	For each locator, provide a host name and/or address (separated by '@', if you use both), followed by a port number in brackets.
	Examples:
	`locators=address1[port1],address2[port2]`
	`locators=hostName1@address1[port1],hostName2@address2[port2]`
	`locators=hostName1[port1],hostName2[port2]`
	On multi-homed hosts, this last notation will use the default address. If you use bind addresses for your locators, explicitly specify the addresses in the `locators` *list—do not use just the hostname.*
	For more on the bind address, see *Selecting a Network Adapter Through a Bind Address* on page 69.
	For details about using locators and multicast ports, see *Configuring Member Discovery and Communication* on page 59.
		`""`

gemfire property name	Description	Default
log-disk-space-limit	The maximum size in megabytes of all inactive log files combined. This value is compared to the total size of all inactive logs in the same directory as the main log and with the same base name as the main log. If this limit is exceeded, inactive log files are deleted, oldest first, until the total size is within the limit. If set to zero, disk space usage is unlimited. For details about logging, see Chapter 10, *GemFire System Logging*, on page 185.	0
log-file	The file to which a running system member writes log messages. For details about logging, see Chapter 10.	null (standard output) for applications, locator.log for the locator and cacheserver.log for the cacheserver
log-file-size-limit	The maximum size in megabytes to which a log file can grow before it is closed and logging rolls on to a new (child) log file. If set to 0 (the default), log rolling is disabled. For details about logging, see Chapter 10.	0
log-level	The level of detail of the messages written to the system member's log. Valid values are fine, config, info, warning, error, severe, and none. Setting log-level to one of the ordered levels causes all messages of that level and greater severity to be printed. Lowering the log-level reduces system resource consumption while still providing some logging information for failure analysis. For details about logging, see Chapter 10.	config
max-num-reconnect-tries	The maximum number or times to attempt to reconnect to the distributed system when membership roles are missing. The roles (page 55) are based on how a member relates to other members, or what purpose a member fills in a distributed system. These optional roles specify the circumstances where an application or cache server continues operation after incidents such as network failures.	3
max-wait-time-reconnect	The maximum number of milliseconds to wait for the distributed system to reconnect in case one of the membership roles (page 55) is lost. The system attempts to reconnect max-num-reconnect-tries, and this timeout period applies to each reconnection attempt.	10000

gemfire property name	Description	Default
`mcast-address`	The multicast address used to discover other members of the distributed system. Only used if `mcast-port` is non-zero. *Select different multicast addresses and ports for different distributed systems. Do not just use different addresses.* This default multicast address was assigned by IANA (http://www.iana.org/assignments/multicast-addresses). Consult the IANA chart when selecting another multicast address to use with GemFire. This attribute must be consistent across the distributed system. For details about using locators and multicast ports, see *Configuring Member Discovery and Communication* on page 59. *This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication. If multicast is enabled, distributed regions use it for most communication. Partitioned regions only use multicast for a few purposes, and mainly use either TCP or UDP unicast.*	`239.192.81.1` for IPv4 (the default IP version) `FF38::1234` for IPv6
`mcast-flow-control`	A tuning property for the flow-of-control protocol for all no-ack UDP messaging, unicast and multicast. These three settings are separated by commas: `byteAllowance`, `rechargeThreshold`, and `rechargeBlockMs`. For more information, see *Tuning UDP Communication* on page 159. Valid values range from these minimums: `10000,0.1,500` to these maximums: *`no_maximum`*`,0.5,60000]`. *This setting controls only peer-to-peer communication, generally between distributed regions.*	`1048576,0.25, 5000`
`mcast-port`	The multicast port used to communicate with other members of the distributed system. If zero, multicast is disabled for both member discovery and distribution. Valid values are in the range `0..65535`. *Select different multicast addresses and ports for different distributed systems. Do not just use different addresses.* This attribute must be consistent across the distributed system. For details about using locators and multicast ports, see *Configuring Member Discovery and Communication* on page 59. *This setting controls only peer-to-peer communication, generally between distributed regions. See the note in* `mcast-address` (page 53).	`10334`

gemfire property name	Description	Default
`mcast-recv-buffer-size`	The size of the socket buffer used for incoming multicast transmissions. You should set this high if there will be high volumes of messages. The default setting of 1048576 is higher than the default OS maximum buffer size on Unix, which should be increased to at least 1 megabyte to provide high-volume messaging on Unix systems. For information on how to increase the receive buffer size for Unix systems, see *Configuring Multicast Speed Limits* on page 163. Valid values are in the range `2048..`*`OS_maximum`*. *This setting controls only peer-to-peer communication, generally between distributed regions. See the note in* `mcast-address` (page 53).	`1048576`
`mcast-send-buffer-size`	The size of the socket buffer used for outgoing multicast transmissions. Valid values are in the range `2048..`*`OS_maximum`*. *This setting controls only peer-to-peer communication, generally between distributed regions. See the note in* `mcast-address` (page 53).	`65535`
`mcast-ttl`	How far multicast messaging goes in your network. System performance might be improved by reducing how far your multicast messaging goes in your network. A setting of 0 constrains multicast messaging to the machine. *This setting controls only peer-to-peer communication, generally between distributed regions. See the note in* `mcast-address` (page 53).	`32`
`member-timeout`	The timeout interval, in milliseconds, used to determine whether another system member is alive. When another member appears to be gone, GemFire tries to contact it twice before quitting. This property sets the timeout interval between each of these attempts. Valid values are in the range `1000..600000`.	`5000`
`membership-port-range`	The range of ports available for unicast UDP messaging and for TCP failure detection. This is specified as two integers separated by a minus sign. Different members can use different ranges. GemFire randomly chooses two unique integers from this range for the member, one for UDP unicast messaging and the other for TCP failure detection messaging. Additionally, the system uniquely identifies the member using the combined host IP address and UDP port number. You may want to restrict the range of ports that GemFire uses so the product can run in an environment where routers only allow traffic on certain ports.	1024-65535
`name`	A symbolic name used to identify the system member.	`""`

gemfire property name	Description	Default
remove-unresponsive-client	When this property is set to true, the primary server drops unresponsive clients from all secondaries and on itself. A client with HARegion queue capacity full is treated as unresponsive client. Set this property to true to avoid blocking puts on the server when maximum capacity client's HARegion queue is reached.	false
roles	A comma-delimited list of strings specifying the membership roles that a member performs in the distributed system. These optional roles specify the circumstances under which an application or cache server continues operation after incidents such as network failures. Any number of members can be configured to perform the same role, and a member can be configured to perform any number of roles. For related configuration options, see max-num-reconnect-tries (page 52) and max-wait-time-reconnect (page 52). For details on member relationships, see *Managing Member Relationships* on page 453 of the *GemFire Enterprise Developer's Guide*.	""
security-*	Any custom properties needed by the AuthInitialize or Authenticator callbacks.	""
security-client-accessor	Static creation method returning an AccessControl object, which determines authorization of client-server cache operations. It specifies the callback that should be invoked in the pre-operation phase, which is when the request for the operation is received from the client.	""
security-client-accessor-pp	Specifies the callback that should be invoked in the post-operation phase, which is when the operation has completed on the server but before the result is sent to the client. The post-operation callback is also invoked for the updates that are sent from server to client through the notification channel.	""
security-client-auth-init	Static creation method returning an AuthInitialize object, which obtains credentials for peers in a distributed system. The obtained credentials should be acceptable to the Authenticator specified through the security-peer-authenticator property on the peers.	""
security-client-authenticator	Static creation method returning an Authenticator object, which is used by a peer to verify the credentials of the connecting peer.	""
security-client-dhalgo	For secure transmission of sensitive credentials like passwords, you can encrypt the credentials using the Diffie-Hellman key exchange algorithm. You do this by setting the security-client-dhalgo system property on the clients to the name of a valid symmetric key cipher supported by the JDK.	""
security-log-file	Sets the name of the log file for security log messages. If this property is not specified, the log file specified in the log-file property is used for security logging.	""

gemfire property name	Description	Default
security-log-level	Specifies the logging level detail for the security log messages. The default log level is config.	config
security-peer-auth-init	Static creation method returning an AuthInitialize object, which obtains credentials for peers in a distributed system. The obtained credentials should be acceptable to the Authenticator specified through the security-peer-authenticator property on the peers.	""
security-peer-authenticator	Static creation method returning an Authenticator object, which is used by a peer to verify the credentials of the connecting peer.	""
security-peer-verifymember-timeout	The timeout value (in milliseconds) used by a peer to verify membership of an unknown authenticated peer requesting a secure connection.	1000
server-bind-address	The network adapter card a GemFire server binds to for client/server communication. You can use this to separate the server's client/server communication from its peer-to-peer communication, spreading the traffic load. This is a machine-wide attribute used for communication with clients in client/server and multi-site installations. It has no effect on locator location. This is only relevant for servers on multi-homed hosts—machines with multiple network cards. Specify the IP address, not the hostname, because each network card may not have a unique hostname. An empty string (the default) causes the servers to listen on the same card that is used for peer-to-peer communication. This is either the bind-address (page 49) or, if that is not set, the machine's default card. For more information on multi-homed hosts and bind addresses, see *Selecting a Network Adapter Through a Bind Address* on page 69.	""
socket-buffer-size	The receive buffer sizes (in bytes) of the TCP/IP connections used for data transmission. To minimize the buffer size allocation required for distributing large, serializable messages, the messages are sent in chunks. This setting determines the size of the chunks. Larger buffers can handle large messages more quickly, but take up more memory.	32768
socket-lease-time	The length of time, in milliseconds, that a thread can have exclusive access to a socket it is not actively using. If a thread loses its lease to a socket it must re-acquire a socket the next time it sends a message. A value of zero causes socket leases to never expire. This property is ignored if conserve-sockets is true. Valid values are in the range 0..600000.	60000

gemfire property name	Description	Default
ssl-ciphers	A space-separated list of the valid SSL ciphers for this connection. You can specify any to use any ciphers that are enabled by default in the configured JSSE provider. For more information on the ssl- *parameters, see *Configuring SSL* on page 91.	any
ssl-enabled	Indicates whether to use SSL for member communications. Valid values are true and false. A true setting requires the use of locators. This attribute must be consistent across the distributed system.	false
ssl-protocols	A space-separated list of the valid SSL protocols for this connection. You can specify any to use any protocol that is enabled by default in the configured JSSE provider.	any
ssl-require-authentication	Indicates whether to require authentication for member communication. Valid values are true and false.	true
start-locator	Automatically starts a locator in the current process when the member connects to the distributed system and stops the locator when the member disconnects from the distributed system. Specify the locator with an optional address or host specification and a required port number, in one of these formats: start-locator=address[port1] start-locator=port1 If you do not specify the address, the address assigned to the member is used for the locator. The address is the member's bind-address (page 49), if set, or the default machine address. *This locator is automatically added to the list of* locators (page 51) *in this set of* gemfire.properties. For more information about locators, see Appendix , *Using Locators For Peer and Client/Server Discovery*, on page 62. To run a locator as a separate process that is not tied to any system member's lifecycle, see Appendix A, *The gemfire Command-line Utility*, on page 227.	"" (does not start a locator)
statistic-archive-file	The file to which a running system member writes statistic samples. An empty string disables statistic archiving. The .gz suffix causes this archive file to be compressed. To archive statistics without compression, omit the .gz suffix. You can view archived statistics with the gemfire stats command. For details about statistics archiving, see *Controlling the Size of Archive Files* on page 236.	null

gemfire property name	Description	Default
`statistic-sample-rate`	The rate, in milliseconds, at which statistics are sampled. Operating system statistics are only updated when a sample is taken. If statistic archiving is enabled then these samples are written to the archive. Valid values are in the range `100..60000`. Lowering the sample rate for statistics reduces system resource use while still providing some statistics for system tuning and failure analysis. For details about statistics archiving, see *Controlling the Size of Archive Files* on page 236.	`1000`
`statistic-sampling-enabled`	Whether to collect and archive statistics on the member. If `false`, archiving is disabled and operating system statistics are no longer updated. Turning statistics sampling off saves on resources, but it also takes away potentially valuable information for ongoing system tuning and about unexpected system problems. For details about statistics, see Appendix B, *System Statistics*, on page 235. *This setting does not apply to partitioned regions, where statistics are always enabled.*	`false`
`tcp-port`	The TCP port to listen on for cache communications. If set to zero, the operating system selects an available port. Each process on a machine must have its own TCP port. Note that some operating systems restrict the range of ports usable by non-privileged users, and using restricted port numbers can cause runtime errors in GemFire startup. Valid values are in the range `0..65535`.	`0`
`udp-fragment-size`	The maximum fragment size, in bytes, for transmission over UDP unicast or multicast sockets. Smaller messages are combined, if possible, for transmission up to the fragment size setting. For more information, see *Tuning UDP Communication* on page 159. Valid values are in the range `1000..60000`.	`60000`
`udp-recv-buffer-size`	The size of the socket buffer used for incoming UDP point-to-point transmissions. If `disable-tcp` is set to `false`, then a reduced buffer size of 65535 is used by default. The default setting of 1048576 is higher than the default OS maximum buffer size on Unix, which should be increased to at least 1 megabyte to provide high-volume messaging on Unix systems. For information on how to increase the receive buffer size for Unix systems, see *Configuring Multicast Speed Limits* on page 163. Valid values are in the range `2048..`*`OS_maximum`*.	`1048576`
`udp-send-buffer-size`	The size of the socket buffer used for outgoing UDP point-to-point transmissions. Valid values are in the range 2048..OS_maximum.	`65535`

Chapter 4

Configuring Member Discovery and Communication

This chapter explains how to configure your GemFire Enterprise® systems so the applications can find each other and distribute messages and data between themselves. It provides information on member discovery and messaging between peers in a single distributed system, client discovery of servers and messaging in client/server installations, and communication between distributed systems in multi-site installations. It also discusses how to use bind addresses in your configurations.

GemFire Enterprise offers several combinations of TCP/IP stream sockets, UDP/IP unicast, and UDP/IP multicast for discovery and communication. The combination that is best for your installation depends in large part on your system topology, your network and computing resources, and the nature of your data traffic and application behavior.

In this chapter:

4.1 Member Discovery

Member discovery is how GemFire applications and cache servers find each other without using hard-coded addresses. Peers in a distributed system always use discovery to establish communication. Clients use either server discovery or hard-coded server lists to find their servers. Multi-site installations use fixed remote site addresses, so there is no discovery process.

Peer Discovery

Peer member discovery can be done in one of two ways:

- **Locators using TCP/IP**—With this method, you run GemFire locator processes that manage a dynamic list of running distributed system members. Locators used in this way are *peer locators*. A new member connects to one of the locators to retrieve the member list, which it uses to join the system.

  Locators are the recommended discovery method for production systems.

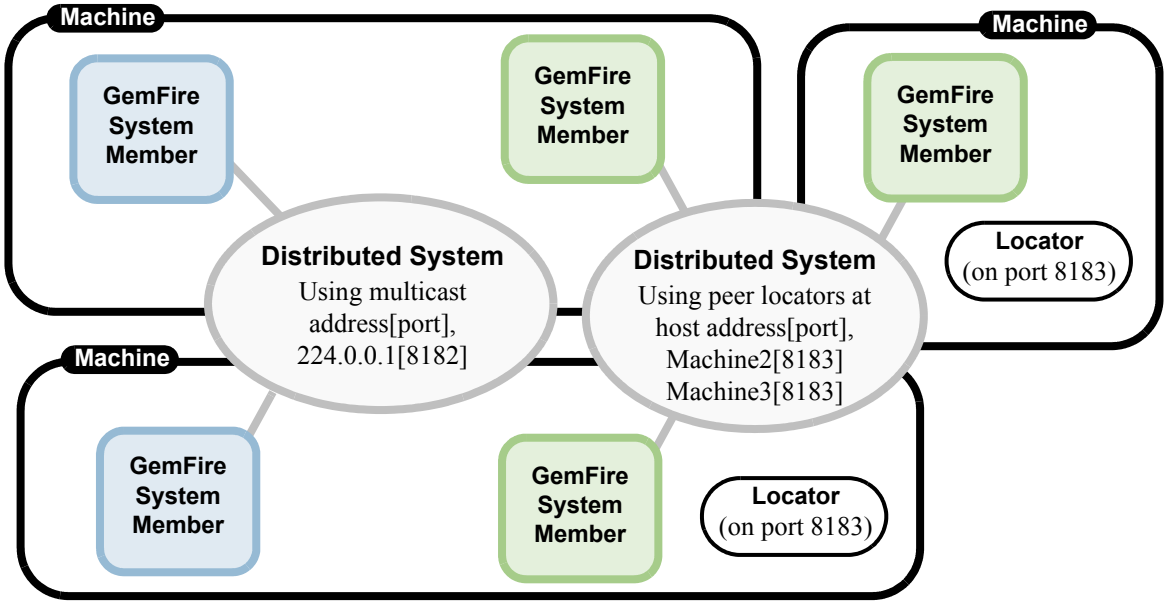
- **UDP/IP multicast**—With this method, new members multicast their presence to an address and port that all members are subscribed to. The existing members respond to establish communication with the new member.

  If multicast is available at your site, it is a convenient way to try out new versions of GemFire Enterprise.

Once they have found each other, members communicate directly, independent of the discovery mechanism. For more information, see *Peer-to-Peer Messaging and Distribution* on page 65.

This figure shows a high-level view of two distributed systems. Here, the system on the left is using multicasting for system member discovery. The system on the right is using locators.

Figure 4.1 Peer-to-Peer System Member Discovery



Member discovery is what defines a distributed system. All applications and cache servers that use the same settings for peer discovery are members of the same distributed system. Each system member has a unique identity and knows the identities of the other members. A member can belong to only one distributed system at a time.

> *Member discovery settings must be consistent throughout the distributed system. Locators are given preference over multicasting for member discovery. If you have both peer locators and multicast configured, the locators are used.*

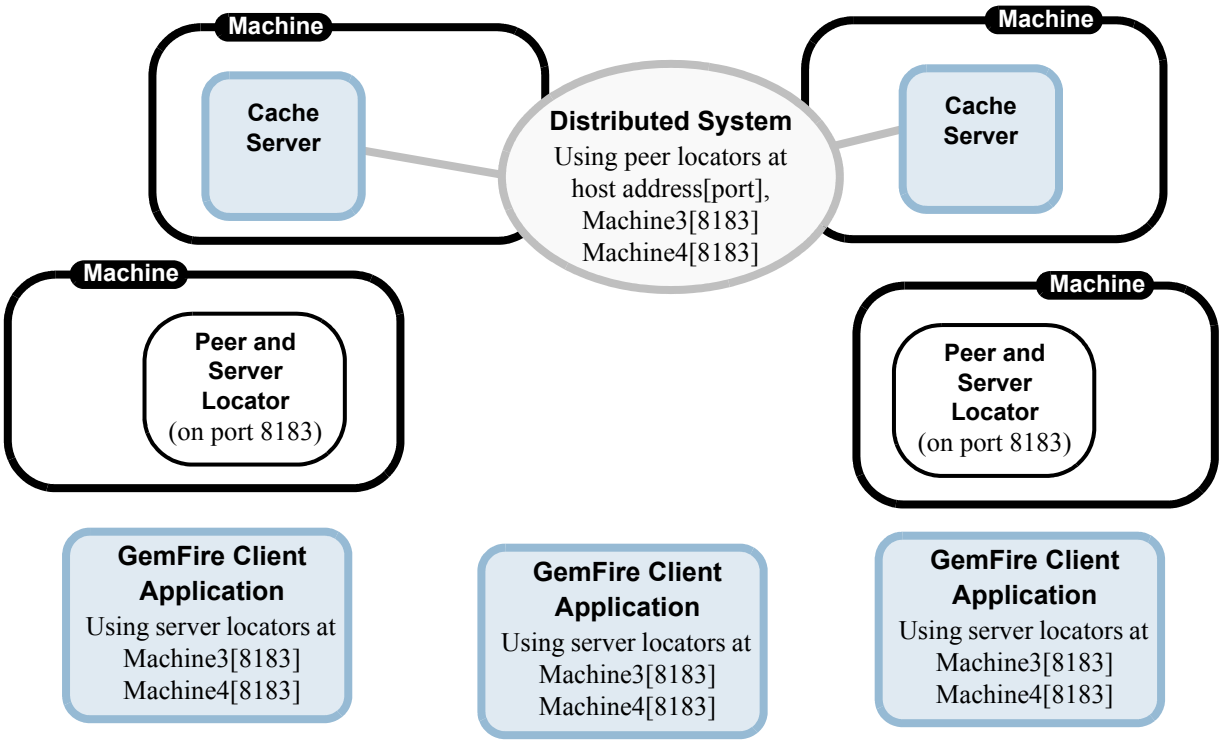
Client/Server Discovery

Locators provide clients with dynamic server discovery and server load balancing. Clients are configured only with locator information, and turn to the locators for directions to the servers to use. The servers can come and go and their capacity to service new client connections will vary. The locators continuously monitor server availability and server load information, providing clients with connection information for the server with the least load at any time.

You do not need to run any special processes to use locators for server discovery. The locators that provide peer discovery in the server system also provide server discovery for clients to the server system.

This figure shows the high-level configuration for a server system using locators for peer and server discovery. The servers use the peer locator functionality to find each other. The clients use these same locators to find the servers. This is the standard configuration.

Figure 4.2 Configuring for Discovery in a Client/Server Installation



For performance and cache coherency, clients and servers must not be run in the same distributed system.

Clients can also use a static server list to identify their servers. In this case, there is no discovery and you cannot add or remove servers while the client is running. This is recommended only for testing in small installations.

Using Locators For Peer and Client/Server Discovery

Locators are configured as peer locators, server locators, or both.

- Peer locators provide distributed system member information for joining members. GemFire peer locators are the recommended mechanism for peer discovery in production systems. Peer locators are required in systems using the GemFire Enterprise security framework or otherwise using SSL for member communication (see Chapter 5, *Security)*.

  A peer locator listens at its host-address and port location for new members and maintains the list of all active members. When a new member joins, it connects to a locator, retrieves the list of members and uses it to establish communication with the rest of the distributed system.

- Server locators give clients dynamic server information and provide server load rebalancing after servers depart or join the system. If you use peer locators in the servers' distributed system, configure those same locators as server locators and use them for client discovery of servers. For information on client/server installations see *Client/Server Architecture and Configuration Basics* on page 201 of the *GemFire Enterprise Developer's Guide*.

  A server locator maintains a dynamic list of all available servers in the system and listens at its host-address and port location for clients.

To ensure the most stable startup and availability, use multiple locators.

Locators are distributed system members of the peer systems they serve. If they are server locators and not peer locators (for when multicast is used for peer member discovery) they must still be run as members of the server distributed system. You can run your locators as standalone applications or you can colocate them in your caching members, so they use the same distributed system connection as your caching threads.

You must give your locators the same distributed system configuration as other distributed system members. In particular, all discovery and communication specifications must be consistent across the distributed system for locator and non-locator members.

To configure your system for locators, follow these steps:

1. Decide how many locators you will use and where they will run. You will use the list of locator host-address and port locations to configure your system members, clients, and the locators themselves, because the locators also need to be able to find one another. You generally want to position your locators on multiple machines so that if one machine goes down or a part of your network fails, some locators remain available.
2. Decide how you will run your locators. Will they run independent of your other GemFire processes or colocated inside your applications? While it adds a little more to your startup and shutdown procedures, running your locators standalone provides the highest reliability and availability of the locator service as a whole. Standalone locators are required for system where split-brain management is enabled.
3. Configure your system members and clients to find the running locators at their addresses and ports. For peer locators, this is usually done through the `locators` configuration line in the members' `gemfire.properties` file. For server locators, this is done in the client's cache configuration file `pool` element.

   If you use bind addresses on any machines where you run locators, see the locators information in Selecting a Network Adapter Through a Bind Address on page 69.

The first process that starts in your distributed system must either be a peer locator or an application with a colocated peer locator. It might be simplest to start all processes with locators first. All configured server locators must be running before any clients are started. Set up your configuration files and startup scripts accordingly.

Locator Property Settings

The colocated locator uses its host application's configuration and log file. The stand-alone locator, started at the command line, always outputs its logging to `locator.log`, but you can specify the directory to use and you can provide a `gemfire.properties` file for the other member configuration options. For more information, see Appendix A, *The gemfire Command-line Utility*, on page 227.

The easiest way to configure your locators is to create a `gemfire.properties` file in the working directory of each locator.

1. Specify all locator addresses and ports. The default port is 10334. Locators need to know about one another, so each locator's `gemfire.properties` file must include the list of all locators in the system. This is the same list you use to configure the other distributed system members:

   ```
   locators=locator1-host-address[port1],locator2-host-address[port2]
   ```

   If you are using bind addresses, see *Selecting a Network Adapter Through a Bind Address* on page 69.

2. Peer locators are distributed system members and must have the same communications configuration as the other members of the distributed system. Especially for any stand-alone locators, make sure to provide the proper communication settings in the `gemfire.properties`. For information on this, see *Peer-to-Peer Messaging and Distribution* on page 65.

3. Specify the security settings. For GemFire Enterprise security options, see Chapter 5, *Security*.

Starting and Stopping GemFire Locators

Stand-alone locators are started individually through the `gemfire` command-line utility and use their own `gemfire.properties` files for configuration. They are started on the command-line like this:

```
gemfire start-locator
```

By default, locators are run as peer locators and as server locators. You can use this startup command to run a locator as a peer but not a server locator:

```
gemfire start-locator -server=false
```

This command runs a locator as a server locator only:

```
gemfire start-locator -peer=false
```

For details, see Appendix A, *The gemfire Command-line Utility*, on page 227.

Embedded locators start and stop automatically with their host applications. For information on this, see `start-locator` on page 57.

Application developers can also define and start locators through the API. See *Developing System Administration Tools* on page 479 of the *GemFire Enterprise Developer's Guide* and the online Java API documentation.

Configuring Your Processes to Find the Running Locators

Peer-to-Peer

Each process that joins a distributed system must supply the list of locator `host-address[port]` pairs to the distributed system connection. You configure GemFire processes to use peer locators either in the `gemfire.properties` file or through the API:

1. In the `gemfire.properties` file, set the `locators` (page 51) attribute to the list of `host-address[port]` pairs of your locators. This is the same list you use to configure the locators themselves:

   ```
   locators=locator1-host-address[port1],locator2-host-address[port2]
   ```

- Application developers can manage locator lists through the API. For cache servers, see *Developing System Administration Tools* on page 479 of the *GemFire Enterprise Developer's Guide*. For applications, see *GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide*.

Client/Server

In a client/server installation, the list of server locators is specified in a `Pool` instance in the clients. Each client supplies the list of locator `host-address[port]` pairs to its connection `Pool` instance. You configure GemFire processes to use server locators in the `cache.xml` file or through the API: You do not need to provide the complete list of locators to the clients at startup, but you should provide as complete a list as possible. The locators maintain a dynamic list of locators and servers and provides the information to the clients as needed. See the online Java documentation and the *Client/Server Architecture and Configuration Basics* on page 201 of the *GemFire Enterprise Developer's Guide*.

Using Multicast for Peer Discovery

UDP/IP multicasting is the default for peer-to-peer membership discovery, and it is the easiest to use out of the box when you are trying out a new version of GemFire Enterprise.

You can use multicasting for peer-to-peer discovery even if you use locators for client discovery of servers. To do this, set the multicast properties as indicated below, and start your locators with the `-peer` locator property set to `false`.

> *Locators are recommended for discovery in production systems.*

To use multicasting for discovery, leave the `locators` attribute for the distributed system blank, or remove the line from the `gemfire.properties` file:

```
locators=
```

To configure a non-default multicast address or port, set the `mcast-port` (page 53) and the `mcast-address` (page 53) in the `gemfire.properties` file of each application and cache server:

```
mcast-address=IP_address
mcast-port=port
```

> *Use both unique ports and unique addresses for your systems. Do not use the same port number for different systems. Some operating systems may not keep communication separate between systems that use unique addresses but the same port number.*

To use multicast for the distribution of region operations, you need to also enable it at the region level. For details, see *Peer-to-Peer Messaging and Distribution* on page 65.

You can use the default port and address if you wish. Setting the port to 0 (zero) disables multicast for the member.

4.2 Peer-to-Peer Messaging and Distribution

This section explains your options for messaging and distribution between members of a distributed system, and shows how to configure them. All applications and cache servers in a distributed system must have the same protocols configured for peer-to-peer communication. The protocols are configured at the VM level in the `gemfire.properties` file.

Choosing the Protocols to Use

For general messaging and region operations distribution, GemFire uses either TCP or UDP unicast. The default is TCP. You can use the chosen unicast protocol for all communications or, if you want to, you can target specific regions to use UDP multicast for operations distribution. For information on how to set your protocol choices, see *Configuring Your Protocols* on page 66.

This section discusses the differences between the protocol choices. The best combination for your installation depends in large part on how you use your data and event messaging.

TCP

TCP (Transmission Control Protocol) provides reliable in-order delivery of the system messages. TCP is more appropriate than UDP if the data is partitioned, if the distributed system is small, or if network loads are unpredictable.

TCP is preferable to UDP unicast in smaller distributed systems because it implements more reliable communications at the operating system level than UDP.

In smaller systems, TCP performance can be substantially faster than UDP. As the size of the distributed system increases, however, the relatively small overhead of UDP makes it the better choice. TCP adds new threads and sockets to every member, causing more overhead as the system grows.

UDP

UDP (User Datagram Protocol) is a connectionless protocol which uses far fewer resources than TCP. Adding another process to the distributed system incurs little overhead for UDP messaging, while TCP adds new threads and sockets to every process in the system.

UDP on its own is not reliable however, and messages are restricted in size to 64k bytes or less, including overhead for message headers. Large messages must be fragmented and transmitted as multiple datagram messages. Consequently, UDP is slower than TCP in many cases and unusable in other cases if network traffic is unpredictable or heavily congested.

UDP is used in GemFire for both unicast and multicast messaging. GemFire implements retransmission protocols to ensure proper delivery of messages over UDP.

UDP Unicast

UDP unicast is the alternative to TCP for general messaging. UDP is more appropriate than TCP for unicast messaging when there are a large number of processes in the distributed system, the network is not congested, cached objects are small, and applications can give the cache enough processing time to read from the network. If you disable TCP, GemFire uses UDP for unicast messaging.

UDP Multicast

Your options for general messaging and for default region operations messaging is between TCP and UDP unicast. You can choose to replace the default with UDP multicast for operations distribution of some or all of your regions. For every region where you want to use multicast, you set an additional attribute on the region itself.

Multicast is most appropriate when the majority of processes in a distributed system are using the same cache regions and need to get updates for them, such as when the processes define replicated regions or have their regions configured to receive all events.

When multicast is enabled for a region, all processes in the distributed system receive all events for the region. Every member receives each message for the region and has to unpack it, schedule it for processing, and then process it, all before discovering whether it is interested in the message. Multicasting is suitable, therefore, for regions that are of general interest in the distributed system, where most or all members have the region defined and are interested in receiving most or all messages for the region. Multicasting should not be used for regions that are of little general interest in the distributed system.

If you provide multicast configuration settings in the `gemfire.properties` at connection time, you can then enable multicast communication for cache operations on any or all of your data regions. GemFire still sends unicast messages when appropriate.

If data is partitioned, multicast is not a useful option. Even with multicast enabled, partitioned regions still use unicast for almost all purposes.

Configuring Your Protocols

This section shows how to configure the various protocols using `gemfire.properties` and `cache.xml`.

TCP

Messaging and data distribution through TCP is the default. You can specify it in the `gemfire.properties` file with this entry:

```
disable-tcp=false
```

For details, see `disable-tcp` (page 50).

UDP Unicast

Disable TCP to configure the system to use UDP unicast for general messaging. Add this entry to the `gemfire.properties` file:

```
disable-tcp=true
```

For details, see `disable-tcp` (page 50).

> *Disabling TCP here does not prevent you from using TCP locators. You can configure UDP for communication and locators for member discovery.*

For each member, GemFire selects a unique port for UDP unicast communication. You can further restrict the range by setting `membership-port-range` (page 54). In the `gemfire.properties` file, specify the membership port range, like this:

```
membership-port-range=1024-60000
```

UDP Multicast

Improperly configured multicast can affect production systems. If you intend to use multicast on a shared network, work with your network administrator and system administrator from the planning stage of the project.

To configure IP multicast for messaging, enable multicast for the member in `gemfire.properties` and then configure multicast for messaging on a per-region basis in `cache.xml`.

1. In the `gemfire.properties` file, add multicast address and port selections, like these:

   ```
   mcast-address=239.192.81.2
   mcast-port=10596
   ```

   These attributes must be consistent across the distributed system. For more information, see `mcast-address` on page 53 and `mcast-port` on page 53.

2. In the `cache.xml` file, enable multicast for each region that needs multicast messaging:

   ```
   <region-attributes multicast-enabled="true"/>
   ```

   A region requests multicasting for all distributed operations on the region if its `multicast-enabled` attribute is set to `true`. For more information, see `multicast-enabled` on page 113 of the *GemFire Enterprise Developer's Guide*.

Also see *Tuning Socket Communication* on page 151 for details on the communication tuning parameters.

In addition, you may need to address interrelated setup and tuning issues at the GemFire, operating system, and network level. For details on tuning and troubleshooting IP multicast, see *Tuning Multicast Communication* on page 161.

4.3 Standalone Members

You can run a GemFire member as an isolated application that uses the caching APIs but has no working distributed system connection. Running standalone has a faster startup and is appropriate for any member that is isolated from other applications. The primary use case is for client applications.

*Standalone is the recommended configuration for clients in a client/server installation. GemFire automatically configures the distributed system for a `ClientCache` as a standalone. See *Standard Client/Server Deployment* on page 203 of the *GemFire Enterprise Developer's Guide*.*

To run an isolated member, configure the distributed system with:

- `mcast-port` set to `0`
- `locators` set to the empty string

This disables all system member connection information and causes the process to start in standalone mode. Any Java application that configures its distributed system connection in this way obtains access to the GemFire caching APIs but runs with no visibility to any other GemFire Enterprise process.

Standalone members cannot be accessed or monitored from the GemFire JMX agent.

4.4 Client/Server Communication

Servers and clients communicate directly through TCP/IP sockets. For general information on client/server configuration, see *Standard Client/Server Deployment* on page 203 of the *GemFire Enterprise Developer's Guide*.

In addition to servicing clients, each server is a member of its distributed system, connecting to and communicating with its peers through the mechanisms discussed in *Configuring Member Discovery and Communication* on page 59 and *Peer-to-Peer Messaging and Distribution* on page 65.

Clients generally run in some standalone fashion, either by being the sole members of a distributed system or by running as *Standalone Members*, with no active distributed system instantiated. They can belong to a system with other peers, however, in which case they also use the mechanisms discussed in the peer-to-peer sections of this chapter.

4.5 Multi-Site Communication

Gateway hubs and gateways communicate through TCP/IP sockets. The gateway hub listens at a specified address and port for gateway communication from remote sites. Gateways are configured with endpoint information matching the remote gateway hub specifications. The gateway sends connection requests to the gateways hubs to establish the two-way TCP connections. For information on the multi-site configuration, see *Configuring Multi-site Installations* on page 267 of the *GemFire Enterprise Developer's Guide*.

In addition to the site-to-site communication, each gateway hub is a member in its own distributed system, connecting to and communicating with its peers through the mechanisms discussed in *Configuring Member Discovery and Communication* on page 59 and *Peer-to-Peer Messaging and Distribution* on page 65.

4.6 Selecting a Network Adapter Through a Bind Address

This section applies only to systems running on machines that have more than one network interface card, or network adapter. On machines with multiple network cards, one card is used as the default. If you do not want to use the default card for GemFire processes, this section tells you how to specify another card.

A host machine with more than one network adapter is referred to as a multi-homed host. Single-homed hosts (machines with one network adapter) are uniquely identifiable by their adapter's IP address. All communication to and from the outside goes through the one adapter. Multi-homed hosts have multiple adapter cards, so they have multiple IP addresses. On every multi-homed host, one of the addresses is set as the default for network communication. If you wish to have your GemFire processes use the default adapter, no configuration changes are needed.

For a multi-homed host, you can use GemFire configuration attributes to specify the adapters to be used.

> *Specify the IP address, not the hostname, because each network card may not have a unique hostname.*

Locators

You can configure a locator to use a bind address by supplying the address when you start the locator.

- On the command line:

  ```
  gemfire start-locator -address=bind-address -port=portNumber
  ```

- Inside a GemFire application:
  - You can automatically start a colocated locator using the gemfire property `start-locator` (page 57), specifying the bind address for it in that property setting.
  - You can start the locator using the `Locator` class in `com.gemstone.gemfire.distributed`. Use a method that accepts a `bindAddress` argument.

If your locator uses a bind address, make sure every process that accesses the locator has the address as well:

- For peer-to-peer access to the locator, use the same address and port in your `gemfire.properties` `locators` (page 51) list.
- If you use locators for server discovery in a client/server installation, use the same addresses in the locator list in the client's server pool configuration. See *Client/Server and Multi-site* on page 70.

Peer-to-Peer

For GemFire members running on multi-homed hosts, you can specify a non-default network adapter for TCP and UDP unicast communication. All multicast communication goes through the address specified in the `mcast-address` property in the `gemfire.properties` file.

A network adapter used for non-multicast peer-to-peer communication follows this order of preference:

- `bind-address` (page 49)
- machine default

The `bind-address` is not set by default. This example `gemfire.properties` line sets it:

```
bind-address=10.80.10.80
```

> *The* `bind-address` *setting must be the same for all GemFire processes running on the same machine and in the same distributed system.*

Client/Server and Multi-site

GemFire uses TCP/IP for client/server and site-to-site connections. In both situations, server processes listen for communication from client processes (with multi-site, the gateway hub is the server process and the gateway is the client process). The clients know where to connect to the servers either from static server lists or from the locators specified in their locator list. If you use bind addresses for your servers or locators, you need to make sure to use the same addresses in your clients' pool configurations. The servers and locators do not need to know about any bind addresses used for the clients. When clients connect, they send their return addresses with the connection request.

Server Configuration

In server and gateway hub VMs on multi-homed hosts, the network adapter used for client/server and gateway communication follows this order of preference:

- `cacheserver` command-line specification of `server-bind-address` (see page 127)
- `CacheServer` `bind-address` on page 207 of the *GemFire Enterprise Developer's Guide*
- `gemfire.properties` `server-bind-address` (page 56)
- `gemfire.properties` `bind-address` (page 49)
- `CacheServer` default bind address setting, currently set to the machine's default address

By default, none of the bind address settings are set. If you use no settings, the machine's default address is used.

> *The* `server-bind-address` *setting must be the same for all GemFire processes running on the same machine and in the same distributed system.*

If you want all the servers and gateway hubs to communicate over the same non-default adapter as you use for your non-multicast peer-to-peer communication, specify only the `bind-address`. Client/server and gateway communication uses the `bind-address` property value if no bind address is set for the server.

If you want to distribute the load of network traffic for your distributed system, send your client/server and gateway traffic through a different adapter than the peer-to-peer traffic by setting a server bind address. These `gemfire.property` lines specify different non-default addresses for the member:

```
bind-address=10.80.10.80
server-bind-address=10.80.10.81
```

Client Configuration

If you use bind addresses for your servers or for your server locators, you must use the same addresses to configure the server pool used by your clients. If you use bind addresses for your multi-site gateway hubs, your gateways must refer to those hubs using the same addresses. Do not use host names as these resolve to the default machine addresses.

- For multi-site, put the bind-address in the `host` specification. For information see *Gateway Endpoint Attributes* on page 273 of the *GemFire Enterprise Developer's Guide*.
- For client/server, put the bind-address in the locator or server specification, according to where you use it. For information, see *Client Cache Configuration* on page 209 of the *GemFire Enterprise Developer's Guide*.

4.7 Choosing Between IPv4 and IPv6

You can use Internet Protocol version 4 (IPv4) or 6 (IPv6) for your GemFire address specifications.

IPv4 was the first protocol and is still the main one in use, but its address space is expected to be exhausted within a few years. IPv6 succeeds IPv4, and will provide a much greater number of addresses. IPv6 uses a 128-bit address, while IPv4 uses a 32-bit address.

Based on current testing with GemFire, IPv4 is generally recommended. IPv6 connections tend to take longer to form and the communication tends to be slower.

Not all machines support IPv6 addressing. To use IPv6, all machines in your distributed system must support it or you will have connectivity problems.

Do not mix IPv4 and IPv6 addresses. Use one or the other, across the board.

To use IPv6, set the Java property, `java.net.preferIPv6Addresses`, to `true`. IPv4 is the default version.

The addresses are specified in GemFire like this:

IPv4

```
239.192.81.2
```

IPv6

```
2001:db8:85a3:0:0:8a2e:370:7334
```

Chapter 5

Security

The security framework establishes trust between members, and also authorizes cache operations from clients based on that trust. You establish trust by verifying credentials when one process connects to another, for example:

- New members connect to the locator in a peer-to-peer topology.
- Clients connect to cache servers.
- One system connects to another in a multi-site system, using mutual authentication.
- Diffie-Hellman key exchange to encrypt sensitive credentials.

In this chapter:

5.1 Security Features

GemFire Enterprise® provides member authentication and cache access authorization with these features:

- **Flexible plug-in framework**. Plug-in mechanism for authentication of clients and servers and authorization of cache operations from clients. Any security infrastructure can be plugged into the system as long as the plug-ins implement the required GemFire interfaces.
- **Cache server authentication**. Allows peer cache servers into the distributed system if their credentials are authenticated by the locator to which they connect.
- **Client authentication**. Implemented through authentication of client's credentials by a cache server when the client attempts to connect to the server. Multiple users can connect, with separate authorization levels, from within one client application.
- **SSL-based authentication**. Allows configuration of all connections to be SSL- based, rather than plain socket connections.
- **Authorization of cache operations**. Selectively authorized cache operations by clients based on the predefined, associated roles, where the credentials are provided by the client when connecting to the server.
- **Data modification based on authorization**. Allows authorization callbacks to modify or filter data sent from the client to the server. Similarly, after the cache operations complete on the server, a post authorization callback occurs, which can filter or modify results sent to the client. However, the results cannot be modified while using function execution.
- **Sample implementations**. Authentication and authorization sample implementations.

5.2 Implementing Security

GemFire Enterprise can authenticate peer system members, clients, and remote gateways. GemFire can also authorize cache operations on a server from clients. A distributed system using authentication bars malicious peers or clients, and deters inadvertent access to its cache. You can restrict or completely block client operations on a cache server based on the roles and permissions assigned to the credentials submitted by the client.

Use consistent security settings between similar processes in a single distributed system. For example, configure all servers in a system with the same client authentication settings.

You can use GemFire security for secure communication, to authorize system membership, and to authorize specific activities in the cache:

- Use locators for peer discovery within the distributed systems and for client discovery of servers. See *Using Locators For Peer and Client/Server Discovery* on page 62.
- Implement membership authentication. Depending on your installation and security requirements, you may use a combination of peer-to-peer, client/server, and multisite settings.
- If you have a client/server system, implement any authorized access control your servers will use for clients attempting to access or modify the cache.
- If you want to use secure socket layer (SSL) protocol for your peer-to-peer and client/server connections, configure your clients and servers for that.

For all security-related system properties, see the properties starting with `security-` in the gemfire.properties file listings, in *System Properties in the gemfire.properties File* on page 48.

Also see the Javadocs for `com.gemstone.gemfire.security`.

5.3 Implementing Membership Authentication

Authentication is done by initializing credentials in the joining member, sending the credentials to an authenticator member in the system, and receiving authentication to join. Depending on the member, the new member may in turn become an authenticator to other joining members. Members joining a system must trust that existing members are already authenticated.

GemFire provides a flexible framework for your security authentication plug-ins. You choose the method of authentication, such as LDAP or PKCS, and program the plug-ins accordingly.

1. Determine the method of authentication that you will use. It is assumed that you know how to use it.
2. Determine any special properties required for your authentication's credentials initialization. Decide how you will get the properties to the initialization method. Depending on how sensitive the properties are and on your application requirements, you may do a combination of:
   - Passing the additional properties through the `gemfire.properties` file settings or programmatically, with a call to the `ClientCache` creation. All properties starting with `security-` are automatically passed to the `AuthInitialize` implementation.
   - Obtaining the properties dynamically in the `AuthInitialize.getCredentials` method
3. For joining members, program and configure the credentials initialization plug-in:

   3.1 For all joining members, create an implementation of the `com.gemstone.gemfire.security.AuthInitialize` interface:

   - Program a public static method to return an instance of the class.
   - Program the `getCredentials` method to create all properties required by the `Authorize` method via the member's credentials.

     See the Javadocs for `com.gemstone.gemfire.security.AuthInitialize`.

   3.2 For peers and locators, set the `gemfire.properties security-peer-auth-init` to the fully qualified name of the static method you programmed that returns an instance of the `AuthInitialize` class. In these examples, the method is named `create`:

   ```
   // Peer init example where myAuthInitImpl.create returns the
   // instance of AuthInitialize
   security-peer-auth-init=myAuthPkg.myAuthInitImpl.create
   ```

   3.3 For clients and gateways, set the `gemfire.properties security-client-auth-init` to the fully qualified name of the method you programmed that returns an instance of the `AuthInitialize` class:

   ```
   // Client/WAN init example where myAuthInitImpl.create returns the
   // instance of AuthInitialize
   security-client-auth-init=myAuthPkg.myAuthInitImpl.create
   ```

   3.4 For all members, set any additional `gemfire.properties security-*` properties required by your `AuthInitialize` implementation.
4. For authorizing members, program and configure the credentials authorization plug-in:

   4.1 Implement the `com.gemstone.gemfire.security.Authenticator` interface:

   - Program a public static, zero-argument method to return an instance of the class.
   - Program the `authenticate` method to authenticate the credentials and return a `java.security.Principal` object.

     See the Javadocs for `com.gemstone.gemfire.security.Authenticator`.

4.2 For peers and locators, set the `gemfire.properties security-peer-authenticator` to the fully qualified name of the method that returns an instance of the `Authenticator` class:

```
// Peer auth example where myAuthenticatorImpl.create returns the
// instance of Authenticator
security-peer-authenticator=myAuthPkg.myAuthenticatorImpl.create
```

4.3 For servers and gateways, set the `gemfire.properties security-client-authenticator` to the fully qualified name of the method that returns an instance of the `Authenticator` class:

```
// Client/WAN auth example where myAuthenticatorImpl.create
// returns the instance of Authenticator
security-client-authenticator=myAuthPkg.myAuthenticatorImpl.create
```

4.4 For all members, set any additional `gemfire.properties security-*` properties required by your `Authenticator` implementation

5. For all members, provide the list of authenticated locators in the `gemfire.properties`.

When a Member Fails to Join

- Peer credentials are initialized and verified automatically when a member joins a distributed system.
  - If a joining member has invalid credentials, the connection request throws an `AuthenticationFailedException`.
  - If a joining member does not provide credentials, the request throws an `AuthenticationRequiredException`.
- Client credentials are initialized and verified automatically during the initial connection process.
  - If client authentication fails due to invalid credentials, the server sends an `AUTHENTICATION_FAILED` message back to the client. The connection fails, and an `AuthenticationFailedException` is thrown for the current operation.
  - If the client authentication fails due to missing credentials, the server sends a `NO_AUTHENTICATION` message back to the client. The client connection fails, and an `AuthenticationRequiredException` is thrown for the current operation.

JMX

If you use a JMX agent to administer and manage an authentication-enabled GemFire Enterprise distributed system, the agent must provide security credentials. Security properties cannot be passed to a JMX Agent on the command line, but they can be supplied at startup by adding the security-specific system properties to the agent's properties file, `agent.properties`. With the properties specified, the call to `Agent.connectToSystem` causes the agent to be authenticated with the distributed system. GemFire Enterprise security does not manage RMI clients to the JMX Agent. Once connected, the JMX Agent is considered authenticated and any RMI client has access to the connected distributed system. For RMI client authentication, use MX4J security.

Encrypting Credentials with Diffie-Hellman

For secure transmission of sensitive information, like passwords, you can encrypt credentials using the Diffie-Hellman key exchange algorithm. This encryption applies only to client/server authentication.You need to specify the name of a valid symmetric key cipher supported by the JDK. Valid key names, like DES, DESede, AES, and Blowfish, enable the Diffie-Hellman algorithm with the specified cipher to encrypt the credentials. For valid JDK names, see http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html#AppA.

> *Using Diffie-Hellman slows the creation of client/server connections, but it has no impact on performance after the connection has been established.*

In the instructions that follow, it is assumed you understand how to use your security algorithm.

Enabling Server Authentication of Client with Diffie-Hellman

Set this in property in the client's `gemfire.properties`:

- `security-client-dhalgo`. Name of a valid symmetric key cipher supported by the JDK, possibly followed by a key size specification.

This causes the server to authenticate the client using the Diffie-Hellman algorithm.

Enabling Client Authentication of Server

With Diffie-Hellman enabled, your client can authenticate its servers:

1. In server `gemfire.properties`, set:
   - `security-server-kspath`. Path of the PKCS#12 keystore containing the private key for the server.
   - `security-server-ksalias`. Alias name for the private key in the keystore.
   - `security-server-kspasswd`. Keystore and private key password, which should match.
2. In client `gemfire.properties`, set:
   - `security-client-kspasswd`. Password for the public key file store on the client.
   - `security-client-kspath`. Path to the client public key truststore, the JKS keystore of public keys for all servers the client can use. This keystore should not be password-protected.

Setting the Key Size for AES and Blowfish

For algorithms like AES, especially with large keys, you may need Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files from Sun or equivalent for your JDK. This enables encryption of client credentials with challenge-response from server to client to prevent replay and other attacks. It also enables challenge-response from client to server to avoid server-side replay attacks.

For the AES and Blowfish algorithms, you can specify the key size for the `security-client-dhalgo` property by adding a colon and the size after the algorithm specification, like this:

```
security-client-dhalgo=AES:192
```

- For AES, valid key size settings are:
  - AES:128
  - AES:192
  - AES:256
- For Blowfish, you can set the key size between 128 and 448 bits, inclusive.

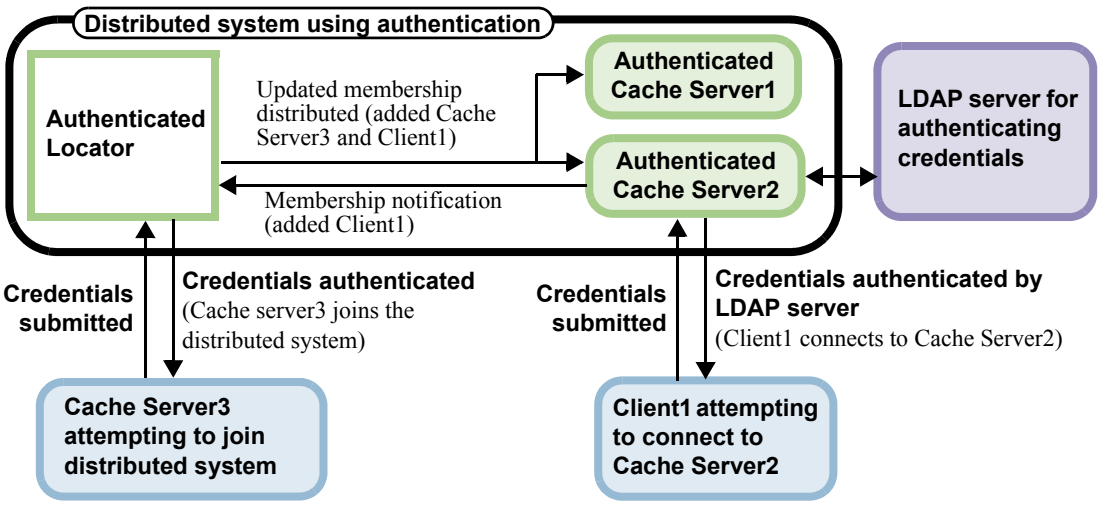
How Authentication Works

With authentication, joining members provide credentials to existing members who check the credentials and either reject the joining member or approves it, returning a `java.security.Principal` object for it, which identifies the member in future operations.

- Clients are authenticated by their server during the connection initialization and for each operation request.
- Joining peer members are authenticated by the locator to which they connect.
- Gateways mutually authenticate each other when they connect.
- Servers may be authenticated by their clients during the connection initialization.

Locators maintain and distribute the authenticated member list. The distributed member list is also authenticated by all members, which prevents an unauthorized application from introducing itself into membership by distributing a member list that includes itself.

Figure 5.1 GemFire Authentication



GemFire authentication provides a flexible plug-in framework. Any security infrastructure can be plugged in to the system as long as the plug-ins implement the required GemFire interfaces.

How Client Authentication Works

The GemFire client can connect in two different ways:

- Process level. Each pool creates a configured minimum number of connections across the server group. The pool accesses the least loaded server for each cache operation. This type of connection is required.

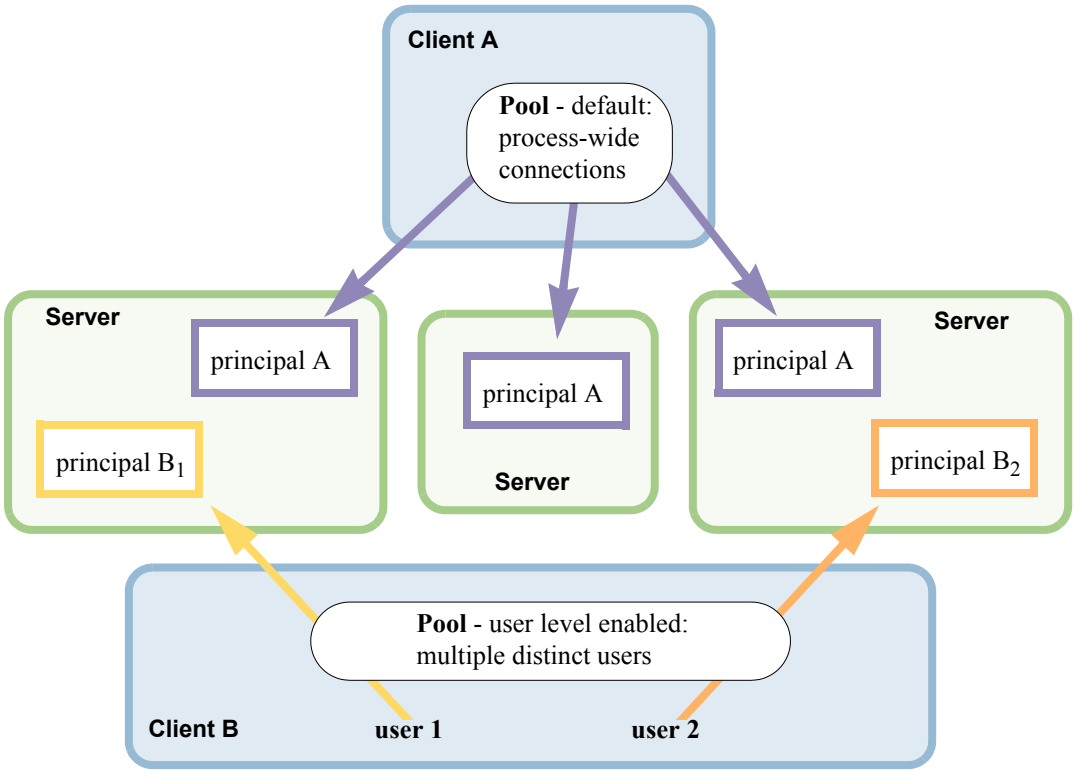
  Process level connections represent the overall client process and are the default way a client accesses the server cache.

- User level. Each client user/pool pair creates a connection to one server and then sticks with it for operations. If the server is unable to respond to a request, the pool selects a new one for the user. This type of connection is created from the process level connection.

  These connections represent individual users established within the client process. These connections are generally used by application servers or web servers that act as clients to GemFire servers. A single application or web server process can service a large number of users, each with their own unique identification and with varied access permissions.

By default, the server pools in clients use process level authentication. You can enable user level authentication by setting the pool's `multiuser-authentication` attribute to `true`. Process level and user level pools can be used inside one client if needed.

Figure 5.2 Client Connections



Client Authentication Process

The client authentication process occurs for each connection established by a pool, regardless of whether the pool is configured for process-wide or single user connections. All credentials are checked for each connection between client and server, including the server-to-client notification channel.

- When the client requests a new connection:
  - The server authenticates the client's credentials and assigns it an internal principal, used to authorize client operations in the server cache
  - The server generates a random unique identifier and returns it to the client to use in its next request
- For each operation request after the initial connection is established:
  - The client sends the request with the unique identifier it received from the server in the last communication.
  - The server verifies the identifier and processes the request, then responds with a new randomly generated unique identifier, for the client to include in its next request.

This ever-changing identifier provides protection against replay attacks, because each client request must include the unique identifier. The server never processes the same request twice. For the most secure communication, add encryption, like Diffie-Hellman.

If the connection fails after the client has sent a request and before the server can respond, the next server request fails due to an invalid unique identifier, and the client pool automatically establishes a new connection to the server system for the client.

5.4 Authentication Examples

This topic discusses the concepts and configurations for sample LDAP and PKCS implementations. Descriptions of their interfaces, classes, and methods are available in the online Java API documentation.

> *Disclaimer: The security samples serve only as example implementations. The implementation and its source code is provided on an "as-is" basis, without warranties or conditions of any kind, either express or implied. You can modify these samples to suit your specific requirements and security providers. GemStone Systems, Inc. takes no responsibility and accepts no liability for any damage to computer equipment, companies, or personnel that might arise from the use of these samples.*

Using an LDAP Server for Client and Peer Authentication

The LDAP sample code in the templates/security directory is `UserPasswordAuthInit.java`, `LdapUserAuthenticator.java`, and `UsernamePrincipal.java`.

In the example, a client or joining peer submits its credentials to a server or locator, which in turn submits the credentials to the LDAP server. To be authenticated, the credentials must match one of the valid entries in the LDAP server. If the submitted credentials result in a connection to the LDAP server, then the connection is authenticated. If the connection to the LDAP server fails, an `AuthenticationFailedException` is sent back and the client or peer connection fails.

These are the `gemfire.properties` file settings for the client, and for all peers in the server system, including the servers and locators.

- Client

```
security-client-auth-init=templates.security.UserPasswordAuthInit.create
security-username="username"
security-password="password"
```

- Server system members

```
security-peer-auth-init=templates.security.UserPasswordAuthInit.create
security-peer-authenticator=
templates.security.LdapUserAuthenticator.create
security-ldap-server="name of ldap server"
security-ldap-basedn="ou=www, dc=xxx, dc=yyy, dc=zzz"
```

LDAP authentication and authorization requires the LDAP server to have entries for each member that is authenticated by the system. The server also requires information to authorize or reject operations by authenticated clients when the authorization callback is invoked.

During the client authentication process, a server searches for a specific entry in the LDAP server. The uid and password parameters submitted by the client are used to search the entries in the LDAP server. The LDAP authenticator is initialized with an LDAP base DN, which is the top level for the LDAP directory tree. The authenticator is also provided with the LDAP server name. The LDAP authenticator can be initialized to make a secure connection by setting the `security-ldap-usessl` property to true.

The sample `LdapUserAuthenticator` class implements the `Authenticator` interface, which verifies the credentials provided in the properties as specified in member ID and returns the principal associated with the client. The `init` method for `LdapUserAuthenticator` gets the LDAP server name from the `security-ldap-server` property in `gemfire.properties`. It also gets the LDAP server base DN name from the `security-ldap-basedn` property, and SSL usage information from the `security-ldap-usessl` property.

Using PKCS for Encrypted Client Authentication

The PKCS sample code in the `templates/security` directory is `PKCSAuthInit.java`, `PKCSAuthenticator.java`, and `PKCSPrincipal.java`.

With this sample, clients send encrypted authentication credentials to a GemFire cache server when they attempt to connect to the server. The credentials are the alias name and digital signature created using the private key retrieved from the provided keystore. The server uses a corresponding public key to decrypt the credentials. If decryption is successful, the client is authenticated and it connects to the server. An unsuccessful decryption generates an `AuthenticationFailedException` that is sent to the client, and the client connection to the server is closed.

These are the `gemfire.properties` file settings for client and server.

- Client

```
security-client-auth-init=templates.security.PKCSAuthInit.create
security-keystorepath=”keystore path”
security-alias=”alias”
security-keystorepass=”keystore password”
```

- Server

```
security-client-authenticator=templates.security.PKCSAuthenticator.create
security-publickey-filepath=”path and name of public key file”
security-publickey-pass=”password of public key file store on the server”
```

The authenticator gets the path to the truststore from the `security-publickey-filepath` property in `gemfire.properties`.

When the client requires authentication, `PKCSAuthInit` gets the alias retrieved from the `security-alias` property, and the keystore path from the `security-keystorepath` property. `PKCSAuthInit` gets the password for the keystore file from the `security-keystorepass` property so the keystore can be opened.

You can generate keys for encryption using the Java `keytool` utility, which is a key and certificate management utility located in the `jre/bin` directory of your Java JDK or JRE installation. The `keytool` utility manages a keystore, or database, of private keys and their associated X.509 certificate chains for authenticating the corresponding public keys. Certificates from trusted entities are also managed using `keytool`. See the Security Tools section at http://java.sun.com/javase/6/docs/technotes/tools for more information about using `keytool`. The public keys from the client keystores should be provided in the public keystore that is referenced by the `security-publickey-filepath` property.

These are the steps to provide the keys, with example utility invocations:

1. Generate a public and private key pair for the client:

```
keytool -genkey \
-alias gemfire8 \
-storetype PKCS12 \
-keyalg RSA \
-keysize 2048 \
-keystore gemfire8.keystore
```

2. Export the self-signed certificate:

```
keytool -export \
-alias gemfire8 \
-keystore gemfire8.keystore \
-rfc \
-file gemfire8.cer
```

3. Import the signed certificate to the truststore:

```
keytool -import \
-alias gemfire8 \
-file gemfire8.cer \
-keystore certificatetruststore
```

Multiple certificates can be imported to the same truststore. The alias name used to generate the key pair and the alias name used to import the certificate to the truststore can be different, but the PKCS sample implementation assumes that both are the same. The credentials authenticator reads the truststore file and loads all the public keys from the truststore, along with the alias names.

5.5 Implementing Authorized Access Control for the Cache

Authorization is available for client/server systems. To use it, your client connections must be authenticated by their servers, as described in previous sections. To set up authorized access control for the cache:

1. Determine the degree of control you want over client access to the server cache.
2. Program and configure the authorization plug-in:

   2.1 Create an implementation of the `com.gemstone.gemfire.security.AccessControl` interface:

   - Program a public static method to return an instance of the class.
   - Program the `init` method to store all properties required by the `AccessControl.authorizeOperation` method at the time the client makes its connection to the server.

     Do as much work here as you can to save time on the individual calls to `authorizeOperation`.

     See the Javadocs for `com.gemstone.gemfire.security.AccessControl`.

   - Program the `authorizeOperation` method to perform whatever pre- and post-operation authorization activities required by your application. For all but function calls, you can filter the post-operation results, to remove any data you do not want your clients to receive. Function calls can only be allowed or disallowed in their entirety.

     The `OperationContext` has the `OperationCode` and a boolean indicating whether the call is pre-operation or post-operation.

   2.2 Set the `gemfire.properties` file uniformly on all servers to implement the plug-in:

   - For pre-operative calls, set `security-client-accessor` to the fully qualified name of the static method you programmed to return an instance of the class.

     ```
     // Pre-op example where myAccessControl.create returns the
     // instance of AccessControl
     security-client-accessor=myAuthPkg.myAccessControl.create
     ```

   - For post-operative calls, set security-client-accessor-pp to the fully qualified name of the static method you programmed to return an instance of the class.

     ```
     // Post-op example where myAuthInitImpl.create returns the
     // instance of AuthInitialize
     security-client-accessor-pp=myAuthPkg.myAccessControl.create
     ```

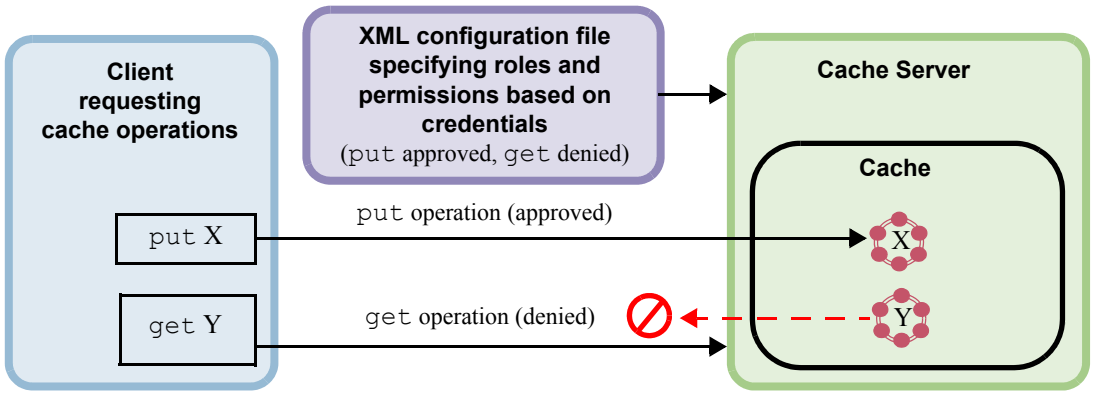
     Your `authorizeOperation` method will be invoked before and after each client operation.

How Authorization Works

The security framework establishes trust between members during authentication. In a client/server system, you can use this trust to grant or withhold a client's cache access and modification requests.

Access rights can be checked before the client operation is performed and before results of the operation are sent back to the client. Access control is done according to your configurations and programmatic plug-ins.

Figure 5.3 GemFire Authorization



The principal, which you associate with the client when it is authenticated, is used by the authorization plug-in to allow or disallow the operation. GemFire security invokes this callback with the principal and the requested operation, and permits or bars the operation depending on the result of the callback. The callback also has access to the operation data, such as the key and value for a `put`, which you can use to determine authorization. In addition, you can program the callback to change some of the operation data, such as the value for a `put` or the operation result.

All client operations sent to the server can be authorized. The operations checked by the server are listed in `com.gemstone.gemfire.cache.operations.OperationContext.OperationCode`.

`Region` *query shortcut methods are all sent to the server as* `query` *operations.*

All client operations that return a result (like `get` and `query`) and notifications can also be authorized in the post-operation phase where the callback can peek and even modify the result being sent out.

Performance

With each new connection request from the client, the system authenticates the connection, instantiates the callback for authorization of requests coming in on that connection, and instantiates the callback for updates sent on the connection from the server to the client.Program to cache as much information as you can in the `AccessControl.init` method phase for quick authorization of each operation on the connection. Then you can use the cached information in `AccessControl.authorizeOperation`, which is called for every client operation. The efficiency of the `authorizeOperation` method directly affects the overall throughput of the GemFire cache.

Programming Considerations

Authorization in the post-operation phase occurs after the operation is complete and before the results are sent to the client. If the operations are not using `FunctionService`, the callback can modify the results of certain operations, such as `query`, `get` and `keySet`. For example, a post-operation callback for a query operation can filter out sensitive data or data that the client should not receive. For all

operations, the callback can completely disallow the operation. However, if the operations are using `FunctionService`, the callback cannot modify the results of the operations, but can only completely allow or disallow the operation.

With querying, the regions used in the query are obtained in the initial parsing phase. The region list is then passed to the post-operation callback unparsed. In addition, this callback is invoked for updates that are sent by the server to the client on the notification channel. This includes updates from a continuous query registered on the server by the client. The operation proceeds if it is allowed by the callback; otherwise a `NotAuthorizedException` is sent back to the client and the client throws the exception back to the caller.

For more advanced requirements like per-object authorization, you could modify the cache value in a `put` operation by the callback in the pre-operation phase to add an authorization token. This token would be propagated through the cache to all cache servers. The token can then be used for fast authorization during region `get` and `query` operations, and it can be removed from the object by changing the operation result. This makes the entire process completely transparent to the clients.

5.6 Authorization Example

This topic discusses the authorization example provided in the product under `templates/security` using `XmlAuthorization.java`, `XmlErrorHandler.java`, and `authz6_0.dtd`.

> *Disclaimer: The security samples serve only as example implementations. The implementation and its source code is provided on an "as-is" basis, without warranties or conditions of any kind, either express or implied. You can modify these samples to suit your specific requirements and security providers. GemStone Systems, Inc. takes no responsibility and accepts no liability for any damage to computer equipment, companies or personnel that might arise from the use of these samples.*

`XmlAuthorization` provides authorization for each region at the operation level by using the permissions specified in an XML file. The sample implementation also shows the post-authorization implementation for the function execution operation. For pre-operation, all the required values are available.

You can configure authorization for all server region operations on a per-region and per-operation basis by using a role-based mechanism. A role can be provided with permissions to execute operations for each region. Each principal name can be associated with a set of roles.

Information such as the region reference, arguments, the operation being invoked, and a reference to the cache instance can be made available to the `XmlAuthorization` callback. If an authenticated client is not authorized to perform an operation, the operation fails with a `NotAuthorizedException`.

Server Settings

These are the `gemfire.properties` file settings for each server:

```
security-client-accessor=templates.security.XmlAuthorization.create
security-authz-xml-uri=<URI of XML file>
```

XML File Sample Settings

The `XmlAuthorization` sample is configured through an XML file, which is described in the `authz6_0.dtd` in the product's `templates/security` directory. See the dtd for documentation about the elements and attributes you use to configure `XmlAuthorization`. To run the example, create an XML file following the dtd specifications.

The user names you use should be the strings returned by the `Principal.getName` method of the `Authenticator` configured on the server.

This section lists an example XML file for the dtd. The example defines these five roles:

1. `reader`
2. `writer`
3. `cacheOps`
4. `queryRegions`
5. `onRegionFunctionExecutor`

The listing below is a sample XML file:

- The permissions for each of the roles are described in the permission tags.
- The `reader`, `writer`, and `cacheOps` roles have no regions mentioned, so they apply to all regions.
- The `queryRegions` role has permissions on `Portfolios` and `Positions` regions.
- The role of `onRegionFunctionExecutor`, can only operate on regions `secureRegion` and `Positions`, and only with functions with ids `SecureFunction` or `OptimizationFunction`, where `optimizeForWrite` is `false`, and `keySet` is `KEY-0` and `KEY-1`.

Example 5.1 Sample XML for Authorization

```
<!DOCTYPE acl PUBLIC
"-//GemStone Systems, Inc.//GemFire XML Authorization 1.0//EN"
"http://www.gemstone.com/dtd/authz6_0.dtd">

<acl>
   <role name="reader">
      <user>reader</user>
      <user>admin</user>
   </role>
   <role name="writer">
      <user>writer</user>
      <user>admin</user>
   </role>
   <role name="cacheOps">
      <user>admin</user>
   </role>
   <role name="queryRegions">
      <user>query</user>
   </role>
   <role name="onRegionFunctionExecutor">
      <user>admin</user>
   </role>
   <permission role="cacheOps">
      <operation>QUERY</operation>
      <operation>EXECUTE_CQ</operation>
      <operation>STOP_CQ</operation>
      <operation>CLOSE_CQ</operation>
      <operation>REGION_CREATE</operation>
      <operation>REGION_DESTROY</operation>
   </permission>
   <permission role="reader">
      <operation>GET</operation>
      <operation>REGISTER_INTEREST</operation>
      <operation>UNREGISTER_INTEREST</operation>
      <operation>KEY_SET</operation>
      <operation>CONTAINS_KEY</operation>
   </permission>
   <permission role="writer">
      <operation>PUT</operation>
      <operation>DESTROY</operation>
      <operation>REGION_CLEAR</operation>
   </permission>
   <permission role="queryRegions" regions="/Portfolios,Positions">
      <operation>QUERY</operation>
      <operation>EXECUTE_CQ</operation>
      <operation>STOP_CQ</operation>
```

```
            <operation>CLOSE_CQ</operation>
        </permission>
        <permission role="onRegionFunctionExecutor" regions=
    "secureRegion,Positions"
            <operation functionIds="SecureFunction,OptimizationFunction"
                optimizeForWrite="false" keySet="KEY-0, KEY-1">EXECUTE_FUNCTION
            </operation>
        </permission>
    </acl>
```

5.7 Configuring SSL

For mutual authentication between members and to protect your data during distribution, you can configure GemFire Enterprise to use the secure sockets layer (SSL) protocol. If configured, SSL is used for all stream-socket communication. GemFire uses SSL connections from the Java Secure Sockets Extension (JSSE) package.

You can use SSL alone or in conjunction with the other GemFire security options.

1. Make sure your Java installation includes the JSSE API and familiarize yourself with its use. For information, see the Sun JSSE website http://java.sun.com/javase/technologies/security.
2. Configure your security provider:
   2.1 Specify the SSL provider in the `lib/security/java.security` file under your JRE home directory. Indicate the providers you are using for your certificate, protocol, and cipher suites. Your Java installation should include information on how to modify this file for this. The security file is usually self-documenting.
   2.2 Specify provider-required configuration settings. These are usually keystore and truststore configurations. Your provider documentation should include specific configuration requirements.
3. Configure your distributed system members for SSL:
   3.1 Use locators for member discovery within the distributed systems and for client discovery of servers.
   3.2 Configure all system members for SSL communication. In `gemfire.properties`, set:

```
ssl-enabled=true
ssl-protocols=any
```

   To use SSL for mutual authentication, in `gemfire.properties`, set:

```
ssl-require-authentication=true
```

   and set one of the following:

```
ssl-ciphers=SSL_RSA_WITH_NULL_SHA
ssl-ciphers=SSL_RSA_WITH_NULL_MD5
ssl-ciphers=SSL_RSA_WITH_NULL_MD5 SSL_RSA_WITH_NULL_SHA
```

   There must be a space between the ciphers, if you use both.

How SSL Works

SSL protects your data in transit between applications.

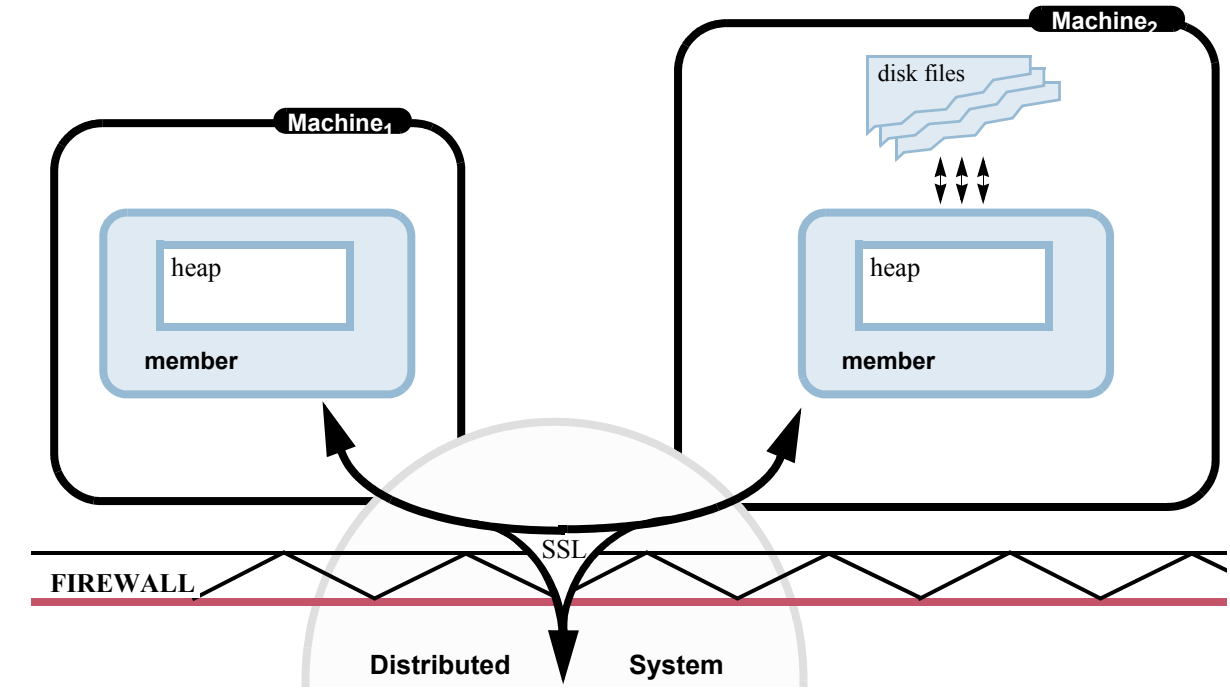
To be secure, the data that is cached in a GemFire Enterprise system must be protected during storage, distribution, and processing. At any time, data in a distributed system may be in one or more of these locations:

- In memory
- On disk
- In transit between processes (for example, in an internet or intranet)

For the protection of data in memory or on disk, GemFire Enterprise relies on your standard system security features such as firewalls, operating system settings, and JDK security settings.

For in transit data, the SSL implementation ensures that only the applications identified by you can share distributed system data. In this figure, the data in the visible portion of the distributed system is secured by the firewall and by security settings in the operating system and in the JDK. The data in the disk files, for example, is protected by the firewall and by file permissions. Using SSL for data distribution provides secure communication between GemFire Enterprise system members inside and outside the firewalls.

Figure 5.4 GemFire Enterprise Security Components With SSL Distribution



SSL Sample Configuration

This is a very simple example of the configuration and startup of GemFire system components with SSL.

Provider-Specific Configuration File

This example uses a keystore created by the Java `keytool` application to provide the proper credentials to the provider. To create the keystore, we ran the following:

```
keytool -genkey \
-alias self \
-dname "CN=trusted" \
-validity 3650 \
-keypass password \
-keystore ./trusted.keystore \
-storepass password \
-storetype JKS
```

This creates a `./trusted.keystore` file to be used later in `config.xml` and command-line configuration settings.

gemfire.properties File

We set these SLL-related setting in the `gemfire.properties`:

```
ssl-enabled=true
mcast-port=0
locators=hostaddress[port]
```

Locator Startup

Before starting other system members, we started the locator with the SSL and provider-specific configuration settings. The provider-specific settings are passed in at the command line and must point to the keystore created using `keytool`. Here are the startup commands:

```
cp ./gfmanager/gemfire.properties .
gemfire start-locator -dir=‘pwd‘ \
-Djavax.net.ssl.keyStore=./trusted.keystore \
-Djavax.net.ssl.keyStorePassword=password \
-Djavax.net.ssl.trustStore=./trusted.keystore \
-Djavax.net.ssl.trustStorePassword=password
```

Other Member Startup

Applications and cacheservers can be started similar to the locator startup, with the appropriate `gemfire.properties` file and the same properties provided at the command line.

Notes

The Java keystore (JKS) provider used in this example requires system properties describing how to access the keystore and truststore. This is risky, as it places secret information on the command line. Most third party providers support more secure sign-on mechanisms.

5.8 Security Logging

You can configure security logging separate from the central logs with these `gemfire.properties` settings.

Table 5.1 System Properties for Security Logging

`security-log-file`	Sets the name of the log file for security log messages. If this property is not specified, the log file specified in the `log-file` property is used for security logging.
`security-log-level`	Specifies the logging level detail for the security log messages. The default log level is `config`.

All security log lines are prefixed with `security-`, like `[security-warning <warning log entry>`. This lets you filter security related logging, if you use the central log file.

For general information on logging, see Chapter 10, *GemFire System Logging*, on page 185.

Security Event Logging Levels

Security-related events are logged as follows.

Warning

- Unsuccessful authentication attempts in the server logs for clients, and locator logs for peers (the locator or locators that acted as the group coordinator).
- Authorization failure for an operation in the server logs.

Info

- Successful authentication attempts in the server logs for clients, and locator logs for peers (the locator or locators that acted as the group coordinator).
- Successful initialization of the `AccessControl` callback for client connections.

Finest

Do not use this level unless asked to do so by GemStone GemFire support. This level generates very large log files.

- Successful authorization for each operation.

Chapter 6

Managing Disk Stores

GemFire Enterprise® disk stores enable you to persist data as a backup to the in-memory copy and allow you to overflow data when memory use gets too high, by using disk stores as an extension of the in-memory cache. These two options can be used individually or together.

In this chapter:

6.1 Introduction to Disk Stores

Disk storage is available for these cached data types:

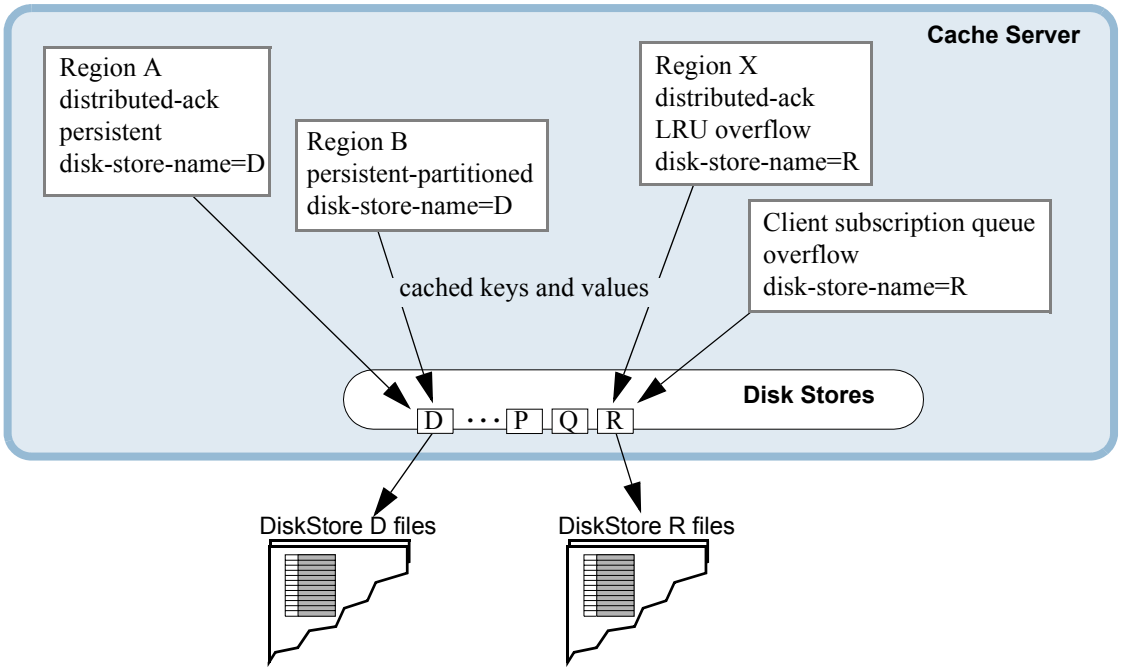
- **Cached regions**. Persist and/or overflow data from your cached data regions. See *Using Disk for Backup and Overflow* on page 170 of the *GemFire Enterprise Developer's Guide*.
- **Server's client subscription queues**. Overflow the messaging queues to control memory use. See *Overflowing the Client Queue to Disk* on page 257 of the *GemFire Enterprise Developer's Guide*.
- **Gateway messaging queues**. Persist these for high availability. These queues always overflow. See *Gateway Queue Persistence* on page 275 of the *GemFire Enterprise Developer's Guide*.

You define disk stores at the cache level. Each disk store can be used by multiple regions and queues.

Each member has its own set of disk stores, completely separate from the disk stores of any other member. For each disk store, you define where and how the data is stored to disk. You can store data from multiple regions and queues in a single disk store.

This figure shows a member with disk stores D through R defined. The member has two persistent regions using disk store D and an overflow region and an overflow queue using disk store R.

Figure 6.1 Disk Store Use



What GemFire Writes to the Disk Store

- List of members that host the store and information on their status, such as running or offline and time stamps
- List of regions in the disk store
- For each region:
  - Region configuration attributes pertaining to loading and capacity management, used to load the data quickly on startup
  - Region data operations

Disk Store State

Disk store access and management differs according to whether the store is online or offline.

While a member is running, its disk stores are online in the GemFire system. When the member closes its cache and exits, its disk stores go offline. When the member starts up again, its disk stores come back online.

Online

Online, a disk store is owned and managed by its member process. To run operations on an online disk store, use API calls in the member process or use the `gemfire` command-line tool. The tool joins the distributed system and sends requests to members that have disk stores.

Offline

Offline, the disk store is just a collection of files in your host file system. The files are open to access by anyone with the right file system permissions. You can copy the files using your file system commands, for backup or to move your member's disk store location. You can also run some GemFire maintenance operations on the offline disk store, like file compaction and validation.

> *The files for a disk store are used by GemFire as a group. Treat them as a single entity. If you copy them, copy them all together. Do not change the file names.*

When a disk store is offline, its data is unavailable to the GemFire distributed system. For partitioned regions, the data is split between multiple members, so you may be able to access the region, but have some of its data only present in an offline disk store. If you try to access an entry whose only copy is stored on disk by an offline member, the operation returns a `PartitionOfflineException`.

Disk Store File Names and Extensions

Disk store files include store management and access control files and the operation log, or oplog, files, consisting of one file for deletions and another for all other operations. The next tables describe file names and extensions; they are followed by example disk store files.

File name part	Values	Used for	Examples
First part: usage identifier	`OVERFLOW`	Oplog data from overflow regions and queues only.	`OVERFLOWoverflowDS1_1.crf`
	`BACKUP`	Oplog data from persistent and persistent+overflow regions and queues.	`BACKUPoverflowDS1.if` `BACKUPDEFAULT.if`
	`DRLK_IF`	Access control - locking the disk store.	`DRLK_IFoverflowDS1.lk` `DRLK_IFDEFAULT.lk`

Second part: disk store name	*name specified by you*	Non-default disk stores.	name=”`overflowDS1`”
			`DRLK_IFoverflowDS1.lk`
			
			name=”`persistDS1`”
			`BACKUPpersistDS1_1.crf`

	`DEFAULT`	Default disk store name, used when persistence or overflow are specified on a region or queue but no disk store is named.	`DRLK_IFDEFAULT.lk` `BACKUPDEFAULT_1.crf`

Third part: oplog sequence number	Sequence number in the format `_n`	Oplog data files only. Numbering starts with 1.	`OVERFLOWoverflowDS1_1.crf`
			`BACKUPpersistDS1_2.crf`
			`BACKUPpersistDS1_3.crf`

.

File extension	Used for	Notes
`if`	Disk store metadata	Stored in the first `disk-dir` listed for the store. Negligible size - not considered in size control.
`lk`	Disk store access control	Stored in the first `disk-dir` listed for the store. Negligible size - not considered in size control.
`crf`	Oplog: create, update, and invalidate operations	Pre-allocate 90% of the total `max-oplog-size` at creation.
`drf`	Oplog: delete operations	Pre-allocate 10% of the total `max-oplog-size` at creation.

Example 6.1 Example files for Disk Stores persistDS1 and overflowDS1

```
bash-2.05$ ls -tlra persistData1/
total 8
-rw-rw-r--   1 jpearson users        188 Mar  4 06:17 BACKUPpersistDS1.if
drwxrwxr-x   2 jpearson users        512 Mar  4 06:17 .
-rw-rw-r--   1 jpearson users          0 Mar  4 06:18 BACKUPpersistDS1_1.drf
-rw-rw-r--   1 jpearson users         38 Mar  4 06:18 BACKUPpersistDS1_1.crf
drwxrwxr-x   8 jpearson users        512 Mar  4 06:20 ..
bash-2.05$

bash-2.05$ ls -ltra overflowData1/
total 1028
drwxrwxr-x   8 jpearson users        512 Mar  4 06:20 ..
-rw-rw-r--   1 jpearson users          0 Mar  4 06:21 DRLK_IFoverflowDS1.lk
-rw-rw-r--   1 jpearson users          0 Mar  4 06:21 BACKUPoverflowDS1.if
-rw-rw-r--   1 jpearson users    1073741824 Mar  4 06:21 OVERFLOWoverflowDS1_1.crf
drwxrwxr-x   2 jpearson users        512 Mar  4 06:21 .
```

Example 6.2 Default Disk Store Files for Persistent Region

```
bash-2.05$ ls -tlra
total 106
drwxrwxr-x   8 jpearson users       1024 Mar  8 14:51 ..
-rw-rw-r--   1 jpearson users       1010 Mar  8 15:01 defTest.xml
drwxrwxr-x   2 jpearson users        512 Mar  8 15:01 backupDirectory
-rw-rw-r--   1 jpearson users          0 Mar  8 15:01 DRLK_IFDEFAULT.lk
-rw-rw-r--   1 jpearson users  107374183 Mar  8 15:01 BACKUPDEFAULT_1.drf
-rw-rw-r--   1 jpearson users  966367641 Mar  8 15:01 BACKUPDEFAULT_1.crf
-rw-rw-r--   1 jpearson users        172 Mar  8 15:01 BACKUPDEFAULT.if
drwxrwxr-x   3 jpearson users        512 Mar  8 15:01 .
```

Disk Store Operation Logs

At creation, each operation log is initialized at the max-oplog-size (page 101), with the size divided between the `crf` and `drf` files. When the oplog is closed, GemFire shrinks the files down to the space used in each file.

When an operation log is full, GemFire automatically closes it and creates a new log with the next sequence number. This is called oplog rolling. You can also request an oplog rolling through the API call `DiskStore.forceRoll`. You may want to do this immediately before compacting your disk stores, so the latest oplog is available for compaction.

See Disk Store File Names and Extensions on page 98.

Log compaction can change the names of the disk store files. File number sequencing is usually altered, with some existing logs removed or replaced by newer logs with higher numbering. See Compacting Disk Store Log Files on page 112. GemFire always starts a new log at a number higher than any existing number.

This example listing shows the logs in a system with only one disk directory specified for the store. The first log (`BACKUPCacheOverflow_1.crf` and `BACKUPCacheOverflow_1.drf`) has been closed and the system is writing to the second log:

Example 6.3 Files After One Operation Log Roll, and After the Files Are Closed

```
bash-2.05$ ls -tlra
total 55180
drwxrwxr-x   7 jpearson users        512 Mar 22 13:56 ..
-rw-rw-r--   1 jpearson users          0 Mar 22 13:57 BACKUPCacheOverflow_2.drf
-rw-rw-r--   1 jpearson users     426549 Mar 22 13:57 BACKUPCacheOverflow_2.crf
-rw-rw-r--   1 jpearson users          0 Mar 22 13:57 BACKUPCacheOverflow_1.drf
-rw-rw-r--   1 jpearson users     936558 Mar 22 13:57 BACKUPCacheOverflow_1.crf
-rw-rw-r--   1 jpearson users       1924 Mar 22 13:57 BACKUPCacheOverflow.if
drwxrwxr-x   2 jpearson users       2560 Mar 22 13:57 .
```

The system rotates through all available disk directories to write its logs. The next log is always started in a directory that has not reached its configured capacity, if one exists.

When Oplogs Reach the Configured Disk Capacity

If no directory exists that is within its capacity limits, how GemFire handles this depends on whether automatic compaction is enabled.

- If auto-compaction is enabled, GemFire creates a new oplog in one of the directories, going over the limit, and logs a warning that reports:

  ```
  Even though the configured directory size limit has been exceeded a
  new oplog will be created. The current limit is of XXX. The current
  space used in the directory is YYY.
  ```

  So when auto-compaction is enabled, `dir-size` does not limit how much disk space is used. GemFire will perform auto-compaction, which should free space, but the system may go over the configured disk limits.

- If auto-compaction is disabled, GemFire does not create a new oplog, operations in the regions attached to the disk store block, and GemFire logs this error:

  ```
  Disk is full and rolling is disabled. No space can be created
  ```

6.2 Configuring Disk Stores

You can store data to disk without configuring any disk stores in your member. If you do this, GemFire uses the default disk store. The default name is `DEFAULT` and it is initially configured with all of the default disk store settings. See *Using the Default Disk Store* on page 105.

Disk Store Configuration Parameters

You define your disk stores in `<disk-store>` subelements of your cache declaration in `cache.xml`. For information on the `cache.xml` file, see *cache.xml File* on page 499 of the *GemFire Enterprise Developer's Guide*. All disk stores are available for use by all of your regions and queues. These `<disk-store>` attributes and subelements in have corresponding setter and getter methods in the `DiskStoreFactory` and `DiskStore` APIs.

Table 6.1 Disk store configuration attributes

disk-store attribute	Description	Default
`name`	String used to identify this disk store. All regions and queues select their disk store by specifying this name.	
`allow-force-compaction`	Boolean indicating whether to allow manual compaction through the API or command-line tools.	`false`
`auto-compact`	Boolean indicating whether to automatically compact a file when it reaches the compaction-threshold.	`true`
`compaction-threshold`	Percentage of garbage allowed in the file before it is eligible for compaction. Garbage is created by entry destroys, entry updates, and region destroys and creates. Surpassing this percentage does not make compaction occur—it makes the file eligible to be compacted when a compaction is done.	`50`
`max-oplog-size`	The largest size, in megabytes, to allow an operation log to become before automatically rolling to a new file. This size is the combined sizes of the oplog files. See *Disk Store File Names and Extensions* on page 98.	`1024`
`queue-size`	For asynchronous queueing. The maximum number of operations to allow into the write queue before automatically flushing the queue. Operations block until the queue is flushed. A value of zero implies no size limit. Reaching this limit or the `time-interval` limit will cause the queue to flush.	`0`
`time-interval`	For asynchronous queueing. The number of milliseconds that can elapse before data is flushed to disk. Reaching this limit or the `queue-size` limit causes the queue to flush.	`1000`
`write-buffer-size`	Size of the buffer used to write to disk.	`32768`
disk-store subelement	**Description**	**Default**
`<disk-dirs>`	Defines the system directories where the disk store is written and their maximum sizes.	`. with no size limit`

<disk-dirs> Element

The `<disk-dirs>` element defines the host system directories to use for the disk store. It contains one or more single `<disk-dir>` elements, made up of:

- The directory specification, provided as the text of the `disk-dir` element.
- An optional `dir-size` attribute specifying the maximum amount of space, in megabytes, to use for the disk store in the directory. By default, there is no limit. The space used is calculated as the combined sizes of all oplog files. See *Disk Store File Names and Extensions* on page 98.

You can specify any number of `disk-dir` subelements to the `disk-dirs` element. The data is spread evenly among the active disk files in the directories, keeping within any limits you set.

Example:

```
<disk-dirs>
   <disk-dir>/host1/users/gf/memberA_DStore</disk-dir>
   <disk-dir>/host2/users/gf/memberA_DStore</disk-dir>
   <disk-dir dir-size="20480">/host3/users/gf/memberA_DStore</disk-dir>
</disk-dirs>
```

> *The directories must exist when the disk store is created or the system throws an exception. GemFire does not create directories.*

Use different `disk-dir` specifications for different disk stores. You cannot use the same directory for the same named disk store in two different members.

The Disk Store API

Use the Java `DiskStoreFactory` to configure and create a disk store and the `DiskStore` API to manage it. Besides setting and accessing configuration, the `DiskStore` API has a few methods for management. See the online Java documentation.

com.gemstone.gemfire.cache

- **DiskStore**.View the disk store configuration and manage the store. You can flush any logs in the asynchronous queue, force rolling to a new oplog, and force a compaction of the disk store logs.
- **DiskStoreFactory**. Configure and create a disk store
- **Cache**. Create a `DiskStoreFactory` and retrieve any `DiskStore` by name.

Related Configurations and APIs

The `cache.xml` and APIs for regions and queues connect your disk stores with your cached data:

- **Regions**. Region attributes include settings to assign the region to a named disk store, `disk-store-name` and to specify whether disk writes are synchronous, `disk-synchronous`. See *Using Disk for Backup and Overflow* on page 170 of the *GemFire Enterprise Developer's Guide*.
- **Server's client subscription queues**. The client subscription `disk-store-name` attribute is used to assign the server client queues to a named disk store. See *Overflowing the Client Queue to Disk* on page 257 of the *GemFire Enterprise Developer's Guide*.
- **Gateway queues**. The Gateway queue `disk-store-name` attribute is used to assign the queue to a named disk store. See *Gateway Queue Persistence* on page 275 of the *GemFire Enterprise Developer's Guide*.

Defining and Setting Up Your Disk Stores

In this procedure it is assumed that you understand how to configure GemFire using `gemfire.properties` and `cache.xml` and that you have your overall system and cache definitions in place. See Chapter 3, *Configuring the System*, on page 41 of this book and *GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide*.

1. Work with your system designers and developers to plan for anticipated disk storage requirements in your testing and production caching systems. Take into account space and functional requirements.

   Besides the disk stores you specify, GemFire has a default disk store in the application's working directory that it uses when disk use is configured with no disk store name specified.

   - For efficiency, separate data that is only overflowed in separate disk stores from data that is persisted or persisted and overflowed. Regions can be overflowed, persisted, or both. Server subscription queues are only overflowed. Gateway queues are always overflowed and may be persisted. Assign them to overflow disk stores if you do not persist, and to persistence disk stores if you do.
   - When calculating your disk requirements, figure in your data modification patterns and your compaction strategy. Obsolete operations are only removed from the oplogs during compaction. You need enough space to store all operations that are done between compactions. See *Compacting Disk Store Log Files* on page 112.

2. Work with your host system administrators to determine where to place your disk store directories, based on your anticipated disk storage requirements and the available disks on your host systems.
   - Make sure the new storage does not interfere with other processes that use disk on your systems. If possible, store your files to disks that are not used by other processes, including virtual memory or swap space. If you have multiple disks available, for the best performance, place one directory on each disk.
   - Use different directories for different members. You can use any number of directories for a single disk store.

3. In the locations you have chosen, create all directories you will specify for your disk stores to use. GemFire throws an exception if the specified directories are not available when a disk store is created. You do not need to populate these directories with anything.

4. Choose disk store names that reflect how the stores should be used and that work for your operating systems. Disk store names are used in the disk file names:
   - Use disk store names that satisfy the file naming requirements for your operating system. For example, if you store your data to disk in a Windows system, your disk store names could not contain any of these reserved characters, `< > : " / \ | ? *`.
   - Do not use very long disk store names. The full file names must fit within your operating system limits. On Linux, for example, the standard limitation is 255 characters.

5. Configure each disk store:

   You can configure the default disk store, along with any others. Use the name "`DEFAULT`*". See* *Using the Default Disk Store* on page 105.

   5.1 Set the name.

   ```
   name="serverOverflow"
   ```

   5.2 Configure the directory locations and the maximum space to use for the store.

   ```
   <disk-dirs>
       <disk-dir>c:\overflow_data</disk-dir>
       <disk-dir dir-size="20480">d:\overflow_data</disk-dir>
   </disk-dirs>
   ```

   5.3 As needed, modify the store's file compaction behavior. In conjunction with this, plan and program for any manual compaction. See *Compacting Disk Store Log Files* on page 112.

   ```
   compaction-threshold="40"
   auto-compact="false"
   allow-force-compaction="true"
   ```

   5.4 As needed, modify the maximum size of a single oplog (see *Disk Store File Names and Extensions* on page 98). When the current files reach this size, the system rolls forward to a new file. You get better performance with relatively small maximum file sizes.

   ```
   max-oplog-size="512"
   ```

   5.5 As needed, modify the queue management parameters for any asynchronous queueing to the disk store. Each region can be configured for synchronous or asynchronous queueing (region attribute `disk-synchronous`). Server client queues and gateway queues operate synchronously.

   When either `queue-size` or `time-interval` is reached, enqueued data is flushed to disk. `DiskStore` also provides a `flushToDisk` method to synchronously write unwritten data to disk.

   ```
   queue-size="10000"
   time-interval="15"
   ```

   5.6 As needed, modify the size of the buffer used for writing to disk:

   ```
   write-buffer-size="65536"
   ```

Complete disk store XML configuration example:

```
<disk-store name="serverOverflow" compaction-threshold="40"
    auto-compact="false" allow-force-compaction="true"
    max-oplog-size="512" queue-size="10000"
    time-interval="15" write-buffer-size="65536">
    <disk-dirs>
        <disk-dir>c:\overflow_data</disk-dir>
        <disk-dir dir-size="20480">d:\overflow_data</disk-dir>
    </disk-dirs>
</disk-store>
```

Using the Default Disk Store

Whenever you use disk without specifying the disk store to use, GemFire uses the disk store named "DEFAULT".

For example, these configurations specify persistence and/or overflow, but do not specify the disk-store-name. Because no disk store is specified, these use the disk store named "DEFAULT":

Example 6.4 Region Persistence and Overflow

```
<region refid="PARTITION_PERSISTENT_OVERFLOW"/>
```

Example 6.5 Gateway queue persistence

```
<gateway-hub id="EU" port="33333">
    <gateway id="US">
      <gateway-endpoint id="US-1" host="ethel" port="11111"/>
      <gateway-queue maximum-queue-memory="50"
          batch-size="100" batch-time-interval="1000"/>
    </gateway>
</gateway-hub>
```

Example 6.6 Server Subscription Queue Overflow

```
<cache-server port="40404">
   <client-subscription eviction-policy="entry" capacity="10000"/>
</cache-server>
```

Changing the Behavior of the Default Disk Store

GemFire initializes the default disk store with the default settings listed in *Disk Store Configuration Parameters* on page 101. You can modify the behavior of the default disk store by specifying the attributes you want for the disk store named "DEFAULT". The only thing you can't change about the default disk store is the name.

This changes the default disk store to allow manual compaction and to use multiple, non-default directories:

```
<disk-store name="DEFAULT" allow-force-compaction="true">
   <disk-dirs>
      <disk-dir>/export/thor/customerData</disk-dir>
      <disk-dir>/export/odin/customerData</disk-dir>
      <disk-dir>/export/embla/customerData</disk-dir>
   </disk-dirs>
</disk-store>
```

6.3 Running a System with Disk Stores

When you use disk stores, keep your system optimized by following these guidelines:

1. When you start up, start all the members that have persistent regions at roughly the same time. Create and use startup scripts for consistency and completeness. For information on how startup works, see *Starting Up With Disk Stores* on page 106.
2. Shut down your system using the `gemfire shut-down-all` command. This is an ordered shutdown that positions your disk stores for a faster startup. See *Shutting Down with Disk Stores* on page 109
3. Decide on a file compaction policy and, if needed, develop procedures to monitor your files and execute regular compaction. See *Compacting Disk Store Log Files* on page 112.
4. Decide on a backup strategy for your disk stores and follow it. You can back up by copying the files while the system is offline or you can back up the online system using the `gemfire` command. See *Backing Up and Restoring a Disk Store* on page 115.
5. If you remove any persistent region or change its configuration while your disk store is offline, consider synchronizing the regions in your disk stores. See *Keeping Your Offline Disk Store In Sync with Your Cache* on page 119.

For general information on system startup and shutdown, see *Starting and Stopping the Distributed System* on page 124.

Starting Up With Disk Stores

Start all of the member processes in parallel, so that they can negotiate to determine which member has the most up-to-date copy of each region's data.

This is an example bash script for starting members in parallel. The script waits for the startup to finish and exits with an error status if one of the jobs fails.

Example 6.7 Sample bash Script for System Startup

```
#!/bin/bash
ssh servera "cd /my/directory; cacheserver start &
ssh serverb "cd /my/directory; cacheserver start &

STATUS=0;
for job in `jobs -p`
do
echo $job
  wait $job;
  JOB_STATUS=$?;
  test $STATUS -eq 0 && STATUS=$JOB_STATUS;
done
exit $STATUS;
```

Most Recent Data from the Last Run

When you shut down a member that is persisting data, the data remains in the disk store files, available to be reloaded when the member starts up again. If more than one member has the same persistent region or queue, the last member to exit leaves the most up-to-date data on disk.

GemFire stores information on member exit order in the disk stores, so it can start your members with the most recent data set:

- For a persistent replicate, the last member to exit leaves the most recent data on disk.
- For a partitioned region, where the data is split into buckets:
  - If you use `gemfire shut-down-all`, all online partitioned region data hosts are synchronized before shutting down so all hold the most recent data copy.
  - Otherwise, different members might host different most recent buckets.

The Startup Process

When you start a member with disk stores, the stores are loaded back into the cache to initialize the member's persistent regions.

- If any region does not hold all most recent data in the system:
  - Region creation is blocked, waiting for the members with the most recent data.

    If your log level is `info` or below, the system provides messaging about the wait. Here, the disk store for `hostA` has the most recent data for the region and the `hostB` member is waiting for it.

    ```
    [info 2010/04/09 10:48:26.039 PDT CacheRunner <main> tid=0x1]
    Region /persistent_PR initialized with data from
    /10.80.10.64:/export/straw3/users/jpearson/GemFireTesting/hostB/
    backupDirectory created at timestamp 1270834766425 version 0 is
    waiting for the data previously hosted at
    [/10.80.10.64:/export/straw3/users/jpearson/GemFireTesting/hostA/
    backupDirectory created at timestamp 1270834763353 version 0] to
    be available
    ```

    During normal startup, especially with partitioned regions that were not shut down using the `gemfire shut-down-all` command, you can expect to see some waiting messages.
  - When the most recent data is available, the system updates the local region as needed, logs a message like this, and continues with startup.

    ```
    [info 2010/04/09 10:52:13.010 PDT CacheRunner <main> tid=0x1]
          Done waiting for the remote data to be available.
    ```
- If the disk store has data for a region that is never created, the data remains in memory. See *Taking a Region Out of Your Cache Configuration* on page 119.

Each member's persistent regions load and go online as quickly as possible, not waiting unnecessarily for other members to complete.

When Member Startup Hangs

If a most recent disk store does not come online, your other members will wait indefinitely rather than come online with stale data.

Check for missing disk stores with `gemfire list-missing-disk-stores` command. See *Listing Missing Disk Stores* on page 120.

- If no disk stores are missing, your cache initialization is slow for some other reason. See *Member Process Seems to Hang* on page 201.
- If disk stores are missing that you think should be there:
  - Make sure you have started the member. Check the logs for any failure messages.
  - Make sure your disk store files are accessible. If you move your member or disk store files, you must update your disk store configuration to match.
- If disk stores are missing that you know about, because you have deleted them or their files are otherwise unavailable, revoke them so the startup can continue. See Revoking a Missing Disk Store (page 121).

Handling Catastrophic Loss of Disk Store Data

If you cannot recover a missing a disk store, revoke it from the system *during startup* so the other members can start. You revoke a disk store by telling online members that a missing member's disk store is no longer the most recent.

Use the `gemfire revoke-missing-disk-store` command, passing it the specifications for the store listed by `gemfire list-missing-disk-stores`. See *Handling Missing Disk Stores* on page 120.

Example Startup Scenarios

Stop Order for a Replicate Persistent Region

1. Member A ($M_{A)}$ exits first, leaving persisted data on disk for RegionP.
2. M_B continues to run operations on RegionP, which update its disk store and leave the disk store for M_A in a stale condition.
3. M_B exits, leaving the most up-to-date data on disk for RegionP.

Restart Order Scenario 1

1. M_B is started. GemFire recognizes M_B as having the most recent disk data for RegionP and initializes it from disk.
2. M_A is started, recovers its data from disk, and updates it as needed from the data in M_B.

Restart Order Scenario 2

1. M_A is started first. GemFire recognizes that M_A does not have the most recent disk data and waits for M_B to start before creating RegionP in M_A.
2. M_B is started. GemFire recognizes M_B as having the most recent disk data for RegionP and initializes it from disk.
3. M_A recovers its RegionP data from disk and updates it as needed from the data in M_B.

Shutting Down with Disk Stores

To shut down:

1. Have all members with persistent disk stores running, if possible
2. Shut down using the `gemfire` command-line tool:

```
gemfire shut-down-all
```

Make sure this `gemfire` *call can find a* `gemfire.properties` *file for the system.*

The tool provides an ordered shutdown to your system that gives you the fastest startup times.

This is particularly useful for persistent partitioned region shutdown, as it synchronizes all of the online partitioned region data before shutdown. This means every disk store has the most recent data and does not require updates from other members at startup.

6.4 The gemfire Command

The `gemfire` command-line tool has a number of options for examining and managing your disk stores. The `gemfire` tool, along with the `cache.xml` file and the `DiskStore` APIs, are the management tools for your online and offline disk stores.

Each of these commands operates either on the online disk stores or offline disk stores:

gemfire command	On or Offline	See . . .
`shut-down-all`	On	*Shutting Down with Disk Stores* on page 109
`validate-disk-store`	Off	*Validating a Disk Store* on page 111
`compact-all-disk-stores`	On	*Compacting Disk Store Log Files* on page 112
`compact-disk-store`	Off	*Compacting Disk Store Log Files* on page 112
`backup`	On	*Backing Up and Restoring a Disk Store* on page 115
`modify-disk-store`	Off	*Keeping Your Offline Disk Store In Sync with Your Cache* on page 119
`list-missing-disk-stores`	On	*Handling Missing Disk Stores* on page 120
`revoke-missing-disk-store`	On	*Handling Missing Disk Stores* on page 120

For complete command syntax for any `gemfire` command, run `gemfire -h <command>` at the command line. The `gemfire` commands are also described in Appendix A, *The gemfire Command-line Utility*, on page 227.

Online Operations

For online operations, `gemfire` connects to a distributed system and sends the operation requests to the members that have disk stores. These commands will not run on offline disk stores.

You must provide the command with a distributed system specification in a `gemfire.properties` file. See *Specifying the Configuration File Locations* on page 42.

Offline Operations

For offline operations, `gemfire` runs the command against the specified disk store and its specified directories. You must specify all directories for the disk store.

This will not run on online disk stores. The tool locks the disk store while it is running, so the member cannot start in the middle of an operation.

If you try to run an offline command for an online disk store, you get a message like this:

```
ERROR: Operation "validate-disk-store" failed because:  disk-store=ds1:
com.gemstone.gemfire.cache.DiskAccessException: For DiskStore: ds1: Could
not lock "hostA/ds1dir1/DRLK_IFds1.lk". Other JVMs might have created
diskstore with same name using the same directory., caused by
java.io.IOException: The file "hostA/ds1dir1/DRLK_IFds1.lk" is being used
by another process.
```

Validating a Disk Store

The `gemfire validate-disk-store` command verifies the health of your offline disk store and gives you information about the regions in it, the total entries, and the number of records that would be removed if you compacted the store. Use this:

- Before compacting an offline disk store to help decide whether it's worth doing.
- Before restoring or modifying a disk store.
- Any time you want to be sure the disk store is in good shape.

Example:

```
gemfire validate-disk-store ds1 hostB/bupDirectory
/partitioned_region entryCount=6 bucketCount=10
Disk store contains 1 compactable records.
Total number of region entries in this disk store is: 6
```

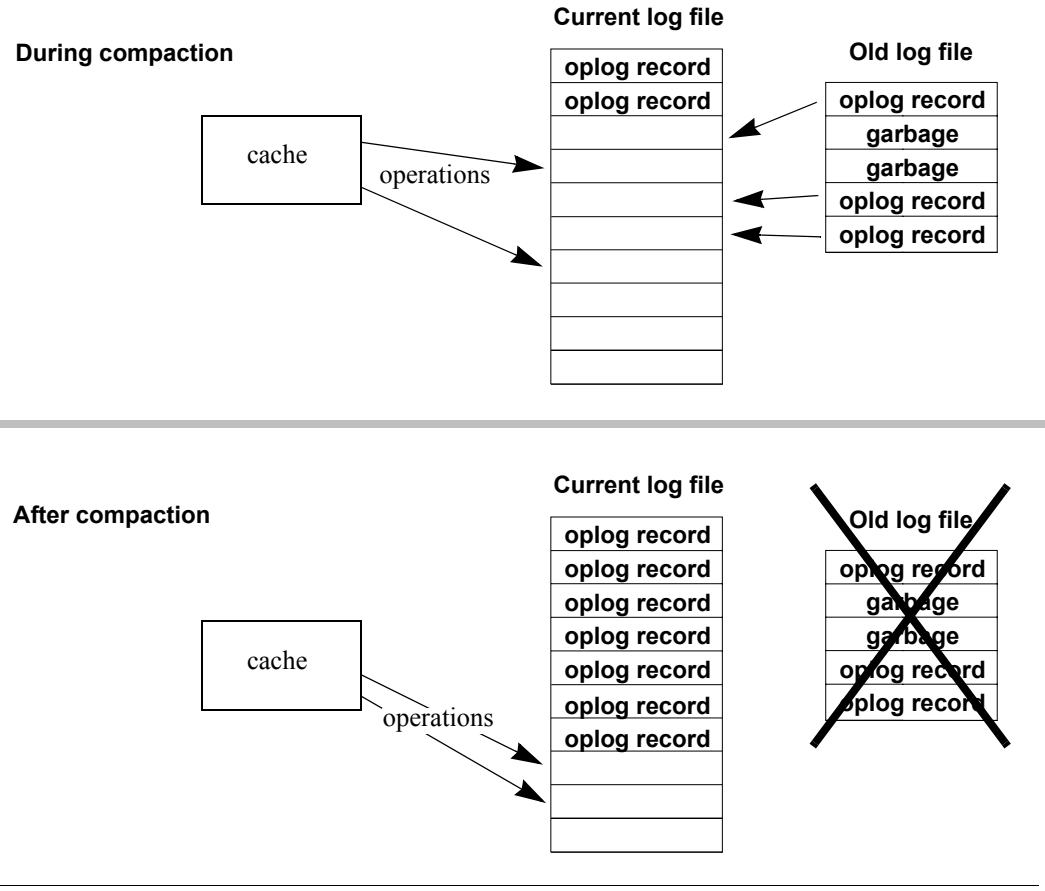
Compacting Disk Store Log Files

When a cache operation is added to a disk store, any preexisting operation record for the same entry becomes obsolete, and GemFire marks it as garbage. For example, when you create an entry, the create operation is added to the store. If you update the entry later, the update operation is added and the create operation becomes garbage. GemFire does not remove garbage records as it goes, but it tracks the percentage of garbage in each operation log, and provides mechanisms for removing garbage to compact your log files.

GemFire compacts an old operation log by copying all non-garbage records into the current log and discarding the old files. As with logging, oplogs are rolled as needed during compaction to stay within the max oplog setting. See *Disk Store Operation Logs* on page 100.

You can configure the system to automatically compact any closed operation log when its garbage content reaches a certain percentage. You can also manually request compaction for online and offline disk stores. For the online disk store, the current operation log is not available for compaction, no matter how much garbage it contains.

Figure 6.2 Online Disk Store Log File Compaction



Offline compaction runs essentially in the same way, but without the incoming cache operations. Also, because there is no current open log, the compaction creates a new one to get started.

Online Compaction

Old log files become eligible for online compaction when their garbage content surpasses a configured percentage of the total file. A record is garbage when its operation is superseded by a more recent operation for the same object. During compaction, the non-garbage records are added to the current log along with new cache operations. Online compaction does not block current system operations.

Automatic

When `auto-compact` (page 101) is `true`, GemFire automatically compacts each oplog when its garbage content surpasses the `compaction-threshold` (page 101).

This takes cycles from your other operations, so you may want to disable this and only do manual compaction, to control the timing.

Manual

To run manual compaction, set `allow-force-compaction` (page 101) to `true`. This causes GemFire to maintain extra data about the files so it can compact on demand. This is disabled by default to save space.

You can run manual online compaction at any time while the system is running. Oplogs eligible for compaction based on the `compaction-threshold` (page 101) are compacted into the current oplog.

You can:

- Compact the logs for a single online disk store through the API:

  ```
  myCache.getDiskStore("myDiskStore").forceCompaction();
  ```

  This method first rolls the oplogs and then compacts them.

- Compact all online disk stores in a distributed system from the command-line:

  ```
  gemfire compact-all-disk-stores
  ```

  Make sure this `gemfire` *call can find a* `gemfire.properties` *file for the system.*

Offline Compaction

Offline compaction is a manual process. All log files are compacted as much as possible, regardless of how much garbage they hold. Offline compaction creates new log files for the compacted log records.

Compact individual offline disk stores following this command syntax.

```
gemfire compact-disk-store myDiskStoreName /firstDir /secondDir
-maxOplogSize=maxMegabytesForOplog
```

You must provide all of the directories in the disk store. If no oplog max size is specified, GemFire uses the system default.

Offline compaction can take a lot of memory. If you get a `java.lang.OutOfMemory` error while running this, you made need to increase your heap size. See the `gemfire` command help for instructions on how to do this.

Performance Benefits of Using Manual Compaction

You can improve performance during busy times if you disable automatic compaction and run your own manual compaction during lighter system load or during downtimes. You could run the API call after your application performs a large set of data operations. You might run `gemfire compact-all-disk-stores` every night when system use is very low.

To follow a strategy like this, you need to set aside enough disk space to accommodate all non-compacted disk data. You might need to increase system monitoring to make sure you do not overrun your disk space. You may be able to run only offline compaction. If so, you can set `allow-force-compaction` to `false` and avoid storing the information required for manual online compaction.

Directory Size Limits

If you reach the disk directory size limits during compaction:

- For automatic compaction, the system logs a warning, but does not stop.
- For manual compaction, the operation stops and returns a `DiskAccessException` to the calling process, reporting that the system has run out of disk space.

Example

In this example, the disk store compaction had nothing to do in the `*_3.*` files, so they were left alone. The `*_4.*` files had garbage records, so the oplog from them was compacted into the new `*_5.*` files.

Example 6.8 Disk Store Compaction

```
bash-2.05$ ls -ltra backupDirectory
total 28
-rw-rw-r--   1 jpearson users          3 Apr  7 14:56 BACKUPds1_3.drf
-rw-rw-r--   1 jpearson users         25 Apr  7 14:56 BACKUPds1_3.crf
drwxrwxr-x   3 jpearson users       1024 Apr  7 15:02 ..
-rw-rw-r--   1 jpearson users       7085 Apr  7 15:06 BACKUPds1.if
-rw-rw-r--   1 jpearson users         18 Apr  7 15:07 BACKUPds1_4.drf
-rw-rw-r--   1 jpearson users       1070 Apr  7 15:07 BACKUPds1_4.crf
drwxrwxr-x   2 jpearson users        512 Apr  7 15:07 .

bash-2.05$ gemfire validate-disk-store ds1 backupDirectory
/root: entryCount=6
/partitioned_region entryCount=1 bucketCount=10
Disk store contains 12 compactable records.
Total number of region entries in this disk store is: 7

bash-2.05$ gemfire compact-disk-store ds1 backupDirectory
Offline compaction removed 12 records.
Total number of region entries in this disk store is: 7

bash-2.05$ ls -ltra backupDirectory
total 16
-rw-rw-r--   1 jpearson users          3 Apr  7 14:56 BACKUPds1_3.drf
-rw-rw-r--   1 jpearson users         25 Apr  7 14:56 BACKUPds1_3.crf
drwxrwxr-x   3 jpearson users       1024 Apr  7 15:02 ..
-rw-rw-r--   1 jpearson users          0 Apr  7 15:08 BACKUPds1_5.drf
-rw-rw-r--   1 jpearson users        638 Apr  7 15:08 BACKUPds1_5.crf
-rw-rw-r--   1 jpearson users       2788 Apr  7 15:08 BACKUPds1.if
drwxrwxr-x   2 jpearson users        512 Apr  7 15:09 .
bash-2.05$
```

Backing Up and Restoring a Disk Store

Backups and restores are done differently for online and offline distributed systems.

Online Backup

The GemFire backup creates a backup of disk stores for all members running in the distributed system when the backup command is invoked. The backup works by passing commands to the running system members. Each member with persistent data creates a backup of its own configuration and disk stores. The backup does not block any activities in the distributed system, but it does use resources.

> *Only use the* `gemfire backup` *command to create backup files from a running distributed system.*
>
> *Do not try to create backup files from a running system using file copy commands. You will get incomplete and unusable copies.*

1. You might want to compact your disk store before running the backup. See the `gemfire compact-all-disk-stores` command in *Online Compaction* on page 113.
2. Run the backup during a period of low activity in your system. The backup does not block system activities, but it uses file system resources on all hosts in your distributed system and can affect performance.
3. Configure each member's `cache.xml` with any files or directories you want backed up in addition to the standard files. The standard files are listed in *What the Online Backup Saves* on page 116.

   Each directory specified is copied recursively, with any disk stores found excluded from this user-specified backup. Example:

   ```
   <backup>./myExtraBackupStuff</backup>
   ```

   You cannot restore a configuration file back into a `jar` file. You can, however, back up a `jar` file and have it automatically restored along with everything else. To do this, add the file to your specifications. Example:

   ```
   <backup>myJarFile.jar</backup>
   ```

4. Back up to a directory that all members can access. Make sure the directory exists and has the proper permissions for your members to write to it and create subdirectories.

   The directory you specify for backup can be used multiple times. Each backup first creates a top level directory for the backup, under the directory you specify, identified to the minute.

   You can use one of two methods:

   - Use a single physical location, such as a network file server. Example:

     ```
     /export/fileServerDirectory/gemfireBackupLocation
     ```

   - Use a directory that is local to all host machines in the system. Example:

     ```
     ./gemfireBackupLocation
     ```

5. Make sure there is a `gemfire.properties` file for the distributed system in the directory where you run the `gemfire` command. The command backs up all disk stores in the specified distributed system.
6. Make sure all members with persistent data are running in the system. Offline members cannot back up their disk stores. The tool gives a message telling you about any members that are offline:

   ```
      The backup may be incomplete. The following disk stores are not
   online:
          DiskStore at hostc.gemstone.com /home/dsmith/dir3
   ```

7. Run the command, providing your backup directory location. Example:

```
gemfire backup /export/fileServerDirectory/gemfireBackupLocation
```

8. The tool reports on the success of the operation. If the operation is successful, you see a message like this:

```
Connecting to distributed system: locators=warsaw.gemstone.com[26340]
The following disk stores were backed up:
   DiskStore at hosta.gemstone.com /home/dsmith/dir1
   DiskStore at hostb.gemstone.com /home/dsmith/dir2
Backup successful.
```

   If the operation does not succeed at backing up all known members, you see a message like this:

```
Connecting to distributed system: locators=warsaw.gemstone.com[26357]
The following disk stores were backed up:
   DiskStore at hosta.gemstone.com /home/dsmith/dir1
   DiskStore at hostb.gemstone.com /home/dsmith/dir2
The backup may be incomplete. The following disk stores are not
online:
   DiskStore at hostc.gemstone.com /home/dsmith/dir3
```

   A member that fails to complete its backup is noted in this ending status message and leaves the file `INCOMPLETE_BACKUP` in its highest level backup directory. Offline members leave nothing, so you only have this message from the backup operation itself.

What the Online Backup Saves

For each member with persistent data, the backup includes:

1. Disk store files for all stores containing persistent region data
2. Any files or directories you have configured to be backed up in `cache.xml` `<backup>` elements.

   Example:

```
<backup>./systemConfig/gf.jar</backup>
<backup>/users/jpearson/gfSystemInfo/myCustomerConfig.doc</backup>
```

3. Configuration files from the member startup.

   3.1 `gemfire.properties`, with the properties the member was started with

   3.2 `cache.xml`, if used

   These files are not automatically restored, to avoid interfering with more recent configurations. In particular, if these are extracted from a master `.jar` file, copying the separate files into your working area could override the files in the `.jar`. If you want to back up and restore these files, add them as custom `<backup>` elements.

4. A restore script, written for the member's operating system, that copies the files back to their original locations. For example, in Windows, the file is `restore.bat` and in Linux, it is `restore.sh`.

Example 6.9 Backup Directory Structure and Contents

```
bash-2.05$ ls -R 2010-04-10-11-35/
2010-04-10-11-35/:
straw_14871_53406_34322  straw_14872_53410_34326

2010-04-10-11-35/straw_14871_53406_34322:
README.txt  config      diskstores  restore.sh

2010-04-10-11-35/straw_14871_53406_34322/config:
cache.xml           gemfire.properties

2010-04-10-11-35/straw_14871_53406_34322/diskstores:
ds1

2010-04-10-11-35/straw_14871_53406_34322/diskstores/ds1:
dir0  dir1

2010-04-10-11-35/straw_14871_53406_34322/diskstores/ds1/dir0:
BACKUPds1.if     BACKUPds1_2.drf  BACKUPds1_3.drf  BACKUPds1_4.drf
BACKUPds1_2.crf  BACKUPds1_3.crf  BACKUPds1_4.crf

      ... repeat for additional disk store directories

   ... repeat for additional disk stores

... repeat for additional members
```

Backup directory - date and time of backup: YYYY-MM-DD-hh-mm

Next level - one directory per member: machine _ member ID

One directory per disk store

Offline Members: Manual Catch-Up to an Online Backup

If you must have a member offline during an online backup, you can manually back up its disk stores. Do one of the following:

- Keep the member's backup and restore separated, doing offline manual backup and offline manual restore, if needed. See *Offline File Backup and Restore* on page 118.
- Bring this member's files into the online backup framework manually and create a restore script by hand, from a copy of another member's script:
  a. Duplicate the directory structure of a backed up member for this member.
  b. Rename directories as needed to reflect this member's particular backup, including disk store names.
  c. Clear out all files but the restore script.
  d. Copy in this member's files.
  e. Modify the restore script to work for this member.

Restoring an Online Backup

The restore script copies files back to their original locations. You can do this manually if you wish.

1. Restore your disk stores when your members are offline and the system is down.
2. Read the restore scripts to see where they will place the files and make sure the destination locations are ready. The restore scripts refuse to copy over files with the same names.
3. Run the restore scripts. Run each script on the host where the backup originated.

What GemFire Restores

The restore copies these back to their original location:

1. Disk store files for all stores containing persistent region data
2. Any files or directories you have configured to be backed up in the `cache.xml` `<backup>` elements.

Offline File Backup and Restore

With the system offline, you copy and restore your files using your file system commands.

Backup

To back up your offline system:

1. Consider compacting your disk stores before backing them up. See *Validating a Disk Store* on page 111 and *Compacting Disk Store Log Files* on page 112.
2. Copy all disk store files—and any other files you want to save—to your backup locations.

Restore

To restore a backup of an offline system:

1. Make sure the system is either down or not using the directories you will use for the restored files.
2. Reverse your backup file copy procedure, copying all the backed up files into the directories you want to use.
3. Make sure your members are configured to use the directories where you put the files.
4. Start the system members.

Keeping Your Offline Disk Store In Sync with Your Cache

These recommendations are aimed at optimizing disk store use and data loading at startup.

Changing Region Configuration

When your disk store is offline, you can keep the configuration for its regions up-to-date with your `cache.xml` and API settings. The disk store retains region capacity and load settings, including entry map settings (initial capacity, concurrency level, load factor), LRU eviction settings, and the statistics enabled boolean. If the configurations do not match at startup, the `cache.xml` and API override any disk store settings and the disk store is automatically updated to match. So you do not need to modify your disk store to keep your cache configuration and disk store synchronized, but you will save startup time and memory if you do.

Example:

```
gemfire modify-disk-store myDiskStoreName /firstDiskStoreDir
/secondDiskStoreDir /thirdDiskStoreDir -region=/partitioned_region
-initialCapacity=20
```

To list all modifiable settings and their current values for a region, run `modify-disk-store` with no actions specified.

Example:

```
gemfire modify-disk-store myDiskStoreName /firstDiskStoreDir
/secondDiskStoreDir /thirdDiskStoreDir -region=/partitioned_region
```

Taking a Region Out of Your Cache Configuration

This applies to the removal of regions while the disk store is offline. Regions you destroy through API calls are automatically removed from the disk store.

In your application development, when you discontinue use of a persistent region, remove it from the member's disk store as well.

Perform the following operations with caution. You are permanently removing data.

You can do this in one of two ways:

- Delete the entire set of disk store files. Your member will initialize with an empty set of files the next time you and start it.
- Selectively remove the discontinued region from the disk store.

  Example:

  ```
  gemfire modify-disk-store myDiskStoreName /firstDiskStoreDir
  /secondDiskStoreDir /thirdDiskStoreDir -region=/partitioned_region
  -remove
  ```

You might remove a region from your application if you decide to rename it or to split its data into two entirely different regions. Any significant data restructuring can cause you to retire some data regions.

To guard against unintended data loss, GemFire maintains the region in the disk store until you manually remove it. Regions in the disk stores that are not associated with any region in your application are still loaded into temporary regions in memory and kept there for the life of the member. The system has no way of detecting whether the cache region will be created by your API at some point, so it keeps the temporary region loaded and available.

Handling Missing Disk Stores

This section applies to disk stores that hold the latest copy of your data for at least one region.

Listing Missing Disk Stores

The `gemfire` list missing disk stores command lists all disk stores with most recent data that are being waited on by other members.

For replicated regions, this command only lists missing members that are preventing other members from starting up. For partitioned regions, this command also lists any offline data hosts, even when other data hosts for the region are online, because their offline status may be causing `PartitionOfflineException`s in cache operations or preventing the system from satisfying redundancy.

Example:

```
gemfire list-missing-disk-stores
Connecting to distributed system: mcast=/239.192.81.2:12348
DiskStore at straw.gemstone.com
/export/straw3/users/jpearson/testGemFire/hostB/DS1
```

Make sure this `gemfire` *call can find a* `gemfire.properties` *file for the system.*

The disk store directories listed for missing disk stores may not be the directories you have currently configured for the member. The list is retrieved from the other running members—the ones who are reporting the missing member. They have information from the last time the missing disk store was online. If you move your files and change the member's configuration, these directory locations will be stale.

Disk stores usually go missing because their member fails to start. The member can fail to start for a number of reasons, including:

- Disk store file corruption, see *Validating a Disk Store* on page 111
- Incorrect distributed system configuration for the member
- Network partitioning
- Drive failure

Revoking a Missing Disk Store

This section applies to disk stores for which both of the following are true:

- Disk stores that have the most recent copy of data for one or more regions or region buckets.
- Disk stores that are unrecoverable, such as when you have deleted them, or their files are corrupted or on a disk that has had a catastrophic failure.

When you cannot bring the latest persisted copy online, use the revoke command to tell the other members to stop waiting for it. Once the store is revoked, the system finds the remaining most recent copy of data and uses that.

Once revoked, a disk store cannot be reintroduced into the system.

`Use gemfire list missing disk stores to properly identify the disk store you need to revoke. The revoke command takes the host and directory in input, as listed by that command.`

Example:

```
gemfire list-missing-disk-stores
Connecting to distributed system: mcast=/239.192.81.2:12348
DiskStore at straw.gemstone.com
/export/straw3/users/jpearson/testGemFire/hostB/DS1
gemfire revoke-missing-disk-store straw.gemstone.com
/export/straw3/users/jpearson/testGemFire/hostB/DS1
Connecting to distributed system: mcast=/239.192.81.2:12348
revocation was successful ...
```

Make sure these `gemfire` *calls can find a* `gemfire.properties` *file for the system.*

Chapter 7

Administering the Distributed System

This chapter describes some operations required to administer a GemFire Enterprise® distributed system. It covers topics like starting and stopping your systems, configuring and managing your cache server members, and managing memory resources for partitioned and non-partitioned data regions.

In this chapter:

7.1 Starting and Stopping the Distributed System

This section explains how to start up and shut down the GemFire processes in a distributed system. The processes you need to start and stop include applications, cache servers, and locators.

You can start cache servers and locators from the command line.

You can use the `com.gemstone.gemfire.admin` API to manage GemFire processes, like locators, agent, and cacheserver.

For persistent regions, see the information on how startup and shutdown are performed in *Running a System with Disk Stores* on page 106.

Startup

Create startup scripts for your processes, including the locators and cache servers, to ensure that the right sequences are followed consistently.

Start server systems first, then client systems.

For each system, use this start order. All of these processes are optional:

1. Locators. For details about starting locators, see *Using Locators For Peer and Client/Server Discovery* on page 62.
2. Server systems. When you start members that persist data to disk, start all members close together if you can, so that they can negotiate to determine which member has the most up to data copy of each region.
   2.1 Cache Servers. For details about starting cache servers, see *Configuring and Running the GemFire Cache Server* on page 126.
   2.2 Java applications. For startup considerations useful to application developers, see *GemFire Members and Member Caches* on page 69 of the *GemFire Enterprise Developer's Guide*.
3. Client systems.

Startup After Losing Data on Disk

This pertains to catastrophic loss of disk files. If you lose data stored on disk, your next startup may hang, waiting for the lost disk stores to come back online. When you start your system, use the `gemfire` command to see if any disk stores are missing and, if needed, revoke missing disk stores so your system startup can complete. See *Handling Missing Disk Stores* on page 120.

Shutdown

If any members persist data to disk, have them running when you shut down, if possible.

The `gemfire` command line tool stops your running system in an orderly manner and makes the next startup as efficient as possible:

```
gemfire shut-down-all
```

This call requires a `gemfire.properties` *file for the system you are shutting down.*

Option for System Member Shutdown Behavior

The `DISCONNECT_WAIT` property sets the maximum amount of time each individual step in the shutdown process can take before being forced to end. Each outstanding operation at the time of shutdown is given this grace period, so the total length of time that the cache member takes to shut down depends on the number of operations as well as the `DISCONNECT_WAIT` setting.

During this shutdown process, GemFire produces messages such as:

```
Disconnect listener still running:
```

The `DISCONNECT_WAIT` default is 10000 milliseconds. To configure it, set this system property on the Java command line used for member startup:

```
-DDistributionManager.DISCONNECT_WAIT=milliseconds
```

7.2 Configuring and Running the GemFire Cache Server

The cache server is a GemFire process that runs as a long-lived, configurable member of a distributed system. The cache server is used primarily for hosting long-lived data regions and for running standard GemFire processes such as the `CacheServer` in a client/server configuration. For more information on client/server caching, see *Client/Server Architecture and Configuration Basics* on page 201 of the *GemFire Enterprise Developer's Guide*.

This section describes cache server configuration and the `bin/cacheserver` command-line utility. The cache server can also be configured and managed through the API, as discussed in *Developing System Administration Tools* on page 479 of the *GemFire Enterprise Developer's Guide* and in the online Java API documentation.

Cache Server Configuration and Log Files

The cache server uses a working directory for its configuration files and log files. Typically, you provide a `gemfire.properties` file and a `cache.xml` file in the cache server's working directory. These are the defaults and configuration options:

- For the distributed system connection, the cache server looks for a `gemfire.properties` file located according to the search rules outlined in *GemFire Configuration Files* on page 42. If none is present, the default system configuration is used. For information on this file, see *System Properties in the gemfire.properties File* on page 48.
- For cache configuration, the cache server uses the declarative `cache.xml` file. The file specification comes either from the `cache-xml-file` (page 49) gemfire property or from a `cache-xml-file` distributed system attribute input at the command line. The GemFire cache server can only be programmed through application plug-ins. For details on application plug-ins, see *Controlling Data Flow With Application Plug-ins* on page 193 of the *GemFire Enterprise Developer's Guide*.
- For logging output, the cache server defaults to `cacheserver.log` in the working directory. You can specify a different log file in a distributed system attribute input at the command line.

The cacheserver Command-Line Utility

The `$GEMFIRE/bin/cacheserver` command-line tool allows you to start and stop the cache server process. This section lists the command-line options and gives an example startup sequence.

Starting the Cache Server

The cache server startup syntax is:

```
cacheserver start [-J<vmarg>]* [<attName>=<attValue>]* [-dir=
<workingdir>] [-classpath=<classpath>] [-rebalance] [-server-port=
<server-port>] [-server-bind-address=<server-bind-address>]
```

Table 7.1 cacheserver Command-Line Options

Option	Description
`-J=vmarg`	A VM option passed to the `cacheserver` VM. Any number of `-J` options may be used. As an example, `-J-Xmx1024m` sets the VM heap to 1GB.
`attName=attValue`	Distributed system property name/value pair, for example `cache-xml-file=/serverConfig/cache.xml`. Any number of these may be specified. See *System Properties in the gemfire.properties File* on page 48.
`-dir=workingDir`	The directory where the `cacheserver` writes its status file. If not otherwise specified, this is also where the server looks for the `gemfire.properties` and `cache.xml`. Default: current working directory
`-classpath=classpath`	Location of user classes. This path is appended to the `CLASSPATH` environment variable.
`-rebalance`	Causes the server to kick off a partitioned region rebalance on startup. See Rebalancing Partitioned Regions *on page 187 of the* GemFire Enterprise Developer's Guide.
`-server-port=server-port`	Overrides the `port` (page 209) setting in the `<cache-server>` element of the `cache.xml` file. Use this to start multiple cacheservers using the same configuration files but different ports.
`-server-bind-address=server-bind-address`	Overrides the `bind-address` (page 207) setting in the `<cache-server>` element of the `cache.xml` file. Use this to start multiple cacheservers using the same configuration files but different addresses.

This sample startup sequence starts two cache servers, using a single XML file for cache configuration and different incoming client connection ports:

Example 7.1 Sample cacheserver Start Sequence for Two Servers (bash Version)

```
> cd CS1WorkingDir

> cacheserver start mcast-port=10338 cache-xml-file=/serverConfig/cache.xml
-server-port=40404

> cd CS2WorkingDir

> cacheserver start mcast-port=10338 cache-xml-file=/serverConfig/cache.xml
-server-port=40405
```

This example does the same thing using a `gemfire.properties` file to set the multicast port and `cache.xml` file:

Example 7.2 gemfire.properties File

```
#contents of D:\gfeserver\gemfire.properties
#Tue May 09 17:53:54 PDT 2006
mcast-port=10338
cache-xml-file=D:\gfeserver\cacheCS.xml
```

Example 7.3 Two cacheservers Using the Same Properties File and Setting Unique Ports

```
C:\> cd CS1

C:\CS1> cacheserver start -J-DgemfirePropertyFile=
D:\gfeserver\gemfire.properties -server-port=40404

C:\CS1> cd \CS2

C:\CS2> cacheserver start -J-DgemfirePropertyFile=
D:\gfeserver\gemfire.properties -server-port=40405
```

Stopping the Cache Server

To stop the cache server, enter the following command:

```
> cacheserver stop [-dir=workingDir]
```

where *`workingDir`* is the working directory for the running cache server.

Checking the Cache Server's Status

To obtain status information for a cache server, enter the following command:

```
> cacheserver status [-dir=workingDir]
```

where *`workingDir`* is the working directory for the running cache server.

7.3 Handling Network Outages

This section explains how to keep your distributed system from splitting into two separate running systems when members lose the ability to see each other, as shown in Figure 7.1 on page 130. The typical cause of this problem is a network failure. When a network failure, or partitioning, occurs, the problem could result in data inconsistencies or a forced disconnect. The solution for this problem is to stop one of the two subgroups from continuing to operate independently.

Handling network outages is based on the participation of a lead member and a group management coordinator. The coordinator is a member that manages the entry and exit of other members of the distributed system. With network partition detection, the coordinator is always a GemFire locator. The lead member is always the oldest member of the distributed system that does not have a locator running in the same VM and is not using the administrator interface. The two situations that cause GemFire to declare a network partitioning condition are:

- If both a locator and the lead member abnormally leave the distributed system within a configurable period of time, the caches of members who are unable to see the locator and the lead member are immediately closed and disconnected.

  Only abnormal loss of the locator and lead member cause GemFire to declare a network partition. If a lead member's distributed system is disconnected normally, GemFire automatically elects a new one and continues to operate. If a locator is disconnected, a secondary locator takes over.

- If no locator can be contacted by a member, the member closes its cache and disconnects from the distributed system. Since only locators can make membership decisions, a member that cannot contact any locator cannot know if it is isolated from the lead member.

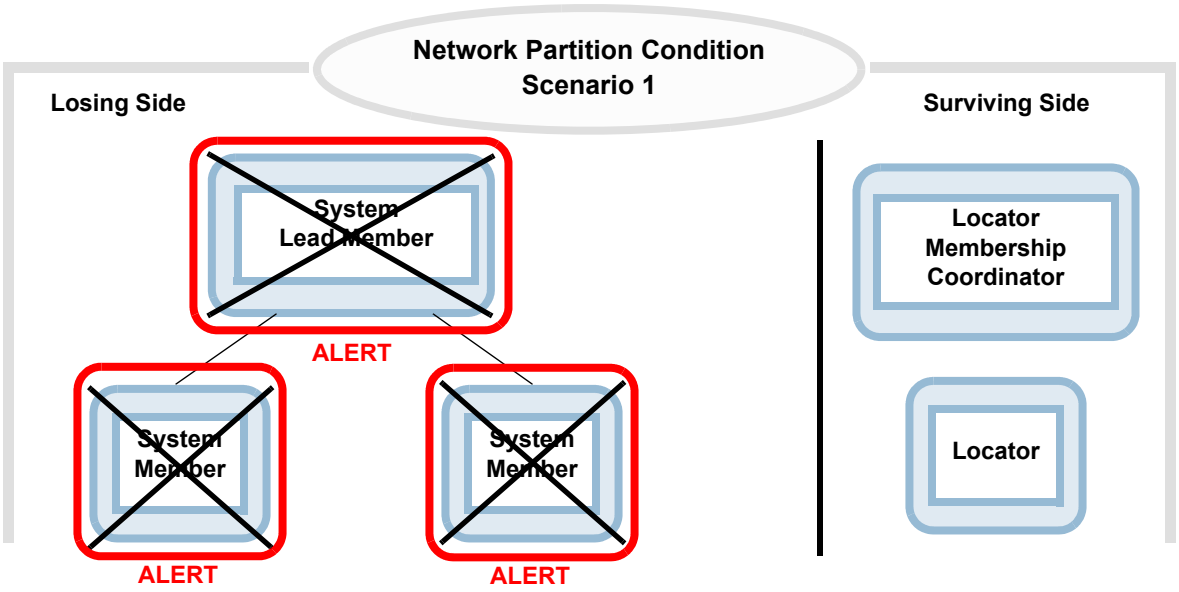
Network partitioning handling allows only one subgroup to form and survive. The distributed systems are disconnected and the caches of other subgroups are closed. When a shutdown occurs, alerts are generated through the GemFire logging system, explaining to administrators what action, if any, to take. Refer to *Network Partitioning, Slow Response, and Member Removal Alerts* on page 211 for information about alerts.

Constraints on Deployment

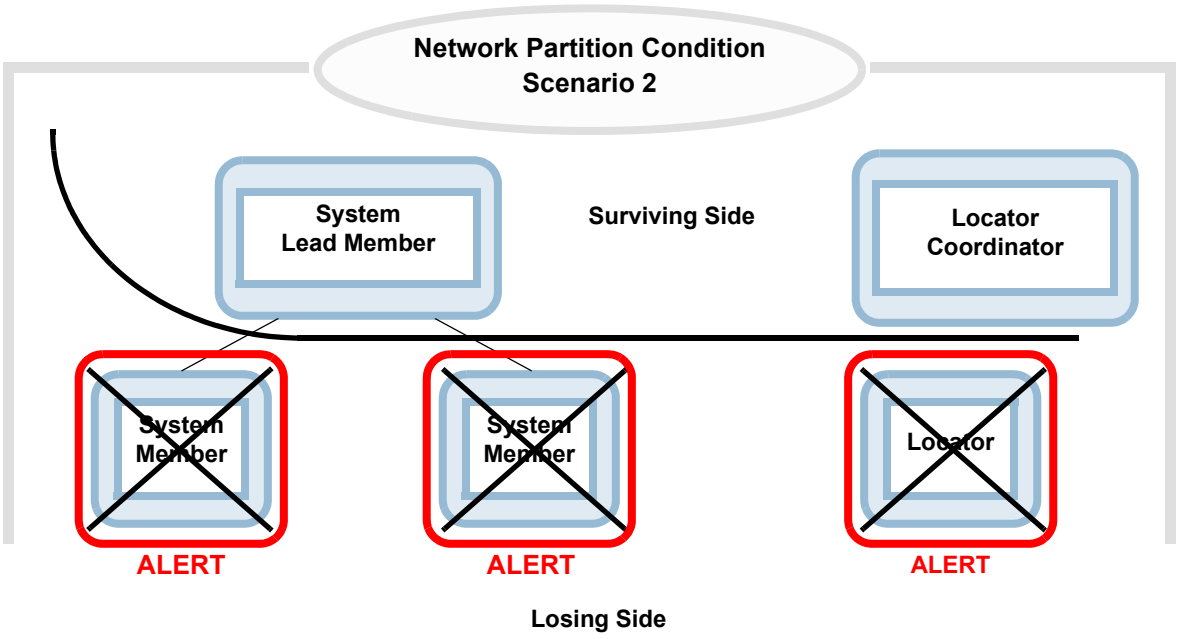
The network partition detection system in GemFire imposes constraints on how a system is deployed.

- Network partition detection does not protect against content skew if the `DISTRIBUTED-NO-ACK` region distribution scope is used. Use `DISTRIBUTED-ACK` or `GLOBAL` scope, or partitioned regions.
- Locators must be used for system member discovery, and they must not be colocated with processes using the GemFire cache. Part of the network partition detection algorithms are built into the locator processes and depend on their presence, so it is required that locators are not involved in cache activities.
- At least two locators should be used. If only one locator is running, and it abnormally terminates, any member with network partition detection enabled would close its cache and disconnect its distributed system.
- Locators must not run on the same machine as other members of the distributed system, but they can be run on machines that have only client caches.

Figure 7.1 Network Failure—Network Partition Configurations



1. Locators see the lead member leave and remain active.
2. Members lose sight of all locators and are closed because there is no one to make membership decisions. Each member generates an alert. They cannot tell whether the lead member is alive and do not know who the current lead member may be.



1. Failure detection uses member-timeout to remove lost members.
2. On the losing side, each member sees the loss of the coordinator and lead member and shuts down, causing each member and the locator to generate an alert.
3. On the surviving side, members still see the lead member and locator, and remain active.

Losing Side

If a lead member is lost, each member must determine if network partitioning has occurred. The loss must be from failure, not the normal shutdown process. This is true for all lead member and isolated member failure processes.

The member records the loss of the lead member and examines its history to see if another process that is eligible to become coordinator has also left in the correlation window. If so, network partitioning may have occurred and the member shuts down. If another coordinator hasn't left, the active coordinator immediately designates a new lead member from the remaining members. If there are no eligible members available, there is no lead member until an eligible member joins the distributed system. If a process eligible to be the coordinator is lost subsequent to the lead member leaving, and the losses both occur within the correlation window, network partitioning may have occurred and the members in the losing partition are disconnected.

Any member that has enabled network partition detection and is not hosting a locator is eligible to be designated as the lead member by the group coordinator. When a coordinator is also present, the presence of the lead member determines which group of members survive when there is a network partition.

A member that detects a network partition disconnects its distributed systems and closes its cache. If a network partition caused the loss, the processes in the other partition (eligible coordinator, lead member, and processes still able to see them) continue to run, electing a new coordinator if necessary.

An important side effect of this is that loss of only two processes, a coordinator and the current lead member, cause all other processes in the distributed system to disconnect. Any clients to the system are unaffected. Any locators should not be on the same machine as an application that could be selected as the lead member.

In Scenario 2 for the previous figure, all peer VMs receive a `RegionDestroyedException` with `Operation: FORCED_DISCONNECT`. If a `CacheListener` is installed, the `afterRegionDestroy` callback is invoked with the `RegionDestroyedEvent` that is logged for the losing side VMs, as shown in the following example.

Example 7.4 afterRegionDestroy Callback Invoked by RegionDestroyedEvent

```
[info 2008/05/01 11:14:51.853 PDT <CloserThread> tid=0x4a] Invoked
splitBrain.SBListener: afterRegionDestroy in client1
   whereIWasRegistered: 14291
   event.isReinitializing(): false
   event.getDistributedMember(): thor(14291):40440/34132
   event.getCallbackArgument(): null
   event.getRegion(): /TestRegion
   event.isDistributed(): false
   event.isExpiration(): false
   event.isOriginRemote(): false
   Operation: FORCED_DISCONNECT
   Operation.isDistributed(): false
   Operation.isExpiration(): false
```

On the losing side, the peer VM process IDs are 14291 (lead member) and 14296, and the locator VM is 14289.

Peers still actively performing operations on the cache may see `ShutdownExceptions` or `CacheClosedExceptions` with `Caused by: ForcedDisconnectException`.

Losing side members get `RegionDestroyedException(Operation: FORCED_DISCONNECT)`, and may see `Shutdown` or `CacheClosed` exceptions with `Caused by: ForcedDisconnectException` if losing side members are executing region or entry operations at the time of the network partition. The surviving side administrator VMs see `memberCrashed` events if they have a `SystemMembershipListener` installed.

If a member using the `Admin` interface on the losing side has an `AlertListener` configured, its `alert` callback is invoked for all system logging above the configured alert level, as shown in the following example.

Example 7.5 alert Callback Invoked for System Logging Above the Configured Alert Level

```
[info 2008/05/01 11:14:42.126 PDT <Pooled Message Processor2> tid=0x41]
Invoked splitBrain.SBAlertListener in client with vmID 1, pid 14289
   alert.getConnectionName(): gemfire1_thor_14291
   alert.getDate(): Thu May 01 11:14:42 PDT 2008
   alert.getLevel(): WARNING
   alert.getMessage(): unable to send message to
   biscuit.gemstone.com/10.80.10.70:50972 (128 bytes);Operation was not
   permitted by datagram socket.

alert.getSourceId(): TimeScheduler.Thread tid=0x1d
alert.getSystemMember(): gemfire1_thor_14291
```

Surviving Side

If the locator VM on the surviving side has an `AlertListener` configured, its `alert` callback is invoked for messages above the configured `AdminDistributedSystem.getAlertLevel`, as shown in the following example.

On the surviving side, the peer VM is 7435, the locator (coordinator) is 7444, and the locator VM is 7430.

Example 7.6 alert Callback Invoked for Messages Above the Configured Alert Level

```
[info 2008/05/01 11:14:55.807 PDT <Pooled Message Processor2> tid=0x40]
Invoked splitBrain.SBAlertListener in client with vmID 2, pid 7430
   alert.getConnectionName(): gemfire4_biscuit_7438
   alert.getDate(): Thu May 01 11:14:55 PDT 2008
   alert.getLevel(): WARNING
alert.getMessage(): 15 sec have elapsed while waiting for replies:
<ReplyProcessor21 2688 waiting for 2 replies from [thor(14291):40440/34132,
thor(14296):40442/55944]> on biscuit(7438):50975/57267 whose current
membership list is: [[biscuit(7435):50978/50626, thor(14291):40440/34132,
thor(14296):40442/55944, biscuit(7438):50975/57267]]
   alert.getSourceId(): vm_6_thr_10_client2_biscuit_7438 tid=0x48
   alert.getSystemMember(): gemfire4_biscuit_7438
```

If a member using the `Admin` interface on the surviving side has a `SystemMembershipListener` configured, it processes `memberCrashedEvents` for the peer VMs on the losing side, as shown in the following example.

Example 7.7 Processing MemberCrashedEvents for Peer VMs on the Losing Side

```
[info 2008/05/01 11:15:22.742 PDT <DM-MemberEventInvoker> tid=0x1b] Invoked
splitBrain.SBSystemMembershipListener: memberCrashed in admin2
   event.getDistributedMember(): thor(14291):40440/34132
   event.getMemberId(): thor(14291):40440/34132
[info 2008/05/01 11:15:27.790 PDT <DM-MemberEventInvoker> tid=0x1b] Invoked
splitBrain.SBSystemMembershipListener: memberCrashed in admin2
   event.getDistributedMember(): thor(14296):40442/55944
   event.getMemberId(): thor(14296):40442/55944
```

Enabling Network Partition Detection

Network partition detection is enabled by setting the `enable-network-partition-detection` property in `gemfire.properties` to `true`. This must be done in all locators and in any other process that you wish to be sensitive to network partitioning. Processes that do not have network partition detection enabled are not eligible to be the lead member, so their failure will not trigger declaration of a network partition. When enabled, membership coordination is forced to be available only in locators.

All locators must have the same setting for `enable-network-partition-detection`. If they don't, the system throws a `GemFireConfigException` at startup.

For network partition detection, locators must be used instead of multicast discovery.

When `enable-network-partition-detection` is set to `true`, avoid using `DISTRIBUTED_NO_ACK` scope. When no acknowledgements are required on cache operations, the server performing the operation may perform many operations before detecting a network problem. When acknowledgements are required, the server will immediately detect network problems.

Failure Detection

Network partitioning has a failure detection protocol that is not subject to hanging when network interface cards or machines fail.

Failure detection works by detecting missing datagram heartbeats from the peer left in the membership view, followed by attempting to form a TCP/IP connection, and then sending a `VERIFY_SUSPECT` datagram message to all other processes. Those processes all quickly send several `ARE_YOU_DEAD` datagram messages to the suspect process. If the process does not answer one of these messages with an `I_AM_NOT_DEAD` response, the process is kicked out of membership. It is sent a message to disconnect the distributed system and close the cache.

Failure detection processing is also initiated on a member if the `ack-wait-threshold` setting defined in `gemfire.properties` elapses before receiving a response to a message, or if a TCP/IP connection cannot be made to the member for peer-to-peer (P2P) messaging.

Isolated Members

When a member is isolated from all locators, it is unable to receive membership view changes. It will not even know that the current coordinator has left if there are no other members to take over that role. When in this condition, a member that was using a locator with network partition detection enabled must shut itself down because it cannot detect whether the lead member has left along with the locators.

The only way a member knows that the locators are absent is when it periodically tries to register with the locators and is unable to reach any of them. This registration takes place approximately once per minute in the default GemFire configuration. When network partition detection is enabled, this is shortened to 3 times the `member-timeout` (page 54) interval, which is the period of time required for UDP failure detection to notice a failed member.

New Membership and Loss of Members

The group management system in GemFire has a group coordinator that is responsible for allowing new members into the system and removing old members. It does this by sending a view to each participating process. When the coordinator itself leaves the view (or becomes suspect), other members make a decision on which member will be the new coordinator. Normally, the new coordinator is selected from the full membership set. When network partition prevention is enabled, the role of coordinator is limited to members hosting a locator service.

A network partition-enabled coordinator selects a lead member from the list of non-admin members in the view that have network partition detection enabled. This selection is sent out with the view to all members, so they immediately know who this lead member is. The lead member is used to determine which members will be disconnected if there is a network partition.

The coordinator and processes eligible to become coordinator keep a short-term history of its election as coordinator and any departures from the membership view. These are time-stamped, and each departure in the history notes whether the member was a coordinator, lead member, or a process eligible to become coordinator. This history is used to detect and correlate related departures.

The `departure-correlation-window` (page 50) setting in `gemfire.properties` controls the period of time during which abnormal loss of a locator and a lead member would cause declaration of a network partition. The default setting is 1800 seconds.

Client Cache read-timeout

If the system has clients, the client `read-timeout` period defined in `cache.xml` should be set to be at least three times the server `member-timeout` setting. The client-side `read-timeout` period determines how long a client will wait for a server response.

7.4 Managing Memory

This section covers topics related to memory use in your GemFire installation.

Memory Overhead Introduced by the Cache API

For each entry added to a region, the GemFire Enterprise cache API consumes a certain amount of Java memory to store and manage the data. This overhead is required even when an entry is overflowed or persisted to disk.

The Java cache overhead introduced by a region, using a 32-bit VM, can be approximated as listed below. Actual memory use varies based on a number of factors, including the JVM you are using and the platform you are running on. For 64-bit VMs, the usage will usually be larger than with 32-bit VMs. As much as 80% more memory may be required for 64-bit VMs.

- Add 87 bytes of overhead for each region entry. This value may vary because memory consumption for object headers and object references varies for 64-bit VMs, different VM implementations, and different JDK versions.
- For partitioned regions, add 16 bytes of VM memory per entry.
- If you persist the region or overflow it to disk, add 40 bytes per entry.
- If statistics are enabled, add 16 bytes per entry.
- When using the LRU (least recently used) eviction controller, add 16 bytes for each entry.
- For each optional user attribute, add 52 bytes of VM memory per entry (plus the space used by the user attribute object).
- For global regions, a distributed lock token may be needed for each entry. Each token uses 90 bytes of VM memory.
- For entry expiration, add 147 bytes of VM memory per entry.
- For indexes used in querying, the overhead varies greatly depending on the type of data you are storing and the type of index you create. You can roughly estimate the overhead for some types of indexes as follows:
  - If the index has a single value per region entry for the `indexedExpression`, then the index introduces at most 243 bytes per region entry. An example of this type of index is:

    ```
    fromClause="/portfolios", indexedExpression="id"
    ```

    The maximum of 243 bytes per region entry assumes that each entry has a unique value for the indexed expression. The overhead is reduced if the entries do not have unique index values.
  - If each region entry has more than one value for `indexedExpression`, but no two region entries have the same value for the `indexedExpression`, then the index introduces at most $236C + 75$ bytes per region entry, where C is the average number of values per region entry for the `indexedExpression`.

For suggestions on handling the trade-offs between memory overhead and system performance, see *Garbage Collection* on page 142.

Calculating the Size of Your Data

Objects in GemFire Enterprise are serialized for storage into partitioned regions and for all distribution activities, including overflow and persistence to disk. For optimum performance, GemFire tries to reduce the number of times an object is serialized and deserialized. Because of this, your objects may be stored in serialized form or non-serialized form in the cache. To do capacity planning for your data, therefore, use the larger of the serialized and deserialized sizes. If your objects classes are `DataSerializable`, the non-serialized form will generally be the larger of the two. For information on data serialization, see *Data Serialization* on page 470 of the *GemFire Enterprise Developer's Guide*.

Overhead of Application Objects

This section describes the overhead of your application objects in the cache. These are the estimated values for 32-bit VMs and 64-bit VMs. Sizes may vary slightly between JVMs and platforms.

- **Object header**—12 bytes on 32-bit VMs. (The object header is actually only 8 bytes, but an extra 4 bytes padding is added if the total object size is not a multiple of 8, as is true roughly half the time.) 20 bytes for each object header on 64-bit VMs. Make sure to count the key as well as the value, and to count every object if the key and/or value is a composite object.
- **Field**—On 32-bit VMs, 8 bytes for fields of type double or long, 4 bytes per field for all others. On 64-bit VMs, the size is the same as for 32-bit except for fields that are references to objects, which take 8 bytes.

7.5 Managing Resources for Partitioned Regions

Partitioned regions are typically used to manage large quantities of data distributed across many machines. When administering partitioned regions, common operations include expanding the number of members that host the region to keep up with data volume and taking down individual members to do planned maintenance.

For information on partitioned regions, see *Partitioned Regions* on page 177 of the *GemFire Enterprise Developer's Guide*.

Adding an Extra Partitioned Region Data Host at Run Time

If you begin seeing `PartitionedRegionStorageException`s or log messages notifying that "`Partitioned Region <name> has exceeded local maximum memory configuration`", you may not have enough resources to support the data traffic. If the exception text ends with "`Consider starting another member`," then you must evaluate whether to start more applications or cache servers to provide more capacity. This exception can also indicate other issues; for more factors to consider, see `PartitionedRegionStorageException` on page 205.

Lack of capacity is particularly an issue when a partitioned region is configured for redundancy. Make sure you start enough members to provide the level of reliability you require.

When a partitioned region is configured for high availability, keeping copies of existing data is a higher priority than storing new data. If the region has been very short of space, a new data host can fill up immediately as the missing copies are created. You may have to start more than one additional member to make room for new data.

By default, the new member makes copies of existing data to bring the system up to the configured redundancy level. If you want GemFire to move existing data to the new member, you must configure GemFire to rebalance the system.

Removing a Partitioned Region Data Host

You can shut down a member, such as for planned maintenance, without stopping the other members involved in the partitioned region. If your partitioned region is configured for high availability or persistence, you do not lose any data. For members with only data on disk, data access operations will return a `PartitionOfflineException` while the member is down.

Chapter

8

Monitoring and Tuning the Distributed System

This chapter covers methods for monitoring and tuning the performance of your GemFire Enterprise® system. You can monitor your system during runtime and analyze archived statistics. You can control various performance aspects, including garbage collection, message delivery for slow consumers, and socket use.

In this chapter:

8.1 Monitoring Tools

GemFire Enterprise provides a number of tools for monitoring and tuning your GemFire system. System monitoring is available through the study of archived logging and statistics information. System managing and monitoring is available through the GemFire Enterprise Monitor version 2 (GFMon) program, for which information is available in the GemFire Enterprise Monitor User's Manual. System tuning can be accomplished through declarative configuration files. For information on the configuration files, see *GemFire Configuration Files* on page 42.

Monitoring and tuning capabilities are also provided through the programming APIs. See *Developing System Administration Tools* on page 479 of the *GemFire Enterprise Developer's Guide* for details on the administration API and *Monitoring and Tuning Your Applications* on page 463 of the *GemFire Enterprise Developer's Guide* for more on application monitoring and tuning.

You can use command-line tools to monitor your GemFire system. The command-line tools monitor one locator or cache server at a time. You can retrieve information about the process, such as its ID, and its current status, such as running or stopped. To check the status of a locator, use this command:

```
gemfire status-locator [-dir=locatorDir]
```

To check the status of a cache server, use this command:

```
cacheserver status [-dir=workingDir]
```

For details about the `gemfire` command, see Appendix A. For the `cacheserver` command, see *Configuring and Running the GemFire Cache Server* on page 126.

The GemFire Enterprise installation includes standard statistics for caching and distribution activities and provides an API for creating application-defined statistics. For detailed information on the primary statistics for distribution and high-level caching activities, refer to Appendix B, *System Statistics*, on page 235. The API and the `Region` and `Entry` statistics available from the cache are discussed in *Statistics* on page 464 of the *GemFire Enterprise Developer's Guide*.

8.2 System Member Performance

This section describes configuration parameters you can modify to improve system member performance. Some of these controls relate to the GemFire statistics described in Appendix B, *System Statistics*, on page 235.

Distributed System Member

These properties apply to any cache server or application that connects to the distributed system. The properties can be specified in the `gemfire.properties` file. See *Configuring GemFire System Properties* on page 45.

- **statistic-sampling-enabled**—Turning statistics sampling off saves resources, but it also takes away potentially valuable information for ongoing system tuning and unexpected system problems. If LRU eviction is configured, then statistics sampling must be on.
- **statistic-sample-rate**—Lowering the sample rate for statistics reduces system resource use while still providing some statistics for system tuning and failure analysis.
- **log-level**—As with the statistic sample rate, lowering this reduces system resource consumption. See *Logging Options* on page 187.

JVM Memory Settings

The properties in this section affect how the Java VM uses memory. For the Java application, these properties are set by adding parameters to the `java` invocation. For the cache server, they are added to the command-line parameters for the `cacheserver` startup script.

- **VM heap size**—Your VM may require more memory than is allocated by default. For example, you may need to increase heap size for an application VM that stores a lot of data. You can set a maximum size and an initial size, so if you know you will be using the maximum (or close to it) for the life of the VM, you can speed memory allocation time by setting the initial size to the maximum. This sets both the maximum and initial memory sizes to 1024 megabytes for a Java application:

  ```
  -Xmx1024m -Xms1024m
  ```

  The properties are passed to the cache server on the command line:

  ```
  cacheserver start -J-Xmx1024m -J-Xms1024m
  ```

- **MaxDirectMemorySize**—The VM has a kind of memory called direct memory, which is distinct from normal VM heap memory, that can run out. You can increase the direct buffer memory either by increasing the maximum heap size (see previous VM Heap Size), which increases both the maximum heap and the maximum direct memory, or by only increasing the maximum direct memory using `-XX:MaxDirectMemorySize`.

  The following parameter added to the Java application startup increases the maximum direct memory size to 256 megabytes:

  ```
  -XX:MaxDirectMemorySize=256M
  ```

  The same effect for the cache server:

  ```
  cacheserver start -J-XX:MaxDirectMemorySize=256M
  ```

Garbage Collection

Garbage collection, while necessary, introduces latency into your system by consuming resources that would otherwise be available to your application. If you are experiencing unacceptably high latencies in application processing, you might be able to improve performance by modifying your VM's garbage collection behavior.

> *Garbage collection tuning options depend on the Java virtual machine you are using. Suggestions given here apply to the Sun HotSpot VM. If you use a different JVM, check with your vendor to see if these or comparable options are available to you.*
>
> *Modifications to garbage collection sometimes produce unexpected results. Always test your system before and after making changes to verify that the system's performance has improved.*

The two options suggested here are likely to expedite garbage collecting activities by introducing parallelism and by focusing on the data that is most likely to be ready for cleanup. The first parameter causes the garbage collector to run concurrent to your application processes. The second parameter causes it to run multiple, parallel threads for the "young generation" garbage collection (that is, garbage collection performed on the most recent objects in memory—where the greatest benefits are expected):

```
-XX:+UseConcMarkSweepGC -XX:+UseParNewGC
```

For applications, if you are using remote method invocation (RMI) Java APIs, you might also be able to reduce latency by disabling explicit calls to the garbage collector. The RMI internals automatically invoke garbage collection every sixty seconds to ensure that objects introduced by RMI activities are cleaned up. Your VM may be able to handle these additional garbage collection needs. If so, your application may run faster with explicit garbage collection disabled. You can try adding the following command-line parameter to your application invocation and test to see if your garbage collector is able to keep up with demand:

```
-XX:+DisableExplicitGC
```

Connection Thread Settings

If a large number of peer processes are started concurrently, the distributed system connect time can be improved by setting the `p2p.HANDSHAKE_POOL_SIZE` system property value to the expected number of members. This property controls the number of threads that can be used to establish new TCP/IP connections between peer caches. The threads are discarded if they are idle for 60 seconds.

The default value for `p2p.HANDSHAKE_POOL_SIZE` is `4`. This command-line specification sets the number of threads to 100:

```
-Dp2p.HANDSHAKE_POOL_SIZE=100
```

8.3 Slow Receivers with TCP/IP

This section discusses options for preventing situations that can cause slow receivers of data distributions and it provides methods for handling slow receivers.

> *The slow receiver options control only peer-to-peer communication using TCP/IP. This discussion does not apply to client/server or multi-site communication, or to communication using the UDP unicast or multicast protocols.*

Preventing Problems That Can Cause Slow Receivers

This section discusses some of the potential causes of slow receivers in peer-to-peer communication that should be identified and eliminated during system integration. Work with your network administrator to eliminate any problems you identify.

Slowing is more likely to occur when applications run many threads, send large messages (due to large entry values), or have a mix of region configurations. The problem can also arise from message delivery retries caused by intermittent connection problems.

Host Resources

Make sure that the machines that run GemFire members have enough CPU available to them. Do not run any other heavyweight processes on the same machine.

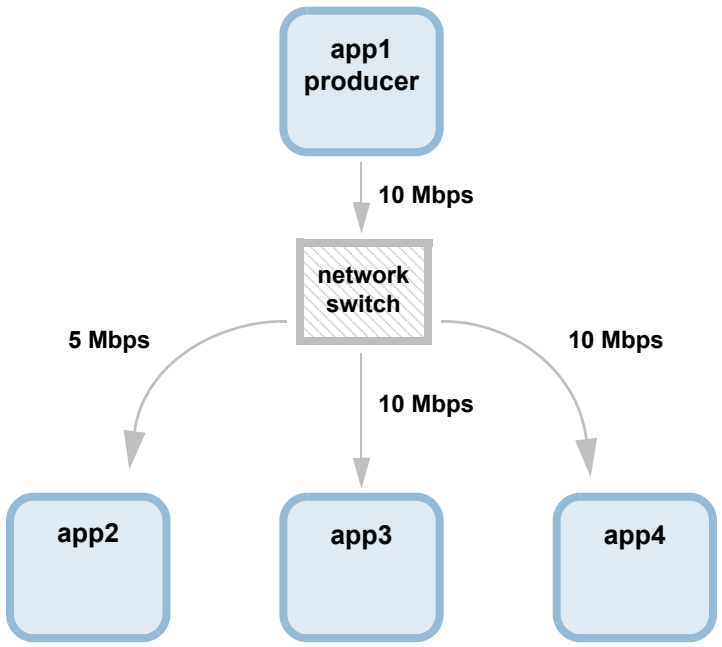
The machines that host GemFire application and cache server processes should have comparable computing power and memory capacity. Otherwise, members on the less powerful machines tend to have trouble keeping up with the rest of the group.

Network Capacity

Eliminate congested areas on the network by rebalancing the traffic load. Work with your network administrator to identify and eliminate traffic bottlenecks, whether caused by the architecture of the distributed GemFire system or by contention between the GemFire traffic and other traffic on your network. Consider whether more subnets are needed to separate the GemFire administrative traffic from GemFire data transport and to separate all the GemFire traffic from the rest of your network load.

The network connections between hosts need to have equal bandwidth. If not, you can end up with a configuration like the multicast example in the following figure, which creates conflicts among the members. For example, if app1 sends out data at 7Mbps, app3 and app4 would be fine, but app2 would miss some data. In that case, app2 contacts app1 on the TCP channel and sends a log message that it's dropping data.

Figure 8.1 Unbalanced Network Capacity Problem



Plan for Growth

Upgrade the infrastructure to the level required for acceptable performance. Analyze the expected GemFire traffic in comparison to the network's capacity. Build in extra capacity for growth and high-traffic spikes. Similarly, evaluate whether the machines that host GemFire application and cache server processes can handle the expected load.

Managing Slow Receivers

This section discusses options for handling slow receivers.

If the receiver fails to receive a message, the sender continues to attempt to deliver the message as long as the receiving member is still in the distributed system. During the retry cycle, GemFire throws warnings that include this string:

```
will reattempt
```

The warnings are followed by an informational message when the delivery finally succeeds.

For distributed regions, the *scope* of a region determines whether distribution acknowledgements and distributed synchronization are required. For details, see `scope` on page 116 of the *GemFire Enterprise Developer's Guide*. Partitioned regions ignore the `scope` attribute, but for the purposes of this discussion you should think of them as having an implicit `distributed-ack` scope.

By default, distribution between system members is performed synchronously. With synchronous communication, when one member is slow to receive, it can cause its producers to slow down as well. This, of course, can lead to general performance problems in the distributed system.

If you are experiencing slow performance and are sending large objects (multiple megabytes), before implementing these slow receiver options make sure your socket buffer sizes are appropriate for the size of the objects you distribute. The socket buffer size is set using `socket-buffer-size` (page 56) in the `gemfire.properties` file.

Managing Slow distributed-no-ack Receivers

You can configure your consumer members so their messages are queued separately when they are slow to respond. The queueing happens in the producer members when the producers detect slow receipt and allows the producers to keep sending to other consumers at a normal rate. Any member that receives data distribution can be configured as described in this section.

The specifications for handling slow receipt primarily affect how your members manage distribution for regions with `distributed-no-ack` scope, where distribution is asynchronous, but the specifications can affect other distributed scopes as well. If no regions have `distributed-no-ack` scope, the mechanism is unlikely to kick in at all. When slow receipt handling does kick in, however, it affects all distribution between the producer and that consumer, regardless of scope.

> *These slow receiver options are disabled in systems using SSL. For information on SSL, see Configuring SSL on page 91.*

Each consumer member determines how its own slow behavior is to be handled by its producers. The settings are specified as distributed system connection properties. This section describes the settings and lists the associated properties. For configuration information, see *System Properties in the gemfire.properties File* on page 48.

- **async-distribution-timeout**—The distribution timeout specifies how long producers are to wait for the consumer to respond to synchronous messaging before switching to asynchronous messaging with that consumer. When a producer switches to asynchronous messaging, it creates a queue for that consumer's messages and a separate thread to handle the communication. When the queue empties, the producer automatically switches back to synchronous communication with the consumer.

  These settings affect how long your producer's cache operations might block. The sum of the timeouts for all consumers is the longest time your producer might block on a cache operation.

- **async-queue-timeout**—The queue timeout sets a limit on the length of time the asynchronous messaging queue can exist without a successful distribution to the slow receiver. When the timeout

is reached, the producer asks the consumer to leave the distributed system, as described in *Forcing the Slow Receiver to Disconnect* on page 146.

- **async-max-queue-size**—The maximum queue size limits the amount of memory the asynchronous messaging queue can consume. When the maximum is reached, the producer asks the consumer to leave the distributed system.

The statistics pertaining to slow receivers are provided in the distribution statistics. See *VM Statistics* on page 278.

Configuring Async Queue Conflation

When the scope is `distributed-no-ack` scope, you can configure the producer to conflate entry update messages in its queues, which may further speed communication. By default, `distributed-no-ack` entry update messages are not conflated. The configuration is set in the producer at the region level. For more information, see `enable-async-conflation` on page 106 of the *GemFire Enterprise Developer's Guide*.

Forcing the Slow Receiver to Disconnect

If either of the queue timeout or maximum queue size limits is reached, the producer sends the consumer a high-priority message (on a different TCP connection than the connection used for cache messaging) telling it to disconnect from the distributed system. This prevents growing memory consumption by the other processes that are queuing changes for the slow receiver while they wait for that receiver to catch up. It also allows the slow member to start fresh, possibly clearing up the issues that were causing it to run slowly.

When a producer gives up on a slow receiver, it logs one of these types of warnings:

```
Blocked for time ms which is longer than the max of asyncQueueTimeout ms
so asking slow receiver slow_receiver_ID to disconnect.

Queued bytes bytes exceeds max of asyncMaxQueueSize so asking slow
receiver slow_receiver_ID to disconnect.
```

When a process disconnects after receiving a request to do so by a producer, it logs a warning message of this type:

```
Disconnect forced by producer because we were too slow.
```

These messages only appear in your logs if logging is enabled and the log level is set to a level that includes `warning` (which it does by default). Logging is specified in the `gemfire.properties` file. For more information on setting your logging properties, see Chapter 10, *GemFire System Logging*, on page 185.

If your consumer is unable to receive even high priority messages, only the producer's warnings will appear in the logs. If you see only producer warnings, you can restart the consumer process. Otherwise, the GemFire failure detection code will eventually cause the member to leave the distributed system on its own.

Use Cases

This section describes the main use cases for the slow receiver specifications.

- **Message bursts**—With message bursts, the socket buffer can overflow and cause the producer to block. To keep from blocking, first make sure your socket buffer is large enough to handle a normal number of messages (see `socket-buffer-size` (page 56)*)*, then set the async distribution timeout to 1. With this very low distribution timeout, when your socket buffer does fill up, the producer quickly switches to async queueing. Use the distribution statistics,

`asyncQueueTimeoutExceeded` and `asyncQueueSizeExceeded`, to make sure your queue settings are high enough to avoid forcing unwanted disconnects during message bursts.

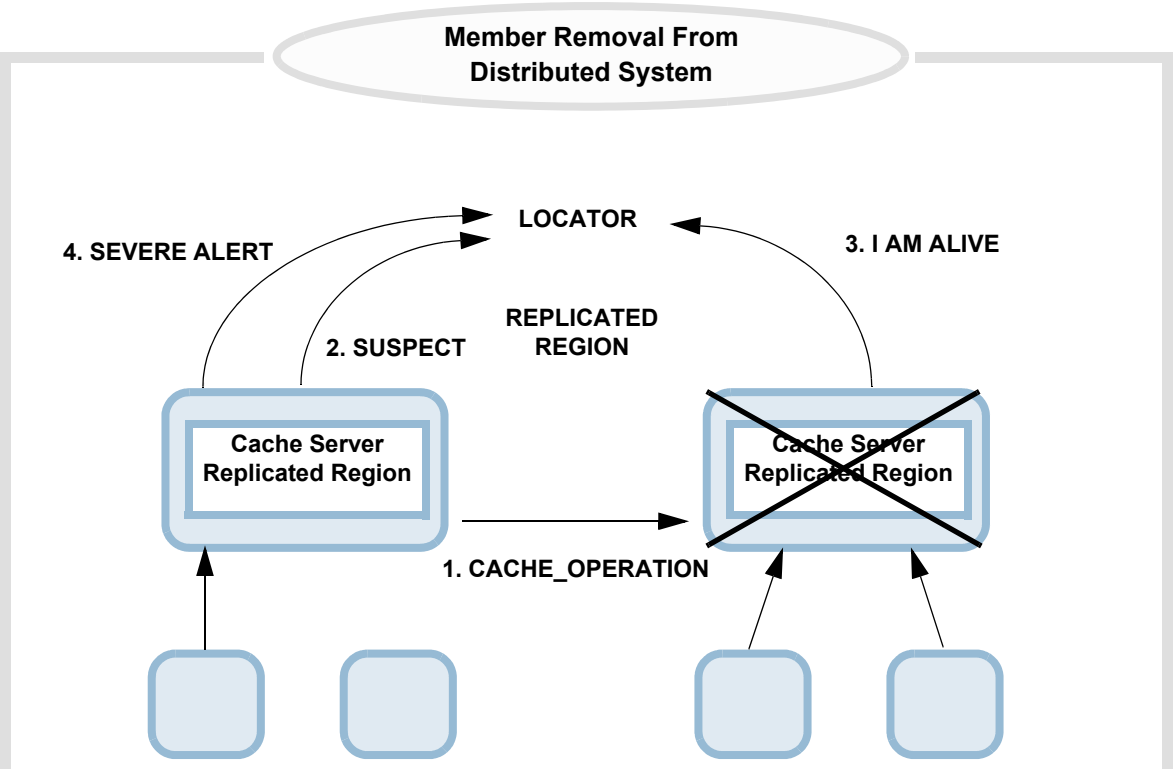
- **Unhealthy or dead members**—When members are dead or very unhealthy, they may not be able to communicate with other distributed system members. The slow receiver specifications allow you to force crippled members to disconnect, freeing up resources and possibly allowing the members to restart fresh. To configure for this, set the distribution timeout high (one minute), and set the queue timeout low. This is the best way to avoid queueing for momentary slowness, while still quickly telling very unhealthy members to leave the distributed system.
- **Combination message bursts and unhealthy members**—To configure for both of the above situations, set the distribution timeout low and the queue timeout high, as for the message bursts scenario.

Managing Slow distributed-ack Receivers

When using a distribution scope other than `distributed-no-ack`, alerts are issued for slow receivers. A member that isn't responding to messages may be sick, slow, or missing. Sick or slow members are detected in message transmission and reply-wait processing code, triggering a warning alert first. If a member still isn't responding, a severe warning alert is issued, indicating that the member may be disconnected from the distributed system. This alert sequence is enabled by setting the `ack-wait-threshold` and the `ack-severe-alert-threshold` to some number of seconds.

When `ack-severe-alert-threshold` is set, regions are configured to use ether `distributed-ack` or `global` scope, or use the `partition` data policy. GemFire will wait for a total of `ack-wait-threshold` seconds for a response to a cache operation, then it logs a warning alert (see *Warning Notifications Before Removal* on page 215). After waiting an additional `ack-severe-alert-threshold` seconds after the first threshold is reached, the system also informs the failure detection mechanism that the receiver is suspect and may be disconnected, as shown in the following figure.

Figure 8.2 Events Leading to Member Severe Alert



The events occur in this order:

1. CACHE_OPERATION - transmission of cache operation is initiated.
2. SUSPECT - identified as a suspect by `ack-wait-threshold`, which is the maximum time to wait for an acknowledge before initiating failure detection.
3. I AM ALIVE - notification to the system in response to failure detection queries, if the process is still alive.

   A new membership view is sent to all members if the suspect process fails to answer with I AM ALIVE.
4. SEVERE ALERT- the result of `ack-severe-wait-threshold` elapsing without receiving a reply.

When a member fails suspect processing, its cache is closed and its `CacheListeners` are notified with the `afterRegionDestroyed` notification. The `RegionEvent` passed with this notification has a `CACHE_CLOSED` operation and a `FORCED_DISCONNECT` operation, as shown in the `FORCED_DISCONNECT` example.

Example 8.1 FORCED_DISCONNECT Operation

```
public static final Operation FORCED_DISCONNECT
= new Operation("FORCED_DISCONNECT",
       true, // isLocal
       true, // isRegion
       OP_TYPE_DESTROY,
       OP_DETAILS_NONE
       );
```

A cache closes due to being expelled from the distributed system by other members. Typically, this happens when a member becomes unresponsive and does not respond to heartbeat requests within the `member-timeout` period, or when `ack-severe-alert-threshold` has expired without a response from the member.

This is marked as a region operation.

Other members see the normal membership notifications for the departing member. For instance, `RegionMembershipListeners` receive the `afterRemoteRegionCrashed` notification, and `SystemMembershipListeners` receive the `memberCrashed` notification.

8.4 Tuning to Reduce Slow distributed-ack Messages

In systems with `distributed-ack` regions, a sudden large number of `distributed-no-ack` operations can cause `distributed-ack` operations to take a long time to complete. The `distributed-no-ack` operations can come from anywhere. They may be updates to `distributed-no-ack` regions or they may be other `distributed-no-ack` operations, like destroys, performed on any region in the cache, including the `distributed-ack` regions.

The main reasons why a large number of `distributed-no-ack` messages may delay `distributed-ack` operations are:

- For any single socket connection, all operations are executed serially. If there are any other operations buffered for transmission when a `distributed-ack` is sent, the `distributed-ack` operation must wait to get to the front of the line before being transmitted. Of course, the operation's calling process is also left waiting.
- The `distributed-no-ack` messages are buffered by their threads before transmission. If many messages are buffered and then sent to the socket at once, the line for transmission might be very long.

You can take these steps to reduce the impact of this problem:

1. If you're using TCP, check whether you have socket conservation enabled for your members. It is configured by setting the GemFire property `conserve-sockets` (page 50) to `true`. If enabled, each application's threads will share sockets unless you override the setting at the thread level. Work with your application programmers to see whether you might disable sharing entirely or at least for the threads that perform `distributed-ack` operations. These include operations on `distributed-ack` regions and also `netSearch`es performed on regions of any distributed scope. If you give each thread that performs `distributed-ack` operations its own socket, you effectively let it scoot to the front of the line ahead of the `distributed-no-ack` operations that are being performed by other threads. The thread-level override is done by calling the `DistributedSystem setThreadsSocketPolicy(false)`. For more information, see the online Java documentation or *Controlling Socket Use* on page 471 of the *GemFire Enterprise Developer's Guide*.
2. Reduce your buffer sizes to slow down the `distributed-no-ack` operations:
   - If you're using UDP (you either have multicast enabled regions or have set `disable-tcp` to `true` in `gemfire.properties`), consider reducing the `byteAllowance` of `mcast-flow-control` (page 53) to something smaller than the default of 3.5 megabytes.
   - If you're using TCP/IP, reduce the `socket-buffer-size` (page 56) in `gemfire.properties`.

   These changes slow down the threads performing `distributed-no-ack` operations and allow the thread doing the `distributed-ack` operations to be sent in a more timely manner.

8.5 Tuning Socket Communication

GemFire Enterprise processes communicate using TCP/IP and UDP unicast and multicast protocols. In all cases, communication uses sockets that you can tune to optimize performance. This section discusses your options for general socket communication tuning. The sections that following provide information specific to tuning UDP unicast and multicast communication.

The adjustments you make to tune your GemFire communication may run up against operating system limits. If this happens, check with your system administrator about adjusting the operating system settings.

All of the settings discussed here are listed as `gemfire.properties` and `cache.xml` settings. They can also be configured through the API and some can be configured at the command line. For information in this, see the online Java documentation and *Configuring GemFire System Properties* on page 45.

Setting Socket Buffer Sizes

When determining buffer size settings, you are trying to strike a balance between communication needs and other processing. Larger socket buffers allow your members to distribute data and events more quickly, but they also take memory away from other things. If you store very large data objects in your cache, finding the right sizing for your buffers while leaving enough memory for the cached data can become critical to system performance.

Ideally, you should have buffers large enough for the distribution of any single data object so you don't get message fragmentation, which lowers performance. Your buffers should be at least as large as your largest stored objects and their keys plus some overhead for message headers. The overhead varies depending on the who is sending and receiving, but 100 bytes should be sufficient. You can also look at the statistics for the communication between your processes to see how many bytes are being sent and received.

If you see performance problems and logging messages indicating blocked writers, increasing your buffer sizes may help.

This table lists the settings for the various member relationships and protocols, and tells where to set them.

Table 8.1 Socket Buffer Size Configuration Properties

Protocol / Area affected	Configuration location	Property name
TCP/IP		
Peer-to-peer send/receive	`gemfire.properties`	`socket-buffer-size`
Client send/receive	`cache.xml <pool>`	`socket-buffer-size`
Server send/receive	`cache.xml <CacheServer>`	`socket-buffer-size`
Gateway hub send/receive	`cache.xml <gateway-hub>`	`socket-buffer-size`
Gateway send/receive	`cache.xml <gateway>`	`socket-buffer-size`
UDP multicast		
Peer-to-peer send	`gemfire.properties`	`mcast-send-buffer-size`
Peer-to-peer receive	`gemfire.properties`	`mcast-recv-buffer-size`
UDP unicast		
Peer-to-peer send	`gemfire.properties`	`udp-send-buffer-size`
Peer-to-peer receive	`gemfire.properties`	`udp-recv-buffer-size`

TCP/IP Buffer Sizes

If possible, your TCP/IP buffer size settings should match across your GemFire installation. At a minimum, follow the guidelines listed here.

Peer-to-Peer

The `socket-buffer-size` setting in `gemfire.properties` should be the same throughout your distributed system.

Client/Server

The client's pool socket buffer size should match the setting for the servers the pool uses, as in these example `cache.xml` snippets:

Example 8.2 Client Socket Buffer Size cache.xml Configuration

```
<pool>name="PoolA" server-group="dataSetA" socket-buffer-size="42000"...
```

Example 8.3 Server Socket Buffer Size cache.xml Configuration

```
<cache-server port="40404" socket-buffer-size="42000">
   <group>dataSetA</group>
</cache-server>
```

Multisite (WAN)

In a multi-site installation using gateways, if the link between sites is not tuned for optimum throughput, it could cause messages to back up in the cache queues. If a receiving queue overflows because of inadequate buffer sizes, it will become out of sync with the sender and the receiver will be unaware of the condition.

The gateway's `<gateway> socket-buffer-size` attribute should match the gateway hub's `<gateway-hub> socket-buffer-size` attribute for the hubs the gateway connects to, as in these example `cache.xml` snippets:

Example 8.4 Gateway Socket Buffer Size cache.xml Configuration

```
<gateway-hub id="EU" port="33333">
   <gateway id="US" socket-buffer-size="42000">
      <gateway-endpoint id="US-1" host="USHost" port="11111"/>
      <gateway-queue overflow-directory="overflow"
         maximum-queue-memory="50" batch-size="100"
         batch-time-interval="1000"/>
   </gateway>
</gateway-hub>
```

Example 8.5 Gateway Hub Socket Buffer Size cache.xml Configuration

```
<gateway-hub id="US" port="11111" socket-buffer-size="42000">
    <gateway id="EU">
      <gateway-endpoint id="EU-1" host="EUHost" port="33333"/>
      <gateway-queue overflow-directory="overflow"
          maximum-queue-memory="50" batch-size="100"
          batch-time-interval="1000"/>
    </gateway>
  </gateway-hub>
```

UDP Multicast and Unicast Buffer Sizes

With UDP communication, one receiver can have many senders sending to it at once. To accommodate all of the transmissions, the receiving buffer should be larger than the sum of the sending buffers. If you have a system with at most five members running at any time, in which all members update their data regions, you would set the receiving buffer to at least five times the size of the sending buffer. If you have a system with producer and consumer members, where only two producer members ever run at once, the receiving buffer sizes should be set at over two times the sending buffer sizes, as shown in this example:

Example 8.6 UDP Socket Buffer Settings in gemfire.properties for a Two-Producer System

```
mcast-send-buffer-size=42000
mcast-recv-buffer-size=90000
udp-send-buffer-size=42000
udp-recv-buffer-size=90000
```

Operating System Limits

Your operating system sets limits on the buffer sizes it allows. If you request a size larger than the allowed, you may get warnings or exceptions about the setting during startup. These are two examples of the type of message you may see:

```
[warning 2008/06/24 16:32:20.286 PDT CacheRunner <main> tid=0x1]
requested multicast send buffer size of 9999999 but got 262144: see
system administration guide for how to adjust your OS

Exception in thread "main" java.lang.IllegalArgumentException: Could not
set "socket-buffer-size" to "99262144" because its value can not be
greater than "20000000".
```

If you think you are requesting more space for your buffer sizes than your system allows, check with your system administrator about adjusting the operating system limits.

Ephemeral TCP Port Limits

If you are repeatedly receiving the following exception:

```
java.net.BindException: Address already in use: connect
```

and if your system is experiencing a high degree of network activity, such as numerous short-lived client connections, this could be related to a limit on the number of ephemeral TCP ports. By default, Windows' ephemeral ports are within the range 1024-4999, inclusive. While this issue could occur with other operating systems, typically, it is only seen with Windows due to a low default limit.

Perform this procedure to increase the limit:

1. Open the Windows **Registry Editor**.
2. Navigate to the following key:

   `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameter`
3. From the **Edit** menu, click **New**, and then add the following registry entry:

   Value Name: `MaxUserPort`

   Value Type: `DWORD`

   Value data: `36863`
4. Exit the **Registry Editor**, and then restart the computer.

This affects all versions of the Windows operating system.

Note for UDP on Unix Systems

Unix systems have a default maximum socket buffer size for receiving UDP multicast and unicast transmissions that is lower than the default settings for `mcast-recv-buffer-size` and `udp-recv-buffer-size`. To achieve high-volume multicast messaging, you should increase the maximum Unix buffer size to at least one megabyte.

Making Sure You Have Enough Sockets

The number of sockets your applications have available to them is governed by operating system limits. Sockets use file descriptors and the operating system's view of your application's socket use is expressed in terms of file descriptors. There are two limits, one on the maximum descriptors available to a single application and the other on the total number of descriptors available in the system. If you get error messages telling you that you have too many files open, you might be hitting the operating system limits with your use of sockets. Your system administrator might be able to increase the system limits so that you have more available. You can also tune your members to use fewer sockets for their outgoing connections. This section discusses socket use in GemFire and ways to limit socket consumption in your GemFire members.

Socket Sharing

You can configure socket sharing for peer-to-peer and client-to-server connections.

Peer-to-Peer

You can configure whether your members share sockets both at the application level and at the thread level. To enable sharing at the application level, set the `gemfire.properties` `conserve-sockets` (page 50) to `true`. Developers can override this setting at the thread level using the `DistributedSystem` API method `setThreadsSocketPolicy`. You might want to enable socket sharing at the application level and then have threads that do a lot of cache work take sole ownership of their sockets. Make sure to program these threads to release their sockets as soon as possible using the `releaseThreadsSockets` method, rather than waiting for a timeout or thread death.

Client

You can configure whether your clients share their socket connections to servers with the `pool` setting `thread-local-connections` (see `thread-local-connections` on page 212 of the *GemFire Enterprise Developer's Guide*). There is no thread override for this setting. All threads either have their own socket or they all share.

Socket Lease Time

You can force the release of an idle socket connection for peer-to-peer and client-to-server connections.

Peer-to-Peer

For peer-to-peer threads that do not share sockets, you can use the `socket-lease-time` (page 56) to make sure that no socket sits idle for too long. When a socket that belongs to an individual thread remains unused for this time period, the system automatically returns it to the pool. The next time the thread needs a socket, it retrieves one from the pool. Socket lease times can be placed on peer connection, with the, on client connections

Client

For client connections, you can affect the same lease-time behavior by setting the `pool` `idle-timeout` (see `idle-timeout` on page 210 of the *GemFire Enterprise Developer's Guide*).

Calculating Connection Requirements

Each type of member has its own connection requirements. Clients need connections to their servers, peers need connections to peers, and so on. Many members have compound roles. Use these guidelines to figure each member's socket needs and to calculate the combined needs of members that run on a single host system.

A VM's socket use is governed by a number of factors, including:

- How many peer members it connects to
- How many threads it has that update the cache and whether the threads share sockets.
- Whether it is a server, a client, or a gateway hub.
- How many connections come in from other processes.

The socket requirements described here are worst-case. Generally, it is not practical to calculate exact socket use for your applications. Socket use varies depending a number of factors including how many members are running, what their threads are doing, and whether threads share sockets.

To calculate any member's socket requirements, add up the requirements for every category that applies to the member. For example, a cache server running in a distributed system with clients connected to it has both peer-to-peer and server socket requirements.

Peer-to-Peer

As a basis, every member of a distributed system maintains two outgoing and two incoming connections to every peer. If threads share sockets, these fixed sockets are the sockets they share.

For every thread that does not share sockets, additional sockets, one in and one out, are added for each peer. This affects not only the member's socket count, but the socket count for every member the member thread connects to.

Table 8.2 Peer Socket Requirements Per VM

Socket description	Number used
Membership failure detection	2
Listener for incoming peer connections (server P2P)	1
Shared sockets (2 in and 2 out) Threads that share sockets use these.	4 * (M-1)
This member's thread-owned sockets (1 in and 1 out for each thread, for each peer member).	(T * 2) * (M-1)
Other member's thread-owned sockets that connect to this member (1 in and 1 out for each). Note that this might include server threads if any of the other members are servers (see *Server*).	Summation over (M-1) other members of (T*2)
M is the total number of members in the distributed system. T is the number of threads in a member that own their own sockets and do not share.	

The threads servicing client requests add to the total count of thread-owned sockets both for this member connecting to its peers and for peers that connect to this member.
See Server Socket Requirements Per VM on page 157.

Server

Servers use one connection for each incoming client connection. By default, each connection is serviced by a server thread. These threads that service client requests communicate with the rest of the server distributed system to satisfy the requests and distributed update operations. Each of these threads uses its own thread-owned sockets for peer-to-peer communication. So this adds to the server's group of thread-owned sockets.

The thread and connection count in the server may be limited by server configuration settings. These are `max-connections` and `max-threads` settings in the `<cache-server>` element of the `cache.xml`. These settings limit the number of connections the server accepts and the maximum number of threads that can service client requests. Both of these limit your servers overall connection requirements:

- When the connection limit is reached, the server refuses additional connections. This limits the number of connections the server uses for clients.
- When the thread limit is reached, threads start servicing multiple connections. This does not limit the number of client connections, but does limit the number of peer connections required to service client requests. Each server thread used for clients uses its own sockets, so it requires 2 connections to each of the server's peers. The `max-threads` setting puts a cap on the number of this type of peer connection that your server needs.

The server uses one socket for each incoming client pool connection. If client subscriptions are used, the server creates an additional connection to each client that enables subscription.

Table 8.3 Server Socket Requirements Per VM

Socket description	Number used
Listener for incoming client connections	1
Client pool connections to server	number of pool connections to this server
Threads servicing client requests (the lesser of the client pool connection count and the server's `max-threads` setting). These connections are to the server's peers.	(2 * number of threads in a server that service client pool connections) * (M-1) These threads do not share sockets.
Subscription connections	2 * number of client subscription connections to this server
M is the total number of members in the server's distributed system.	

With client/server installations, the number of client connections to any single server is undetermined, but GemFire's server load balancing and conditioning keeps the connections fairly evenly distributed among servers.

Servers are peers in their own distributed system and have the additional socket requirements noted in *Peer-to-Peer* on page 152.

Client

Client connection requirements are compounded by how many pools they use. The use varies according to runtime client connection needs, but will usually have maximum and minimum settings. Look for the `<pool>` element in the `cache.xml` for the configuration properties.

Table 8.4 Client Socket Requirements Per VM

Socket description	Number used
Pool connection	summation over the client pools of `max-connections`
Subscription connections	2 * summation over the client pools of `subscription-enabled`

If your client acts as a peer in its own distributed system, it has the additional socket requirements noted in *Peer-to-Peer* on page 152.

Multisite

Gateway-hubs use one socket to listen for incoming connections from remote gateways. For each incoming, the hub opens one connection. In addition, each gateway has one outgoing connection to a remote hub.

Table 8.5 Multisite Socket Requirements Per VM

Socket description	Number used
Listener for incoming connections	number of gateway-hubs defined for the member
Incoming connection	summation over the gateway-hubs of the number of remote gateways configured to connect to the hub
Outgoing connection	summation over the gateway-hubs of the number of gateways defined for the hub

Servers are peers in their own distributed system and have the additional socket requirements noted in *Peer-to-Peer* on page 152.

TCP/IP Peer-to-Peer Handshake Timeouts

Connection handshake timeouts for TCP/IP connections may be alleviated by increasing the connection handshake timeout interval with the system property `p2p.handshakeTimeoutMs`. The default setting is `59000` milliseconds.

This sets the handshake timeout to `75000` milliseconds for a Java application:

```
-Dp2p.handshakeTimeoutMs=75000
```

The properties are passed to the cache server on the command line:

```
cacheserver start -J-Dp2p.handshakeTimeoutMs=75000
```

8.6 Tuning UDP Communication

This section describes configuration adjustments to improve multicast and unicast UDP performance of peer-to-peer communication in your GemFire system. See also the general communication tuning and multicast-specific tuning covered in *Tuning Socket Communication* on page 151 and *Tuning Multicast Communication* on page 161.

You can tune your GemFire UDP messaging to maximize throughput. There are two main tuning goals: to use the largest reasonable datagram packet sizes and to reduce retransmission rates. These reduce messaging overhead and overall traffic on your network while still getting your data where it needs to go. GemFire also provides statistics to help you decide when to change your UDP messaging settings.

UDP Datagram Size

You can change the UDP datagram size with the GemFire property udp-fragment-size (page 58). This is the maximum packet size for transmission over UDP unicast or multicast sockets. When possible, smaller messages are combined into batches up to the size of this setting.

Most operating systems set a maximum transmission size of 64k for UDP datagrams, so this setting should be kept under 60k to allow for communication headers. Setting the fragment size too high can result in extra network traffic if your network is subject to packet loss, as more data must be resent for each retransmission. If many UDP retransmissions appear in `DistributionStats`, you maybe achieve better throughput by lowering the fragment size.

UDP Flow Control

UDP protocols typically have a flow control protocol built into them to keep processes from being overrun by incoming no-ack messages. The GemFire UDP flow control protocol is a credit based system in which the sender has a maximum number of bytes it can send before getting its byte credit count replenished, or recharged, by its receivers. While its byte credits are too low, the sender waits. The receivers do their best to anticipate the sender's recharge requirements and provide recharges before they are needed. If the senders credits run too low, it explicitly requests a recharge from its receivers.

This flow control protocol, which is used for all multicast and unicast no-ack messaging, is configured using a three-part GemFire property mcast-flow-control (page 53). This property is composed of:

- `byteAllowance`—This determines how many bytes (also referred to as credits) can be sent before receiving a recharge from the receiving processes.
- `rechargeThreshold`—This sets a lower limit on the ratio of the sender's remaining credit to its `byteAllowance`. When the ratio goes below this limit, the receiver automatically sends a recharge. This reduces recharge request messaging from the sender and helps keep the sender from blocking while waiting for recharges.
- `rechargeBlockMs`—This tells the sender how long to wait while needing a recharge before explicitly requesting one.

In a well-tuned system, where consumers of cache events are keeping up with producers, the `byteAllowance` can be set high to limit flow-of-control messaging and pauses. VM bloat or frequent message retransmissions are an indication that cache events from producers are overrunning consumers.

UDP Retransmission Statistics

GemFire stores retransmission statistics for its senders and receivers. You can use these statistics to help determine whether your flow control and fragment size settings are appropriate for your system.

The retransmission rates are stored in the `DistributionStats` `ucastRetransmits` and `mcastRetransmits`. For multicast, there is also a receiver-side statistic `mcastRetransmitRequests` that can be used to see which processes aren't keeping up and are requesting retransmissions. There is no comparable way to tell which receivers are having trouble receiving unicast UDP messages.

8.7 Tuning Multicast Communication

This section describes configuration adjustments to improve the UDP multicast performance of peer-to-peer communication in your GemFire system. See also the general communication tuning and UDP tuning covered in *Tuning Socket Communication* on page 151 and *Tuning UDP Communication* on page 159.

Provisioning Bandwidth for Multicast

Multicast installations require much more planning and configuration than TCP installations. By choosing IP multicast, you gain scalability but lose the administrative convenience of TCP. When you install an application that runs over TCP, the network is almost always set up for TCP and other applications are already using it. When you install an application to run over IP multicast it may be the first multicast application on the network.

Multicast is very dependent on the environment in which it runs. Its operation is affected by the network hardware, the network software, the machines, which GemFire processes run on which machines, and whether there are any competing applications. You could find that your site has connectivity in TCP but not in multicast because some switches and network cards do not support multicast. Your network could have latent problems that you would never see otherwise. To successfully implement a distributed GemFire system using multicast requires the cooperation of both system and network administrators.

Bounded Operation Over Multicast

Group rate control is required for GemFire systems to maintain cache coherence. If your application delivers the same data to a group of members, your system tuning effort needs to focus on the slow receivers.

If some of your members have trouble keeping up with the incoming data, the other members in the group may be impacted. At best, slow receivers cause the producer to use buffering, adding latency for the slow receiver and perhaps for all of them. In the worst case, throughput for the group can stop entirely while the producer's CPU, memory and network bandwidth are dedicated to serving the slow receivers.

To address this issue, you can implement a *bounded operation* policy, which sets boundaries for the producer's operation. The appropriate rate limits are determined through tuning and testing to allow the fastest operation possible while minimizing data loss and latency in the group of consumers. This policy is suited to applications such as financial market data, where high throughput, reliable delivery and network stability are required. With the boundaries set correctly, your producer's traffic cannot cause a network outage.

Multicast protocols typically have a flow control protocol built into them to keep processes from being overrun. The GemFire flow control protocol uses the `mcast-flow-control` property to set producer and consumer boundaries for multicast flow operations. The property provides these three configuration settings:

`byteAllowance`	Number of bytes that can be sent without a recharge.
`rechargeThreshold`	Tells consumers how low the producer's initial to remaining allowance ratio should be before sending a recharge.
`rechargeBlockMs`	Tells the producer how long to wait for a recharge before requesting one.

For details on these settings, see `mcast-flow-control` on page 53.

Testing Multicast Speed Limits

TCP automatically adjusts its speed to the capability of the processes using it and enforces bandwidth sharing so that every process gets a turn. With multicast, you have to explicitly set those limits yourself. Without the proper configuration, multicast delivers its traffic as fast as possible, overrunning the ability of consumers to process the data and locking out other processes that are waiting for the bandwidth. You can tune your multicast and unicast behavior using `mcast-flow-control` in `gemfire.properties`.

Using Iperf

Iperf is an open-source TCP/UDP performance tool that you can use to find your site's maximum rate for data distribution over multicast. Iperf can be downloaded from web sites such as the National Laboratory for Applied Network Research (NLANR).

Iperf measures maximum bandwidth, allowing you to tune parameters and UDP characteristics. Iperf reports statistics on bandwidth, delay jitter, and datagram loss. On Linux, you can redirect this output to a file; on Windows, use the `-o` *`filename`* parameter.

Run each test for ten minutes to make sure any potential problems have a chance to develop. Use the following command lines to start the sender and receivers.

Sender

```
iperf -c 224.0.166.111 -u -T 1 -t 100 -i 1 -b 1000000000
```

where:

`-c` *`address`*	Run in client mode and connect to a multicast address
`-u`	Use UDP
`-T` #	Multicast time-to-live: number of subnets across which a multicast packet can travel before the routers drop the packet

Do not set the `-T` parameter above 1 without consulting your network administrator. If this number is too high then the `iperf` traffic could interfere with production applications or continue out onto the internet.

`-t` #	Length of time to transmit, in seconds
`-i` #	Time between periodic bandwidth reports, in seconds
`-b` #	Sending bandwidth, in bits per second

Receiver

```
iperf -s -u -B 224.0.166.111 -i 1
```

where:

`-s`	Run in server mode
`-u`	Use UDP
`-B` *`address`*	Bind to a multicast address
`-i` #	Time between periodic bandwidth reports, in seconds

If your GemFire distributed system runs across several subnets, start a receiver on each subnet.

In the receiver's output, look at the `Lost/Total Datagrams` columns for the number and percentage of lost packets out of the total sent.

Example 8.7 Output From Iperf Testing

```
[  ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[   3] 0.0- 1.0 sec 129 KBytes 1.0 Mbits/sec0.778 ms 61/151 (40%)
[   3] 1.0- 2.0 sec 128 KBytes 1.0 Mbits/sec0.236 ms 0/ 89 (0%)
[   3] 2.0- 3.0 sec 128 KBytes 1.0 Mbits/sec0.264 ms 0/ 89 (0%)
[   3] 3.0- 4.0 sec 128 KBytes 1.0 Mbits/sec0.248 ms 0/ 89 (0%)
[   3] 0.0- 4.3 sec 554 KBytes 1.0 Mbits/sec0.298 ms 61/447 (14%)
```

Rerun the test at different bandwidths until you find the maximum useful multicast rate. Start high, then gradually decrease the send rate until the test runs consistently with no packet loss. For example, you might need to run five tests in a row, changing the `-b` (bits per second) parameter each time until there is no loss:

1. `-b 1000000000` (loss)
2. `-b 900000000` (no loss)
3. `-b 950000000` (no loss)
4. `-b 980000000` (a bit of loss)
5. `-b 960000000` (no loss)

Enter `iperf -h` to see all of the command-line options. For more information, see the Iperf user manual.

Configuring Multicast Speed Limits

Once you have determined what the maximum transmission rate should be, configure and tune your production system. For best performance, the producer and the consumers should run on different machines and each process should have at least one CPU dedicated to it. The following is a list of configuration changes that can improve multicast performance. Check with your system administrator about changing any of the limits discussed here.

- Increase the default datagram size for systems running Microsoft Windows from 1024 bytes to a value that matches your network's maximum transmission unit (MTU), which is typically 1500 bytes. The higher setting should improve the system's network performance.
- Distribution statistics for stack time probes are disabled by default to increase multicast performance. To reduce multicast speed, you can enable time statistics by setting the `gemfire.enable-time-statistics` property to `true`.

  This enables time statistics for a Java application:

  ```
  -Dgemfire.enable-time-statistics=true
  ```

  The time statistics properties are passed to the cache server at the command line:

  ```
  cacheserver start -J-Dgemfire.enable-time-statistics=true
  ```

- Monitor the members that receive data for signs of data loss. A few data loss messages can happen normally during region creation. Data loss monitoring can be done by reviewing the GemFire `DistributionStats` in the statistics archive using the optional GemFire Visual Statistics Display (VSD) tool. For more information on GemFire statistics, see Appendix B, *System Statistics*, on page 235.

  If the cache regions are configured to require acknowledgement, you could see messages timing out as they wait for a response. After a put into a region, the next operations might report that the entry could not be found. Multicast retransmit requests and unicast retransmits can also be monitored to detect data loss. Even when you see data loss, the cause of the problem may have nothing to do with

the network. However, if it happens constantly then you should try testing the flow control rate again.

- If necessary, reconfigure all the `gemfire.properties` files and repeat with lower flow control maximum credits until you find the maximum useful rate for your installation. For details on setting flow control credit rates, see `mcast-flow-control` on page 53.
- Slow system performance might be helped by reducing how far your multicast messaging goes in your network. To modify this setting, see `mcast-ttl` on page 54.
- Reduce multicast latency by disabling batching. By default, GemFire uses batching for operations when the region's `scope` is `distributed-no-ack`. Set the `disableBatching` property to `true` on the application or cacheserver command line:

  ```
  -Dp2p.disableBatching=true
  ```

  For more on setting JVM properties, see *JVM Memory Settings* on page 141.

Run-time Considerations for Multicast

This section provides a few topics specific to running GemFire distributed systems using multicast for messaging and data distribution.

Multicast Health Monitor

The GemFire administration API health monitoring system is supplemented by a `maxRetransmissionRatio` health monitoring setting for distributed system members. This ratio is the number of retransmission requests received divided by the number of multicast datagrams written. If the ratio is at 1.0, the member is retransmitting as many packets as it originally sent. Retransmissions are point-to-point, and many processes may request retransmission, so this number can get quite high if problems occur. The default value for `maxRetransmissionRatio` is 0.2.

For example, consider a distributed system with one producer and two consumers of cache events using multicast to transmit cache updates. The new member is added, which is running on a machine without multicast enabled. As a result, there is a retransmission request for every cache update, and the `maxRetransmissionRatio` changes to 1.0.

Controlling Memory Use on GemFire Hosts With Multicast

Running out of memory can impede a member's performance and eventually lead to severe errors.

When data is distributed over multicast, GemFire incurs a fixed overhead of memory reserved for transmission buffers. A specified amount of memory is reserved for each distributed region. These producer-side buffers are used only when a receiver is not getting enough CPU to read from its own receiving buffer as quickly as the producer is sending. In this case, the receiver complains of lost data. The producer then retrieves the data, if it still exists in its buffer, and resends to the receiver.

Tuning the transmission buffers requires a careful balance. Larger buffers mean that more data remains available for retransmission, providing more protection in case of a problem. On the other hand, a larger amount of reserved memory means that less memory is available for caching.

You can adjust the transmission buffer size by resetting the `mcast-send-buffer-size` parameter in the `gemfire.properties` file:

```
mcast-send-buffer-size=45000
```

The maximum buffer size is constrained only by the limits of your system.

If you are not seeing problems that could be related to lack of memory then do not change the default, since it provides greater protection in case of network problems.

Troubleshooting the Multicast Tuning Process

This section covers some of the issues that may come up during the initial testing and tuning process for multicasting.

Some or All Members Cannot Communicate

If your applications and cache servers cannot talk to each other, even though they are configured correctly, you may not have multicast connectivity on your network. It's common to have unicast connectivity, but not multicast connectivity. See your network administrator.

Multicast is Slower Than Expected

Look for an Ethernet flow control limit. If you have mixed-speed networks that result in a multicast flooding problem, the Ethernet hardware may be trying to slow down the fast traffic.

Make sure your network hardware can deal with multicast traffic and route it efficiently. Some network hardware designed to handle multicast does not perform well enough to support a full-scale production system.

Multicast Fails Unexpectedly

If you find through testing that multicast fails above a round number, for example, it works up to 100 Mbps and fails at all rates over that, suspect that it is failing because it exceeds the network rate. This problem often arises at sites where one of the secondary LANs is slower than the main network

Chapter 9

Using JMX to Administer GemFire

This chapter tells how to use the Java Management Extensions (JMX) Agent to administer and manage a single GemFire Enterprise® distributed system. The JMX Agent provides administrative and operational monitoring along with additional functionality such as health monitoring.

You can use the JMX Agent to perform the following management tasks:

- View the distributed system and its settings
- View distributed system members
- View and modify configuration attributes
- View runtime system and application statistics
- View a cache region and its attributes and statistics
- Monitor the health of a GemFire Enterprise system and its components

The JMX Agent should be run as a separate distributed system member. The JMX Agent manages only a single distributed system.

> *The JMX Agent uses connectors specified by JMX Remote v1.0. For information on JMX, see* http://java.sun.com/products/JavaManagement/index.jsp.

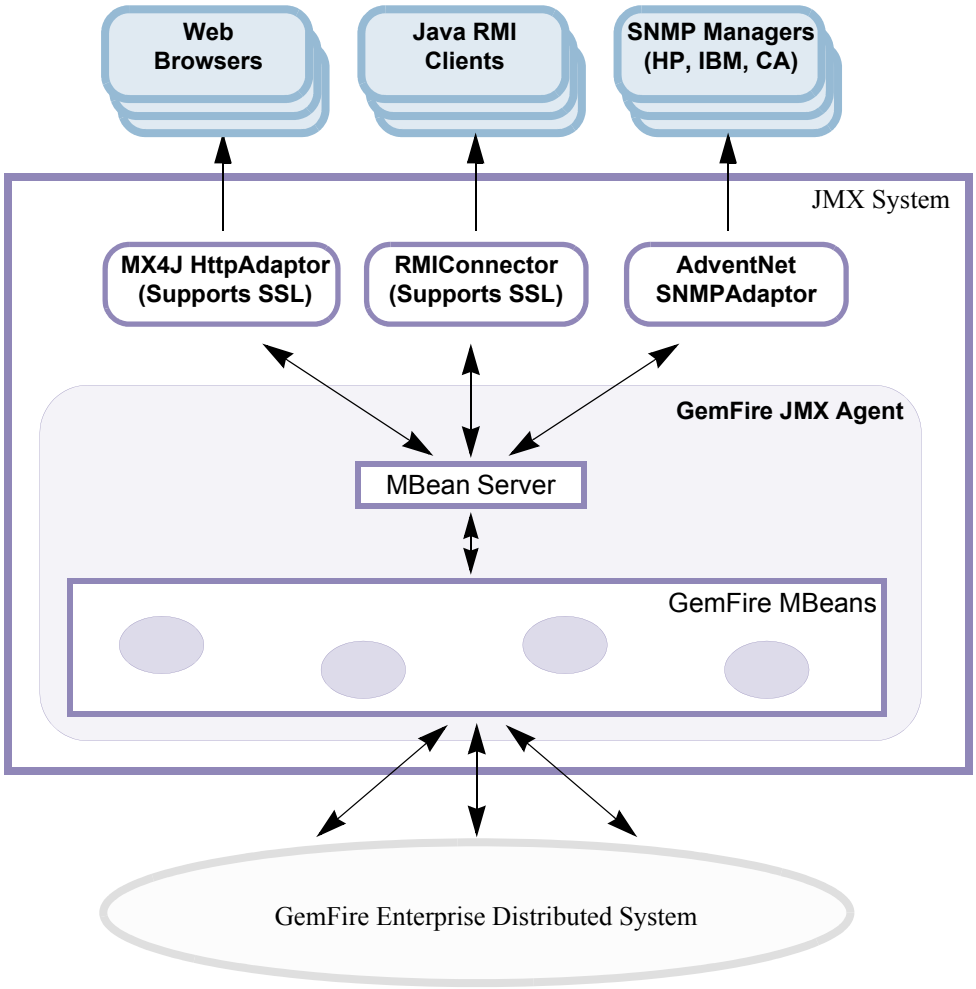
In this chapter:

9.1 Example Configuration

This figure shows a sample configuration in which the JMX agent connects applications to the GemFire Enterprise administrative (`admin`) distributed system. The architecture includes the following components:

- Client applications that connect to the GemFire Enterprise `admin` distributed system via an HTTP web browser, RMI client, or SNMP manager.
- A set of connectors and adaptors that allow clients to contact and interact with the MBeans in the JMX Agent's MBean server (see *Enabling, Disabling, and Configuring Connectors* on page 174).
- A set of GemFire JMX MBeans that are used to manage the `admin` distributed system (see *MBeans* on page 179).

Figure 9.1 The GemFire Enterprise JMX Architecture



9.2 Starting the GemFire JMX Agent

To start the GemFire JMX Agent, you run the API call or invoke the `agent` script (in the GemFire `bin` directory):

```
$GEMFIRE/bin/agent start [-Jvmarg]* [-dir=dir] [prop=value]*
```

vmarg	A VM option passed to the Agent's VM. For example, to define a 1 GB heap, you would include this option: `-J-Xmx1024M`. To set a system property foo.bar to "true", you would add `-J-Dfoo.bar=true`.
dir	The directory in which the Agent's log file is written. The default is the current directory. For details, see *The Agent Log File* on page 178.
prop=value	A configuration property and value passed to the Agent. You can define configuration properties on the command line or in the Agent properties file. For details, see *The Agent Properties File* on page 178.

This command line instantiates a JMX Agent with non-default working directory and property file:

```
$GEMFIRE/bin/agent start -dir=/usr/local/gemfire
      property-file=/usr/local/gemfire/agent/myprops.props
```

When you launch the JMX Agent, you identify the distributed system to which you want to connect by specifying the lookup method (used to discover and communicate with other members of the distributed system) as either IP multicast or the GemFire locator service.

You can specify these attributes as a list of *`prop=value`* pairs on the `agent` command line, for example:

```
mcast-port=0
locators=host1[12345]
```

or

```
mcast-port=10021
```

If you specify a distributed system connection configuration on the `agent` command line, the agent registers an `AdminDistributedSystem` MBean. Alternatively, you can connect by defining the distributed system connection configuration and lookup method in the agent properties file. For details, see *Admin Distributed System Properties* on page 171.

Additional attributes allow you to enable and configure the supported JMX connectors/adaptors (`RMIConnectorServer`, `HTTPAdaptor`, and `AdventNetSNMPAdaptor`). For details, see the tables under *HttpAdaptor* on page 174.

Command-line Arguments

Argument	Comments	Default Value
`property-file`	The name of the properties file to load when starting the JMX Agent. For details, see *The Agent Properties File* on page 178.	
`log-level`	A minimum level of log messages to be written.	`config`
`log-disk-space-limit`	The maximum disk space to allocate for logging, in megabytes in the range `0..1000000`.	`0`
`log-file-size-limit`	The maximum size of the JMX Agent log file, in megabytes in the range `0..1000000`.	`0`
`auto-connect`	If `true`, the JMX Agent automatically connects to the distributed system specified by the arguments `mcast-port`, `mcast-address`, `locators`, and `remote-command`. You can specify these arguments on the command line or in the Agent properties file. For details, see the following section, *Admin Distributed System Properties* on page 171.	`false`
`refresh-interval`	The time interval in seconds after which the system statistics are refreshed.	`5 seconds`

For more command-line arguments, see *Enabling, Disabling, and Configuring Connectors* on page 174 and *SSL Communication* on page 177.

Admin Distributed System Properties

You can specify the following `admin` distributed system-specific properties as a list of *`prop=value`* pairs on the `agent` command line or in the Agent's properties file. The SSL properties listed here affect communication between members of the distributed system.

Argument	Comments	Default Value
`mcast-address`	The multicast address of this distributed system. To use IP multicast, you must also define `mcast-port`, the IP port.	`239.192.81.1`
`mcast-port`	The multicast port, a value in the range `0..65535`. To use IP multicast, you must also define `mcast-address`, the IP address.	`10334`
`membership-port-range`	The range of ports available for unicast UDP messaging and for TCP failure detection. This is specified as two integers separated by a minus sign. Different members can use different ranges. GemFire randomly chooses two unique integers from this range for the member, one for UDP unicast messaging and the other for TCP failure detection messaging. Additionally, the system uniquely identifies the member using the combined host IP address and UDP port number. You may want to restrict the range of ports that GemFire uses so the product can run in an environment where routers only allow traffic on certain ports.	1024-65535
`locators`	A comma-delimited list whose elements have the form `host[port]`. When you use the GemFire locator service, each locator is uniquely identified by the host on which it is running and the port on which it is listening. For details, see *Configuring Member Discovery and Communication* on page 59.	`""`
`remote-command`	A default remote command prefix to use for command invocation on remote machines.	`rsh -n {HOST} {CMD}`
`ssl-enabled`	Indicates whether to use the Secure Sockets Layer (SSL) protocol for communication between members of this distributed system. Valid values are `true` and `false`. A `true` setting requires the use of locators.	`false`
`ssl-protocols`	A space-separated list of the valid SSL protocols for this connection. You can specify `any` to use any protocol that is enabled by default in the configured Java Secure Sockets Extension (JSSE) provider.	`any`
`ssl-ciphers`	A space-separated list of the valid SSL ciphers for this connection. You can specify `any` to use any ciphers that are enabled by default in the configured JSSE provider.	`any`

Argument	Comments	Default Value
`ssl-require-authentication`	Indicates whether to require authentication for communication between members of the `admin` distributed system. Valid values are `true` and `false`.	`true`
`tcp-port`	The TCP port to listen on for cache communications. If set to zero, the operating system selects an available port. Each process on a machine must have its own TCP port. Note that some operating systems restrict the range of ports usable by non-privileged users, and using restricted port numbers can cause runtime errors in GemFire startup. Valid values are in the range `0..65535`.	`0`

E-Mail Notification Properties

You can have the agent provide e-mail notification for alerts and membership change events. You can specify the notification properties in the Agent's properties file or as a list of *prop=value* pairs on the `agent` command line. The properties file is the recommended method.

Argument	Comments	Default Value
`email-notification-enabled`	Whether to send e-mail notifications.	`false`
`email-notification-from`	The from address to put into the e-mail notifications.	`""`
`email-notification-host`	The host where the mail server is running - used to send the notifications. This must be set for mails to be sent. The server's default port is used for the notifications.	`""`
`email-notification-to`	A comma-separated list of e-mail addresses to which to send the notifications. This must be set for mails to be sent.	`""`

This example shows how you might define properties for a e-mail notification in the Agent's properties file.

Example 9.1 Defining E-Mail Notification Properties

```
email-notification-enabled=true
email-notification-from=”dashiell.hammett@gemstone.com”
email-notification-host=”thinman”
email-notification-to”nick@gemstone.com,nora@gemstone.com,asta@gemstone.com”
```

These are the notifications the system sends:

Table 9.1 E-Mail Notifications

Notification Type	E-Mail Subject Line
System Alert—System alert that is raised in the members and can be set by a user.	[Gemfire Alert] Distributed System: <Distributed System Identifier> <System Alert>
Member Crash—Alert of a member crash.	[Gemfire Alert] Distributed System: <Distributed System Identifier> <Member Crash>
Stat Alert—Created via GFMon, this evaluates a threshold for a functional value of one or more statistics.	[Gemfire Alert] Distributed System: <Distributed System Identifier> <Statistics Alert for member>
Membership change—Notification of a member joining, leaving, or being forcibly disconnected	[GemFire Notification]Distributed System:<Distributed System Identifier> <Member Joined> [GemFire Notification]Distributed System:<Distributed System Identifier> <Member Left>

9.3 Enabling, Disabling, and Configuring Connectors

The JMX Agent is supported for use with the `RMIConnectorServer` and MX4J `HttpAdaptor`, and is designed to integrate with the AdventNet SNMP Adaptor for JMX. This section describes these three connectors and tells how to configure the JMX Agent for the selected connector.

- HttpAdaptor
- RMIConnectorServer
- AdventNetSNMPAdaptor

HttpAdaptor

The MX4J `HttpAdaptor` provides an HTML user interface to all MBeans in the MBeanServer. The `HttpAdaptor` provides a functional and easy-to-use interface with no development required, and is particularly useful to developers who want to explore and browse GemFire JMX MBeans. For details, consult the online documentation on the MX4J website:

- http://mx4j.sourceforge.net/docs/index.html—MX4J Guide
- http://mx4j.sourceforge.net/docs/ch05.html—MX4J HttpAdaptor documentation

Access the `HttpAdapter` through your browser, using this URL:

```
http://HttpAdaptor_host:port
```

By default, *`port`* is `8080`.

You can specify the following `HttpAdaptor` properties in the Agent properties file or as a list of *`prop=value`* pairs on the `agent` command line (see *Starting the GemFire JMX Agent* on page 169).

Argument	Comments	Default Value
`http-enabled`	To enable the `HTTPAdaptor`, this must be `true`.	`true`
`http-bind-address`	The machine name or IP address to which the HTTP listening socket should be bound. If this value is "localhost", then the socket is bound to the loopback address (127.0.0.1) and the adapter is only accessible via the URL http://local-host:8080. If null, all network addresses are used.	`null`
`http-port`	The value must be in the range `0..65535`.	`8080`
`http-authentication-enabled`	If `true`, require a password.	`false`
`http-authentication-user`	User name.	`admin`
`http-authentication-password`	User password.	`password`

RMIConnectorServer

The `RMIConnectorServer` allows clients to contact and interact with the MBeans in the JMX Agent's MBean server, as specified by JSR 160 JMX Remote.

Under JRE 1.5, `RMIConnectorServer` is provided by JRE 1.5. For details, see http://java.sun.com/j2se/1.5.0/docs/guide/jmx/tutorial/connectors.html.

You can specify the following `RMIConnectorServer` properties in the Agent properties file or as a list of *`prop=value`* pairs on the `agent` command line (see *Starting the GemFire JMX Agent* on page 169).

Argument	Comments	Default Value
`rmi-bind-address`	An IP address that the JMX Agent uses to communicate with the `admin` distributed system. This is required: ▸ on multi-homed hosts (machines with multiple network cards) ▸ on Windows systems when using IPv6 The `rmi-bind-address` argument must be specified on the `agent start` command line if `jconsole` or `jmanage` are running on a different host. If set to null - "" - all network addresses are used. For details, see *Selecting a Network Adapter Through a Bind Address* on page 69.	`""`
`rmi-enabled`	To enable the `RMIConnectorServer`, this must be `true`.	`true`
`rmi-port`	The RMI registry port, a value in the range `0..65535`.	`1099`
`rmi-registry-enabled`	If `true`, create an MX4J Naming MBean to serve as the RMI registry, and register the `RMIConnector` under the JNDI path `/jmxconnector`. More information is also available at the Java API reference page for `com.gemstone.gemfire.admin.jmx.Agent`.	`true`
`rmi-server-port`	The port to use for the `RMIConnectorServer`. If set to 0 (zero) the server socket uses a dynamically allocated port. You might want to specify the port to use when the JMX agent is behind the firewall, for example. Valid values are in the range `0..65535`.	`0`

AdventNetSNMPAdaptor

GemFire is designed to integrate with the AdventNet SNMP Adaptor for JMX, a third-party product that is available for purchase from AdventNet,http://www.adventnet.com/index.html. AdventNet SNMP Adaptor for JMX enables you to monitor and manage GemFire from a SNMP Manager such as IBM Tivoli, HP OpenView, or CA Unicenter. Configuration of the SNMPAdaptor is complex and varies from customer to customer based on your requirements and what you need to monitor.

You can specify the following AdventNetSNMPAdaptor properties in the Agent properties file or as a list of *prop=value* pairs on the `agent` command line. For more information, see *Starting the GemFire JMX Agent* on page 169.

Argument	Comments	Default Value
`snmp-enabled`	To enable the `AdventNetSNMPAdaptor`, this must be `true`.	`false`
`snmp-directory`	No default value—you must specify a valid directory.	No default
`snmp-bind-address`	An IP address that the JMX agent uses to communicate with the distributed system. For details, see *Selecting a Network Adapter Through a Bind Address* on page 69.	`""`

9.4 SSL Communication

You can configure the JMX Agent to use the Secure Sockets Layer (SSL) protocol for connections outside of GemFire. To do so, you specify the following properties in the Agent's properties file or as a list of *prop*=*value* pairs on the `agent` command line.

Argument	Comments	Default Value
`agent-ssl-enabled`	Indicates whether the JMX Agent uses the Secure Sockets Layer (SSL) protocol for communication outside of GemFire.	`false`
`agent-ssl-protocols`	A space-separated list of the valid SSL protocols for this connection. You can specify `any` to use any protocol that is enabled by default in the configured Java Secure Sockets Extension (JSSE) provider.	`any`
`agent-ssl-ciphers`	A space-separated list of the valid SSL ciphers for this connection. You can specify `any` to use any of the ciphers that are enabled by default in the configured JSSE provider.	`any`
`agent-ssl-require-authentication`	If `true`, require client authentication for RMI and other non-HTTP connectors/adaptors.	`true`
`http-ssl-require-authentication`	If `true`, require client authentication for HTTP adaptors.	`false`

You can also configure a GemFire Enterprise system to use SSL protocol for communication between system members. For details, see *Admin Distributed System Properties* on page 171.

9.5 Properties and Log Files

The Agent Properties File

By default, the Agent properties file is named `agent.properties`. You can specify a different properties file on the command line when you launch the JMX Agent. For details, see *Starting the GemFire JMX Agent* on page 169.

The Agent looks for the properties file in the following locations, in order:

- A directory that you explicitly specify with the `-dir` argument when starting the Agent
- The current directory
- Your home directory (the default)
- The `CLASSPATH`

You can modify the values in the properties file via the HttpAdaptor or any supported JMX interface.

The Agent Log File

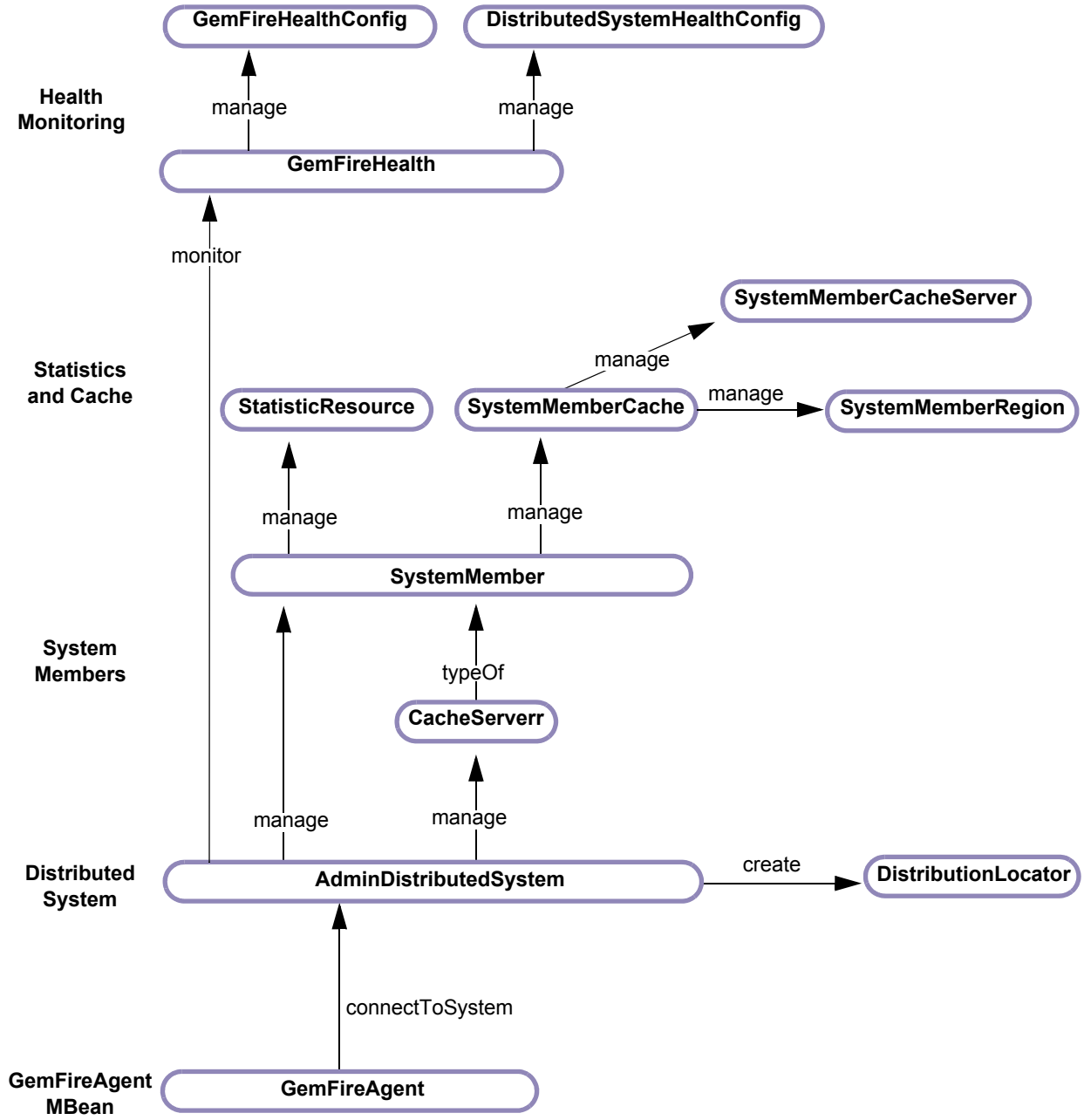
By default, the Agent log file is named `agent.log`. You can specify a different log file as a command-line argument when you launch the JMX Agent. For details, see *Starting the GemFire JMX Agent* on page 169.

9.6 MBeans

GemFire JMX MBeans are ModelMBeans that manage instances of the Admin API objects housed in the JMX Agent's MBeanServer. The JMX Agent hosts an MBeanServer, instances of all MBeans registered for managing a distributed system, and server connectors for various types of clients.

This figure shows the GemFire JMX MBeans used to manage a GemFire Enterprise system. The subsequent paragraphs describe the MBeans individually, along with information about each MBean's key attributes and operations.

Figure 9.2 GemFire JMX MBeans



- **GemFireAgent**—Represents the GemFire JMX Agent. `GemFireAgent` attributes include the name of the Agent properties and log files, limits for log file size and disk usage, locators, bind address information, and SSL information. For details, see *Command-line Arguments* on page 170 and *Admin Distributed System Properties* on page 171. `GemFireAgent` operations include adding and removing SSL vendor properties, managing the log file, saving configuration settings to the properties file, and connecting to the distributed system.

  After the `GemFireAgent` MBean has connected to a distributed system, you can still invoke the operation `connectToSystem` to return the `ObjectName` for the `AdminDistributedSystem` MBean.

- **AdminDistributedSystem**—Represents the GemFire Enterprise distributed system, which is defined by three attributes: `mcastAddress`, `mcastPort`, and `locators`. `AdminDistributedSystem` operations include starting and stopping locators, creating `DistributionLocator` MBeans, managing locators and applications, monitoring GemFire health, and displaying merged logs, licensing information, and system alerts.

  The `AdminDistributedSystem` MBean provides several predefined JMX Notifications that you can use to monitor your distributed system. In addition to these Notifications, you can use the JMX Monitor Service to monitor any attribute of any MBean.

- **DistributionLocator**—Represents a locator within a distributed system. Each locator is identified by its host, port, and bind address attributes. `DistributionLocator` operations include starting, stopping, and removing locators. For more about configuring and running locators, see *Selecting a Network Adapter Through a Bind Address* on page 69.

- **CacheServer**—Represents a GemFire `cacheserver` in a distributed system. A `CacheServer` MBean is a type of `SystemMember` MBean.

  `CacheServer` has all of the `SystemMember` MBean attributes and operations, along with additional attributes for the server's directory (where configuration and logging files are stored). `CacheServer` operations include starting and stopping the GemFire cache server, and creating and removing entries in the agent properties file.

- **SystemMember**—Represents a Java VM running as a member of a distributed system. `SystemMember` operations include creating `StatisticResource` and `SystemMemberCache` MBeans.

  `SystemMember` MBeans have a dynamically added attribute for each GemFire configuration parameter. Some attributes are mutable.

- **StatisticResource**—An MBean to monitor runtime and application statistics. Each `StatisticResource` MBean has dynamically added attributes that correspond to each statistic in the resource.

- **SystemMemberCache**—Represents a GemFire system member cache. When developing or monitoring a caching application, the cache contents provide a rich source of useful information.

- **SystemMemberCacheServer**—Represents a GemFire cache server for a system member cache. The cache server handles data requests and updates from a client tier in a hierarchical cache and manages communication with an external data source. The cache server can be created from the `SystemMemberCache` and can be started and stopped.

- **SystemMemberRegion**—Represents a snapshot of a `Region's` state in the GemFire system member's cache. The interface includes a `refresh` method that updates the snapshot.

- **GemFireHealth**—Allows you to monitor the health of a given distributed system, along with the components residing on individual host machines in the distributed system.

  This MBean is created by invoking the behavior `monitorGemFireHealth` on the `AdminDistributedSystem` MBean. That behavior simultaneously creates the

`DistributedSystemHealthConfig` and default `GemFireHealthConfig` MBeans. You can create additional `GemFireHealthConfig` MBeans for each host in the system.

The `DistributedSystemHealthConfig` and `GemFireHealthConfig` MBeans allow you to configure performance thresholds for each component type in the distributed system, including the distributed system itself. These threshold settings are compared to system statistics to obtain a report on each component's health. A component is considered to be in good health if all of the user-specified criteria for that component are satisfied.

- **DistributedSystemHealthConfig**—Configures how the health of a distributed system is determined.
- **GemFireHealthConfig**—Configures how to determine the health of GemFire components running on a single machine: cache servers, cache instances, and other members of the distributed system.

  `GemFireHealthConfig` extends the `MemberHealthConfig` and `CacheHealthConfig` interfaces.

  `MemberHealthConfig` attributes configure how to determine the health of individual members of the distributed system, using such measures as:

  - VM process size
  - Multicast and retransmissions
  - Incoming message queue sizes
  - Number of timeouts waiting for replies from other members

  These attributes apply to each member that has joined the distributed system.

  `CacheHealthConfig` attributes configure how to determine the health of cache instances, using such measures as:

  - Durations for `netSearch` and `load` operations
  - Cache hit ratio

  These attributes apply to every cache in the distributed system.

9.7 Programming Example

The following brief example shows how you might connect to the JMX agent using the RMIConnector and manipulate the `AdminDistributedSystem` MBean.

Example 9.2 Connecting to the JMX Agent and Manipulating the AdminDistributedSystem MBean

```
JMXServiceURL url = new
JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:1099/jmxconnector");
JMXConnector conn = JMXConnectorFactory.connect(url);
MBeanServerConnection mbsc = conn.getMBeanServerConnection();
ObjectName agentName = new ObjectName("GemFire:type=Agent");
ObjectName distName = (ObjectName) mbsc.invoke(agentName, "connectToSystem",
new Object[] {}, new String[] {});

MBeanInfo distInfo = mbsc.getMBeanInfo(distName);

String description = distInfo.getDescription();
MBeanAttributeInfo[] attrs = distInfo.getAttributes();
MBeanConstructorInfo[] ctors = distInfo.getConstructors();
MBeanNotificationInfo[] nots = distInfo.getNotifications();
MBeanOperationInfo[] opers = distInfo.getOperations();
```

The first line of the example specifies the machine (`localhost`) and port (`1099`) of the machine on which the JMX Agent is running.

```
JMXServiceURL url = new
JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:1099/jmxconnector"
);
JMXConnector conn = JMXConnectorFactory.connect(url);
```

The next line creates a connection to the MBeanServer.

```
MBeanServerConnection mbsc = conn.getMBeanServerConnection();
```

The next line specifies the ObjectName of the JMX Agent. All MBeans are referenced using an ObjectName. If you don't know the ObjectName, you can use the querying capabilities of the MBeanServer to obtain it.

```
ObjectName agentName = new ObjectName("GemFire:type=Agent");
```

The JMX Agent connects to the AdminDistributedSystem? and registers an AdminDistributedSystem MBean and returns the ObjectName for this AdminDistributedSystem MBean instance.

```
ObjectName distName = (ObjectName) mbsc.invoke(agentName,
"connectToSystem", new Object[] {}, new String[] {});
```

The next line asks the MBeanServer for information about the specified AdminDistributedSystem?. JMX allows you to obtain all of this information programmatically.

```
MBeanInfo distInfo = mbsc.getMBeanInfo(distName);
```

These lines provide detailed information: a description of the distributed system, its attributes, constructors, notifications, and operations.

```
String description = distInfo.getDescription();
MBeanAttributeInfo[ ] attrs = distInfo.getAttributes();
MBeanConstructorInfo[ ] ctors = distInfo.getConstructors();
MBeanNotificationInfo[ ] nots = distInfo.getNotifications();
MBeanOperationInfo[ ] opers = distInfo.getOperations();
```

9.8 Stopping the GemFire JMX Agent

To stop the GemFire JMX Agent, issue the following command:

```
$GEMFIRE/bin/agent stop [-dir=dir]
```

where *dir* is the directory in which the Agent is running.

Chapter

10 GemFire System Logging

This chapter describes GemFire Enterprise® system logging. The logs output by your system have their own characteristics, indicative of your system configuration and of the particular behavior of your applications. Because of this, in order to detect anomalies and problems in your system, you must become familiar with the log files your applications generate. Use a sniffer to monitor your logs and, if you begin seeing new or unexpected warnings, errors, or severe messages, contact GemStone Technical Support (page 17).

> *CAUTION: You must run a time synchronization service such as NTP on all hosts to produce useful logs for troubleshooting. Synchronized time stamps ensure that log messages on different hosts can be merged to accurately reproduce a chronological history of a distributed run.*

In this chapter:

10.1 Overview of Logging

GemFire Enterprise provides comprehensive logging messages to help you confirm system configuration and to debug problems in configuration and code. In addition to the system logs discussed here, you can add your own application logs from your Java code. For information on adding custom logging to your applications, see the online Java documentation for the `com.gemstone.gemfire.LogWriter` interface. Both system and application logging is output and stored according to the distributed system configuration parameters in *Logging Options* on page 187.

Logging Categories

Most system logging output falls into one of several general categories.

- **Startup Information**—This includes all information about the system and configuration the process is running with. This describes the Java version, the GemFire native version, the host system, current working directory, and environment settings.
- **Logging management**—These messages pertain to the maintenance of the log files themselves. This information is always in the main log file (see the discussion at *Log File Name* on page 191).
- **Connections and system membership**—These report on the arrival and departure of distributed system members (including the current member) and any information related to connection activities or failures. This includes information on communication between tiers in a hierarchical cache.
- **Distribution messages**—These pertain to the distribution of data between system members. These include message regarding region configuration, entry creation and modification, and region and entry invalidation and destruction.
- **Cache, region, and entry management**—This includes cache initialization, listener activity, locking and unlocking, region initialization, entry updates.
- **Exceptions**—These include standard Java exceptions and GemFire exceptions. These may originate in the current member or be propagated from another member.

The Log Message

Every logged message contains:

- The message header within square brackets:
  - The message level. For possible values, see *Log Level* on page 187.
  - The time the message was logged.
  - The ID of the connection and thread that logged the message. This might be the main program or a system management process.
- The message itself, which can be a string and/or an exception including the exception's stack trace.

```
[config 2005/11/08 15:46:08.710 PST PushConsumer main nid=0x1] Cache
initialized using "file:/Samples/quickstart/xml/PushConsumer.xml".
```

Searching the Log Files

For the clearest picture, merge the log files. Search for lines that begin with these strings:

```
[warning
[error
[severe
```

See also *Producing Data Files for Troubleshooting* on page 196.

10.2 Logging Options

This section lists the logging options available to you in GemFire Enterprise, all of which are specified as GemFire properties. With these logging properties, you can control where and how to log system and application messages. You can also control the amount of disk space your log files are allowed to consume. These settings are generally set as `gemfire.properties` file settings. For more information, see *System Properties in the gemfire.properties File* on page 48.

The logging options are:

- Log Level (page 187)
- Log File Name (page 191)
- Maximum Size of a Single Log File (page 192)
- Maximum Size of All Log Files (page 193)

Log Level

The `log-level` (page 52) determines which of the messages sent to the logger are actually output to the log file. All messages at or above the specified log level are output to the log file. So for example, if the level is set to `info`, all messages with level `info`, `warning`, `error`, and `severe` are output to the log.

The higher the log level, the more important and urgent the message. If you are having problems with your system, a first-level approach is to lower the log-level (thus sending more of the detailed messages to the log file) and recreate the problem. The additional log messages often help uncover the source.

These are the levels, in descending order, along with some example output.

severe (Highest Level)

This level indicates a serious failure. In general, severe messages describe events that are of considerable importance and which will prevent normal program execution. You will likely need to shut down or restart at least part of your system to correct the situation.

This severe error was produced by configuring a system member to connect to a non-existent locator:

```
[severe 2005/10/24 11:21:02.908 PDT nameFromGemfireProperties DownHandler
(FD_SOCK) nid=0xf] GossipClient.getInfo(): exception connecting to host
localhost:30303: java.net.ConnectException: Connection refused
```

error

This level indicates that something is wrong in your system. You should be able to continue running, but the operation indicated by the error message failed.

This error was produced by throwing a `Throwable` from a `CacheListener`. While dispatching events to a customer-implemented `CacheListener`, GemFire catches any `Throwable` thrown by the listener and logs it as an error. The text shown here is followed by the output from the `Throwable` itself.

```
[error 2007/09/05 11:45:30.542 PDT gemfire1_newton_18222
<vm_2_thr_5_client1_newton_18222-0x472e> nid=0x6d443bb0] Exception
occurred in CacheListener
```

warning

This level indicates a potential problem. In general, warning messages describe events that are of interest to end users or system managers, or that indicate potential problems in the program or system.

This message was obtained by starting a client with a `Pool` configured with queueing enabled when there was no server running to create the client's queue:

```
[warning 2008/06/09 13:09:28.163 PDT <queueTimer-client> tid=0xe]
QueueManager - Could not create a queue. No queue servers available
```

And this message was obtained by trying to `get` an entry in a client region while there was no server running to respond to the client request:

```
[warning 2008/06/09 13:12:31.833 PDT <main> tid=0x1] Unable to create a
connection in the allowed time
com.gemstone.gemfire.cache.client.NoAvailableServersException
   at
com.gemstone.gemfire.cache.client.internal.pooling.ConnectionManagerImpl.
borrowConnection(ConnectionManagerImpl.java:166)
. . .
com.gemstone.gemfire.internal.cache.LocalRegion.get(LocalRegion.java:1122
)
. . .
```

info

This level for informational messages. Typically, these messages are for end users and system administrators.

This is a typical info message created at system member startup. The message indicates that no other `DistributionManager`s (therefore, no other system members) are running in the distributed system:

```
[info 2005/10/24 11:51:35.963 PDT CacheRunner main nid=0x1]
DistributionManager straw(7368):41714 started on 224.0.0.250[10333] with
id straw(7368):41714 (along with 0 other DMs)
```

When another system member joins the distributed system, these info messages are output by the members that are already running:

```
[info 2005/10/24 11:52:03.934 PDT CacheRunner P2P message reader for
straw(7369):41718 nid=0x21] Member straw(7369):41718 has joined the
distributed cache.
```

When another member leaves because of an interrupt or through normal program termination:

```
[info 2005/10/24 11:52:05.128 PDT CacheRunner P2P message reader for
straw(7369):41718 nid=0x21] Member straw(7369):41718 has left the
distributed cache.
```

And when another member is killed:

```
[info 2005/10/24 13:08:41.389 PDT CacheRunner DM-Puller nid=0x1b] Member
straw(7685):41993 has unexpectedly left the distributed cache.
```

config

This is the default setting for logging. This level provides static configuration messages that are often used to debug problems associated with particular configurations. The first configuration message logged lists the GemFire properties used for the process. You can use this to verify your startup configuration:

```
[config 2008/08/08 14:28:19.862 PDT CacheRunner <main> tid=0x1] Startup
Configuration:
   ack-severe-alert-threshold="0"
   ack-wait-threshold="15"
   archive-disk-space-limit="0"
   archive-file-size-limit="0"
   async-distribution-timeout="0"
   async-max-queue-size="8"
   async-queue-timeout="60000"
   bind-address=""
   cache-xml-file="cache.xml"
   conflate-events="server"
   conserve-sockets="true"
   departure-correlation-window="1800"
   disable-tcp="false"
   durable-client-id=""
   durable-client-timeout="300"
   enable-network-partition-detection="false"
   enable-time-statistics="false"
   license-file="gemfireLicense.zip"
   license-type="evaluation"
   locators=""
   log-disk-space-limit="0"
   log-file=""
   log-file-size-limit="0"
   log-level="config"
   max-num-reconnect-tries="3"
   max-wait-time-reconnect="10000"
   mcast-address="239.192.81.1"
   mcast-flow-control="1048576, 0.25, 5000"
   mcast-port="10334"
   mcast-recv-buffer-size="1048576"
   mcast-send-buffer-size="65535"
   mcast-ttl="32"
   member-timeout="5000"
   name="CacheRunner"
   roles=""
   security-=""
   security-client-accessor=""
   security-client-accessor-pp=""
   security-client-auth-init=""
   security-client-authenticator=""
   security-client-dhalgo=""
   security-log-file=""
   security-log-level="config"
   security-peer-auth-init=""
   security-peer-authenticator=""
   security-peer-verifymember-timeout="1000"
   server-bind-address=""
   socket-buffer-size="32768"
   socket-lease-time="60000"
   ssl-ciphers="any"
```

```
ssl-enabled="false"
ssl-protocols="any"
ssl-require-authentication="true"
start-locator=""
statistic-archive-file=""
statistic-sample-rate="1000"
statistic-sampling-enabled="false"
tcp-port="0"
udp-fragment-size="60000"
udp-recv-buffer-size="1048576"
udp-send-buffer-size="65535"
```

fine

This level provides tracing information that is generally of interest to developers. It is used for the lowest volume, most important, tracing messages.

> *Generally, you should only use this level if instructed to do so by GemStone technical support. At this logging level, you will see a lot of noise that might not indicate a problem in your application. This level creates very verbose logs that may require significantly more disk space than the higher levels.*

Fine logging tells you which of the licenses from your license file are considered for this process. The license file is scanned until a valid license is found for the current process. The examples below show the examination and rejection of a license file entry for an application that was started on a Linux machine.

```
[fine 2005/10/11 10:04:10.177 PDTMain Thread-0x152a nid=0xb746a2a0]
Checking license "201.103.12.68.development.license" from
"file:/home/users/jpearson/gemfire/gemfireLicense.zip":
   product = GemFire
   platform = Windows
   license-type = development
   license-version = 3.0
   customer-id = 210382988
   customer-name = Test Customer
   group-id = 1
   License never expires.
   License is limited to node(s) "201.103.12.68 201.103.12.87".
   License had no native node limits.
   License is limited to 2 cpus.
   License allows for hyperthreaded cpus.
   Actual number of purchased cpus is "2".

[fine 2005/10/11 10:04:10.178 PDTMain Thread-0x152a nid=0xb746a2a0]
Ignored license "201.103.12.68.development.license" because it was for
platforms "Windows" and not for platform "Linux".
```

finer, finest, and all

These levels are for internal use only. They produce a large amount of data and so consume large amounts of disk space and system resources. Do not use these settings unless asked to do so by GemStone Technical Support.

Log File Name

The name of a GemFire system member's main log is specified by the `log-file` (page 52). This is the name of the most recent log file, actively in use if the member is running, or used for the last run. This file is created when the application starts.

By default, the main log contains the entire log for the member session. If you specify a `log-file-size-limit` (page 52), then this file is rolled into backup, child logs, the main log is used for the current log, and a file with `meta-` prepended to the name is used to track of startup, shutdown, child log management, and other logging management operations. The current log is renamed to the next available child log when the specified size limit is reached.

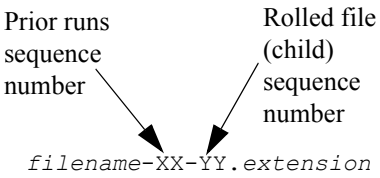
You can always look at the main log file or the `meta-` log file to figure out what the member is currently doing or what it did the last time it was running.

When your application connects with logging enabled, it creates the main log file and, if required, the `meta-` log file. If log files are present when the member starts up, it is renamed to the next available child log in the last session to make way for new logging.

For locators, the log file name is fixed. For the standalone locator, it is always named `locator.log`. For the locator that runs colocated inside another member, the log file is the member's file.

For applications and the cacheserver, your log file specification can be relative or absolute. If no file is specified, the defaults are standard output for applications and `cacheserver.log` for the cacheserver.

Your current, main log file always has the name you specified in `log-file`. The old log files and child log files have names derived from the main log file name. These are the pieces of a renamed log or child log file name where *filename.extension* is your `log-file` specification:



If rolling is not used, the rolled file sequence number is a constant `00` (two zeros).

For a discussion of file renaming, see *How the System Renames Your Logs* on page 191. For a discussion of rolled logs, see *Maximum Size of a Single Log File* on page 192.

Log Naming Recommendation

You will have an easier time deciphering logging output if you keep your members' log files separated. For members running on the same machine, you can accomplish this by starting them in different working directories and using the same, relative `log-file` specification. For example, you could have this `log-file` specification in a common `gemfire.properties` file:

```
log-file=./log/member.log
```

Then start each member in a different directory with this command:

```
java -DgemfirePropertyFile=commonLocation/gemfire.properties ...
```

In this way, each member has its own log files under its own working directory.

How the System Renames Your Logs

The log file that you specify is the base name used for all logging and logging archives. If a log file with the specified name already exists at startup, the distributed system automatically renames it before

creating the current log file. This is a typical directory listing after a few runs with a `log-file` specification, `system.log`:

```
bash-2.05$ ls -tlra system*
-rw-rw-r-- 1 jpearson users 11106 Nov 3 11:07 system-01-00.log
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:08 system-02-00.log
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:09 system.log
bash-2.05$
```

The first run created `system.log` with a timestamp of `Nov 3 11:07`. The second run renamed that file to `system-01-00.log` and created a new `system.log` with a timestamp of `Nov 3 11:08`. The third run renamed that file to `system-02-00.log` and created the file named `system.log` in this listing.

When the distributed system renames the log file, it assigns the next available number to the new file, as *`XX`* of *`filename-XX-YY.extension`*. This next available number depends on existing old log files and also on any old statistics archives. The system assigns the next number that is higher than any in use for statistics or logging. This keeps current log files and statistics archives paired up regardless of the state of the older files in the directory.

Thus, if an application is archiving statistics and logging to `system.log` and `statArchive.gfs`, and it runs in a (Unix) directory with these files:

```
bash-2.05$ ls -tlr stat* system*
-rw-rw-r-- 1 jpearson users 56143 Nov3 11:07 statArchive-01-00.gfs
-rw-rw-r-- 1 jpearson users 56556 Nov3 11:08 statArchive-02-00.gfs
-rw-rw-r-- 1 jpearson users 56965 Nov3 11:09 statArchive-03-00.gfs
-rw-rw-r-- 1 jpearson users 11308 Nov3 11:27 system-01-00.log
-rw-rw-r-- 1 jpearson users 59650 Nov3 11:34 statArchive.gfs
-rw-rw-r-- 1 jpearson users 18178 Nov3 11:34 system.log
```

the directory contents after the run look like this (changed files in **bold**):

```
bash-2.05$ ls -ltr stat* system*
-rw-rw-r-- 1 jpearson users 56143 Nov3 11:07 statArchive-01-00.gfs
-rw-rw-r-- 1 jpearson users 56556 Nov3 11:08 statArchive-02-00.gfs
-rw-rw-r-- 1 jpearson users 56965 Nov3 11:09 statArchive-03-00.gfs
-rw-rw-r-- 1 jpearson users 11308 Nov3 11:27 system-01-00.log
-rw-rw-r-- 1 jpearson users 59650 Nov3 11:34 statArchive-04-00.gfs
-rw-rw-r-- 1 jpearson users 18178 Nov3 11:34 system-04-00.log
-rw-rw-r-- 1 jpearson users 55774 Nov4 10:08 statArchive.gfs
-rw-rw-r-- 1 jpearson users 17681 Nov4 10:08 system.log
```

The statistics and the logging file are renamed using the next integer that is available to both. Thus the logging file sequence jumps past the gap in this case.

Merging Log Files

You can merge multiple log files into a single log file using the command `gemfire merge-logs` (page 230).

Maximum Size of a Single Log File

The `log-file-size-limit` (page 52) sets the maximum size of your individual log files. If you set this, GemFire Enterprise uses rolled log files in conjunction with the main log to keep log file sizes below the specified threshold. The main log file is used for current logging. The child logs hold all old files that were rolled due to reaching the maximum size or member restart.

If the value of `log-file-size-limit` is greater than zero, when the log file reaches the limit, it is renamed to the next available child log name. If there is a file with the log-file name at startup, it is renamed to the next available child log name for the prior loggin sequence.

If you modify `log-file-size-limit` while the distributed system is running, the new value does not take effect until the current active log rolls, using the old limit.

Maximum Size of All Log Files

The `log-disk-space-limit` (page 52) attribute controls the maximum size of all log files combined. By default, `log-disk-space-limit` is `0`, meaning that log space is unlimited.

Whenever a log is rolled, or when an old main log is renamed when a system is started, the combined size of the inactive log files is calculated (that is, the total of all inactive logs in the same directory as the main log and with the same base name as the main log). If the combined size exceeds the `log-disk-space-limit`, the inactive log with the oldest modification time is deleted. This continues until the current space size is less than `log-disk-space-limit`.

If `log-disk-space-limit` is less than or equal to `log-file-size-limit`, then when the active log is made inactive due to its size, it is immediately deleted.

If you modify `log-disk-space-limit` while the distributed system is running, the new value does not take effect until the current active log becomes inactive.

Chapter

11 Troubleshooting and System Recovery

This chapter provides strategies for handling common errors and failure situations. It includes information on how to configure your GemFire Enterprise® system to collect statistics that are helpful in diagnosing common error situations and lists common problems and their solutions. There is also a section on how to recover from system failure.

In this chapter:

11.1 Producing Data Files for Troubleshooting

Save these files, because they are valuable for troubleshooting.

- Log files. Even at the default logging level, the log contains data that may be important. Save the whole log, not just the stack. For comparison, save log files from before, during, and after the problem occurred.
- Statistics archive files.
- Core files.
- For Linux, you can use `gdb` to extract a stack from a core file.
- Crash dumps.
- For Windows, save the Dr. Watson output.

When a problem arises that involves more than one process, a network problem is the most likely cause. When you diagnose a problem, create a log file for each member of all the distributed systems involved. If you are running a client/server architecture, create log files for the clients.

> *CAUTION: You must run a time synchronization service on all hosts for troubleshooting. Synchronized time stamps ensure that log messages on different hosts can be merged to accurately reproduce a chronological history of a distributed run.*

For each process, complete these steps:

1. Make sure the host's clock is synchronized with the other hosts.

   Use a time synchronization tool such as Network Time Protocol (NTP).

2. Enable logging to a file instead of standard output by editing `gemfire.properties` to include this line:

   ```
   log-file=filename
   ```

3. Keep the log level at `info` to avoid filling up the disk. Add this line to `gemfire.properties`:

   ```
   log-level=info
   ```

   > *Running with the log level at `fine` impacts system performance and can fill up your disk.*

4. Run the application again.
5. Examine the log files. To get the clearest picture, merge the files.

   To find all the errors in the log file, search for lines that begin with this string:

   ```
   [error
   ```

   For details, see the `merge-logs` command under *Commands* on page 229.

11.2 Diagnosing System Problems

This section provides possible causes and suggested responses for system problems.

Locator Does Not Start

Locator startup fails with an error like this:

```
ERROR: Operation "start-locator" failed because: Start of locator failed.
The end of
"/gemfire/GemFire65/bin/start_locator.log"
contained this message: "[severe 2010/10/14 11:49:49.119 CEST <main>
tid=0x1] Could not start locator
com.gemstone.gemfire.GemFireConfigException: Unable to contact a Locator
service.  Operation either timed out or Locator does not exist.
Configured list of locators is "[192.168.2.1<v0>:41111]". at
com.gemstone.org.jgroups.protocols.TCPGOSSIP.sendGetMembersRequest(TCPGOS
S
IP.java:189) at
com.gemstone.org.jgroups.protocols.PingSender.run(PingSender.java:86) at
java.lang.Thread.run(Thread.java:637) "..
```

This indicates a mismatch somewhere in the address, port pairs used for locator startup and configuration. The address you use for locator startup must match the address you list for the locator in the `gemfire.properties` `locators` specification. Every member of the locator's distributed system, including the locator itself, must have the complete `locators` specification in the `gemfire.properties`.

Response:

- Check that your `locators` specification includes the address you are using to start your locator.
- If you use a bind address, you must use numeric addresses for the locator specification. The bind address will not resolve to the machine's default address.

Application or Cache Server Process Does Not Start

If the process tries to start and then silently disappears, on Windows this indicates a memory problem.

Response:

- On a Windows host, *decrease* the maximum VM heap size. This property is specified on the command line:

  ```
  cacheserver start -J-Xmx1024m
  ```

  For details, see *JVM Memory Settings* on page 141.

- If this doesn't work, try rebooting.

Application or Cache Server Does Not Join the Distributed System

Response: Check these possible causes.

- Network problem—the most common cause. First, try to `ping` the other hosts.
- Firewall problems. If members of your distributed GemFire system are located outside the LAN, check whether the firewall is blocking communication. GemFire Enterprise is a network-centric distributed system, so if you have a firewall running on your machine, it could cause connection problems. For example, your connections may fail if your firewall places restrictions on inbound or outbound permissions for Java-based sockets. You may need to modify your firewall configuration

to permit traffic to Java applications running on your machine. The specific configuration depends on the firewall you are using.

- Wrong multicast port (when using multicast for membership and discovery). Check the `gemfire.properties` file of this application or cache server to see that the `mcast-port` (page 53) is configured correctly. If you are running multiple distributed systems at your site, each distributed system must use a unique multicast port.
- Can't connect to locator (when using TCP for discovery).
  - Check for an error message that includes this string:

```
[severe 2005/10/24 11:21:02.908 PDT nameFromGemfireProperties DownHandler
(FD_SOCK) nid=0xf] GossipClient.getInfo(): exception connecting to host
localhost:30303: java.net.ConnectException: Connection refused
```

    This error means that the application or cache server is configured to connect to a non-existent locator.

  - Check that the `locators` attribute in this process's `gemfire.properties` has the correct IP address for the locator.
  - Check that the locator process is running. If not, see instructions for related problem, "Data Distribution Has Stopped, Though Member Processes Are Running" on page 208.
- Bind address set incorrectly on a multi-homed host. When you specify the bind address, use the IP address rather than the host name. Sometimes multiple network adapters are configured with the same hostname. See *Selecting a Network Adapter Through a Bind Address* on page 69.
- Wrong version of GemFire Enterprise. A version mismatch can cause the process to hang or crash. Check the software version with the `gemfire version` command.
- Bad IP address in the system `hosts` file. Check that the addresses in your `hosts` file are valid. If this is the problem, the failing member's log file may contain a message of this type:

```
com.gemstone.gemfire.ForcedDisconnectException: Attempt to
connect to distributed system timed out
at
com.gemstone.org.jgroups.protocols.pbcast.GMS.down(GMS.java:786)
at . . .
```

- License problems of various types. See the following topics.

Could Not Connect Using a "XXX" License ...

License types are `evaluation`, `development`, and `production`. All members of a distributed system must have the same license type. For details, see *GemFire Licenses* on page 24.

If you try to run a member configured for development in a running, production distributed system, you get this error:

```
Could not connect using a "development" license because the existing
distributed system node "host/port" is using a "production" license.
```

Response:

- Use the `gemfire license` command to find if there really is a mismatch of license types. The command displays details about the GemFire license for the machine on which the command is run. For details, see *GemFire Licenses* on page 24.
- Check that the `license-type` attribute in `gemfire.properties` is correct. This problem often arises because the license was upgraded, but the type wasn't changed in `gemfire.properties`.

Could Not Connect Because the License Has Limited the Number of Distributed System Members to "3".

This indicates that GemFire is using the evaluation license provided in the product download. With the evaluation license you can run a distributed system with up to three members and with three clients for any server you run. The evaluation license never expires.

Response:

- If you do not yet have development or production licenses, to increase your member capacity, follow the instructions in *GemFire Licenses* on page 24.
- If you do have the licenses, then GemFire is not finding them for some reason:
  - Check that you have the right license file in the right location. By default, the process looks for the license file in the current directory and then in the top product directory (also referred to as *`productDir`*).
  - Check that the `license-file` attribute in `gemfire.properties` has the right license name and location.
  - If the `license-file` attribute is correct, check that the process is reading the `gemfire.properties` file. See "Member Process Does Not Read Settings From the gemfire.properties File" on page 201.

Wrong License Version

A "wrong license version" exception means this application or cache server has a different version of GemFire Enterprise from the other members of the distributed system.

A version mismatch often happens at sites with multiple distributed systems, when an application tries to join the wrong system. This happens most often on systems using multicast.

Response:

- Use the `gemfire license` command to find if there is a version mismatch.
- Check the `gemfire.properties` file of this application or cache server to see that the membership attributes are configured correctly. If you do membership and discovery over multicast, look at `mcast-port` on page 53; for TCP, look at `locators` on page 51.
- If you are running multiple distributed systems, confirm that each distributed system is using a unique port number.

License Needs to Be Replaced

If a license you have been using doesn't work any more, the process logs a message similar to this:

```
No valid license is found that matches host and product.
Cause: license sunset date expired: 8/20/05
```

Response:

Check the log file for a "no valid license" exception and note the cause.

- **Evaluation license—**If the message says the license is expired. This applies to versions prior to 6.5:
  - Check that the `license-type` attribute in `gemfire.properties` is correct. Often the problem arises because the license was upgraded but the type wasn't changed in `gemfire.properties`.
  - If you haven't upgraded to a permanent license, follow the procedure explained in *Obtaining and Installing Production and Development Licenses* on page 24.

    Be sure to change the `license-type` *in* `gemfire.properties` *after you upgrade.*

- **Production or development license—**Check whether a change has been made to the host machine, such as a new IP address, hostname, or network card. Any of these could invalidate the license. Follow the instructions in *Obtaining and Installing Production and Development Licenses* on page 24.

Member Process Seems to Hang

Response:

- **During initialization—**For persistent regions, the member may be waiting for another member with more recent data to start and load from its disk stores. See *When Member Startup Hangs* on page 108 of the *GemFire Enterprise System Administrator's Guide*.

  Wait for the initialization to finish or time out. The process could be busy—some caches have millions of entries, and they can take a long time to load. Look for this especially with cache servers, because their regions are typically replicas and therefore store all the entries in the region. Applications, on the other hand, typically store just a subset of the entries.

  For partitioned regions, if the initialization eventually times out and produces an exception, the system architect needs to repartition the data. For information on data partitioning, see Managing Data in Partitioned Regions *on page 178 of the* GemFire Enterprise Developer's Guide.

- **For a running process**—Investigate whether another member is initializing. Under some optional distributed system configurations, a process can be required to wait for a response from other processes before it proceeds.

Member Process Does Not Read Settings From the gemfire.properties File

Either the process can't find the configuration file or, if it is an application, it may be doing programmatic configuration.

Response:

- Check that the `gemfire.properties` file is in the right directory.
- Make sure the process is not picking up settings from another `gemfire.properties` file earlier in the search path. GemFire looks for a `gemfire.properties` file in the current working directory, the home directory, and the `CLASSPATH`, in that order. For details, see *GemFire Configuration Files* on page 42.
- For an application, check the documentation to see whether it does programmatic configuration. If so, the properties that are set programmatically cannot be reset in a `gemfire.properties` file. See your application's customer support group for configuration changes.

Cache Creation Fails - Must Match DOCTYPE Root

System member startup fails with an error like one of these:

```
Exception in thread "main" com.gemstone.gemfire.cache.CacheXmlException:
While reading Cache XML file:/C:/gemfire/client_cache.xml. Error while
parsing XML, caused by org.xml.sax.SAXParseException: Document root
element "client-cache", must match DOCTYPE root "cache".


Exception in thread "main" com.gemstone.gemfire.cache.CacheXmlException:
While reading Cache XML file:/C:/gemfire/cache.xml. Error while parsing
XML, caused by org.xml.sax.SAXParseException: Document root element
"cache", must match DOCTYPE root "client-cache".
```

GemFire declarative cache creation uses one of two DOCTYPE/root element pairs: cache or client-cache. The name must be the same in both places.

Response:

Modify your `cache.xml` file so it has the proper DOCTYPE/root element matching.

For Peers and Servers:

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC
    "-//GemStone Systems, Inc.//GemFire Declarative Caching 6.5//EN"
"http://www.gemstone.com/dtd/cache6_5.dtd">

<cache>
    ...
</cache>
```

For Clients:

```
<?xml version="1.0"?>
<!DOCTYPE client-cache PUBLIC
    "-//GemStone Systems, Inc.//GemFire Declarative Caching 6.5//EN"
"http://www.gemstone.com/dtd/cache6_5.dtd">

<client-cache>
    ...
</client-cache>
```

Cache Isn't Configured Properly

An empty cache can be a normal condition. Some applications start with an empty cache and populate it programmatically, but others are designed to bulk load data during initialization.

Response:

If your application should start with a full cache but it comes up empty, check these possible causes:

- **No Regions—**If the cache has no regions, the process isn't reading the cache configuration file. Check that the name and location of the cache configuration file match those configured in the `cache-xml-file` attribute in `gemfire.properties`. If they match, the process may not be reading `gemfire.properties`. See "Member Process Does Not Read Settings From the gemfire.properties File."
- **Regions Without Data—**If the cache starts with regions, but no data, this process may not have joined the correct distributed system. Check the log file for messages that indicate other members.

If you don't see any, the process may be running alone in its own distributed system. In a process that is clearly part of the correct distributed system, regions without data may indicate an implementation design error. Contact the application's customer support group.

Unexpected Results for keySetOnServer and containsKeyOnServer

Client calls to `keySetOnServer` and `containsKeyOnServer` can return incomplete or inconsistent results if your server regions are not configured as partitioned or replicated regions.

A non-partitioned, non-replicate server region may not hold all data for the distributed region, so these methods would operate on a partial view of the data set.

In addition, the client methods use the least loaded server for each method call, so may use different servers for two calls. If the servers do not have a consistent view in their local data set, responses to client requests will vary.

The consistent view is only guaranteed by configuring the server regions with partitioned or replicate `data-policy` settings. Non-server members of the server system can use any allowable configuration as they are not available to take client requests.

The following server region configurations give inconsistent results. These configurations allow different data on different servers. There is no additional messaging on the servers, so no union of keys across servers or checking other servers for the key in question.

- Normal
- Mix (replicated, normal, empty) for a single distributed region. Inconsistent results depending on which server the client sends the request to

These configurations provide consistent results:

- Partitioned server region
- Replicated server region
- Empty server region: `keySetOnServer` returns the empty set and `containsKeyOnServer` returns `false`

Response: Use a partitioned or replicate `data-policy` for your server regions. This is the only way to provide a consistent view to clients of your server data set. See data-policy *on page 104 of the* GemFire Enterprise Developer's Guide.

Data Operation Returns PartitionOfflineException

In partitioned regions that are persisted to disk, if you have any members offline, the partitioned region will still be available but may have some buckets represented only in offline disk stores. In this case, methods that access the bucket entries return a `PartitionOfflineException`, similar to this:

```
com.gemstone.gemfire.cache.persistence.PartitionOfflineException: Region
/__PR/_B__root_partitioned__region_7 has persistent data that is no
longer online stored at these locations:
[/10.80.10.64:/export/straw3/users/jpearson/bugfix_Apr10/testCL/hostB/bac
kupDirectory created at timestamp 1270834766733 version 0]
```

Response: Bring the missing member online, if possible. This restores the buckets to memory and you can work with them again. If the missing member cannot be brought back online, or the disk stores for the member are corrupt, you may need to revoke the member, which will allow the system to create the buckets in new members and resume operations with the entries. See *Handling Missing Disk Stores* on page 120.

Entries Are Not Being Evicted or Expired as Expected

Check these possible causes.

- **Transactions**—Entries that are old enough for eviction may remain in the cache if they are involved in a transaction. Further, transactions never time out, so if a transaction hangs, the entries involved in the transaction will remain stuck in the cache.

  If you have a process with a hung transaction, you may need to end the process to remove the transaction. In your application programming, do not leave transactions open ended. Program all transactions to end with a commit or a rollback. See *Eviction and Expiration Operation With Transactions* on page 429.

- **Partitioned regions**—For performance reasons, eviction and expiration behave differently in partitioned regions and can cause entries to be removed before you expect. See *Partitioned Region Note for Eviction* on page 182 of the *GemFire Enterprise Developer's Guide* and *Partitioned Region Note for Idle Time Expiration* on page 179 of the *GemFire Enterprise Developer's Guide*.

Can't Find the Log File

Operating without a log file can be a normal condition, so the process does not log a warning.

Response:

- Check whether the `log-file` attribute is configured in `gemfire.properties`. If not, logging defaults to standard output, and on Windows it may not be visible at all.
- If `log-file` is configured correctly, the process may not be reading `gemfire.properties`. See above, "Member Process Does Not Read Settings From the gemfire.properties File" on page 201.

OutOfMemoryError

An application gets an `OutOfMemoryError` if it needs more object memory than the process is able to give. The messages include `java.lang.OutOfMemoryError`.

Response:

The process may be hitting its virtual address space limits. The virtual address space has to be large enough to accommodate the heap, code, data, and dynamic link libraries (DLLs).

- If your application is out of memory frequently, you may want to profile it to determine the cause.
- If you suspect your heap size is set too low, you can increase direct memory by resetting the maximum heap size, using -Xmx. For details, see *JVM Memory Settings* on page 141.
- You may need to lower the thread stack size. The default thread stack size is quite large: 512kb on Sparc and 256kb on Intel for 1.3 and 1.4 32-bit VMs, 1mb with the 64-bit Sparc 1.4 VM; and 128k for 1.2 VMs. If you have thousands of threads then you might be wasting a significant amount of stack space. If this is your problem, the error may be this:

  ```
  OutOfMemoryError: unable to create new native thread
  ```

  The minimum setting in 1.3 and 1.4 is 64kb, and in 1.2 is 32kb. You can change the stack size using the `-Xss` flag, like this: `-Xss64k`
- You can also control memory use by setting entry limits for the regions. For details, see *Memory Overhead Introduced by the Cache API* on page 135, *Keeping Your Data Current With Expiration* on page 176 of the *GemFire Enterprise Developer's Guide*, and *Controlling Memory Use With Eviction and Overflow* on page 182 of the *GemFire Enterprise Developer's Guide*.

For more on controlling memory in Java VMs, see
http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf.

PartitionedRegionDistributionException

The `com.gemstone.gemfire.cache.PartitionedRegionDistributionException` appears when GemFire fails after many attempts to complete a distributed operation. This exception indicates that no data store member can be found to perform a `destroy`, `invalidate`, or `get` operation.

Response:

- Check the network for traffic congestion or a broken connection to a member.
- Look at the overall installation for problems, such as operations at the application level set to a higher priority than the GemFire processes.
- If you keep seeing `PartitionedRegionDistributionException`, you should evaluate whether you need to start more members. See *Managing Resources for Partitioned Regions* on page 137.

PartitionedRegionStorageException

The `com.gemstone.gemfire.cache.PartitionedRegionStorageException` appears when GemFire can't create a new entry. This exception arises from a lack of storage space for `put` and `create` operations or for `get` operations with a loader. `PartitionedRegionStorageException` often indicates data loss or impending data loss.

The text string indicates the cause of the exception, as in these examples:

```
Unable to allocate sufficient stores for a bucket in the partitioned
region....

Ran out of retries attempting to allocate a bucket in the partitioned
region....
```

Response:

- Check the network for traffic congestion or a broken connection to a member.
- Look at the overall installation for problems, such as operations at the application level set to a higher priority than the GemFire processes.
- If you keep seeing `PartitionedRegionStorageException`, you should evaluate whether you need to start more members. See *Managing Resources for Partitioned Regions* on page 137.

Application Crashes Without Producing an Exception

If an application crashes without any exception, this may be caused by an object memory problem. The process is probably hitting its virtual address space limits. For details, see *OutOfMemoryError* on page 205.

Response: Control memory use by setting entry limits for the regions. See *Memory Overhead Introduced by the Cache API* on page 135, *Keeping Your Data Current With Expiration* on page 176 of the *GemFire Enterprise Developer's Guide*, and *Controlling Memory Use With Eviction and Overflow* on page 182 of the *GemFire Enterprise Developer's Guide*.

Timeout Alert

If a distributed message does not get a response within a specified time, it sends an alert to signal that something might be wrong with the system member that hasn't responded. The alert is logged in the sender's log as a warning.

A timeout alert can be considered normal.

Response:

- If you're seeing a lot of timeouts and you haven't seen them before, check whether your network is flooded.
- If you see these alerts constantly during normal operation, consider raising the `ack-wait-threshold` (page 48) above the default 15 seconds.

Member Produces SocketTimeoutException

A client, server, gateway, or gateway hub produces a `SocketTimeoutException` when it stops waiting for a response from the other side of the connection and closes the socket. This exception typically happens on the handshake or when establishing a callback connection.

Response:

Increase the default socket timeout setting for the member. This timeout is set separately for the client `Pool` and for the `Gateway` and `GatewayHub`, either in the `cache.xml` file or through the API. For details on the client/server configuration, see *Socket Timeout* on page 266 of the *GemFire Enterprise Developer's Guide*. For the gateway, see `socket-read-timeout` on page 273 of the *GemFire Enterprise Developer's Guide*.

Member Logs ForcedDisconnectException, Cache and DistributedSystem Forcibly Closed

A distributed system member's `Cache` and `DistributedSystem` are forcibly closed by the system membership coordinator if it becomes sick or too slow to respond to heartbeat requests. When this happens, listeners receive `RegionDestroyed` notification with an opcode of `FORCED_DISCONNECT`. The GemFire log file for the member shows a `ForcedDisconnectException` with the message

```
This member has been forced out of the distributed system because it did
not respond within member-timeout milliseconds
```

Response:

To minimize the chances of this happening, you can increase the `DistributedSystem` property `member-timeout`. Take care, however, as this setting also controls the length of time required to notice a network failure. It should not be set too high.

Members Cannot See Each Other

Suspect a network problem or a problem in the configuration of transport for memory and discovery.

Response:

- Check your network monitoring tools to see whether the network is down or flooded.
- If you are using multi-homed hosts, make sure a bind address is set and consistent for all system members. For details, see *Selecting a Network Adapter Through a Bind Address* on page 69.
- If TCP, check that all the applications and cache servers are using the same locator address.
- If multicast:
  - Check that all the applications and cache servers are using the same multicast IP address and port.
  - Confirm that the multicast IP address and port are a valid combination.
  - Confirm that multicast is enabled on the network. For details, see *Configuring Member Discovery and Communication* on page 59.

Some New Members Are Not Seen By Existing Members

If your application creates many, many short-lived VM's, the system may fail to recognize some new members as they appear. When a member departs the distributed system, GemFire ignores all messages from that member's address for a period of time called the shun sunset. This keeps the system from trying to process a dead member's spurious messages. If you have members joining and using departed member's addresses before the shun sunset has passed, the system will not recognize them.

Response:

Set the shun sunset low enough to allow the system to recognize your new members. The default sunset is 90 seconds. You can change it using the system property `JGroups.SHUN_SUNSET`, which is specified in seconds.

Note that the available pool of "wildcard" ports on Windows is much smaller than on Linux or Solaris, so this problem is more likely to be seen on Windows.

One Part of the Distributed System Cannot See Another Part

This situation can leave your caches in an inconsistent state. In networking circles, this kind of network outage is called the "split brain problem."

Response:

- Restart all the processes to ensure data consistency. For details, see *Recovering From Network Outages* on page 225.
- Going forward, set up network monitoring tools to detect these kinds of outages quickly.
- Enable network partition detection. See *Handling Network Outages* on page 129.

Data Distribution Has Stopped, Though Member Processes Are Running

Suspect a problem with the network, the locator, or the multicast configuration, depending on the transport your distributed system is using.

Response:

- Check the health of your system members. Search the logs for this string:

  `Uncaught exception`

  An uncaught exception means a severe error, often an `OutOfMemoryError`. See *OutOfMemoryError* on page 205
- Check your network monitoring tools to see whether the network is down or flooded.
- If you are using multicast, check whether the existing configuration is no long appropriate for the current network traffic. See *Multicast Health Monitor* on page 164.
- If you are using locators for membership and discovery, check whether the locators have stopped. For a list of the locators in use, check the `locators` property in one of the application `gemfire.properties` files.
  - Restart the locator processes on the same hosts, if possible. The distributed system begins normal operation, and data distribution restarts automatically.
  - If a locator must be moved to another host or a different IP address, complete these steps:

    a. Shut down all the members of the distributed system in the usual order.

    b. Restart the locator process in its new location.

    c. Edit all the `gemfire.properties` files to change this locator's IP address in the `locators` attribute.

    d. Restart the applications and cache servers in the usual order.
- Create a watchdog daemon or service on each locator host to restart the locator process when it stops.

Distributed-ack Operations Take a very Long Time to Complete

This problem can occur in systems with a great number of `distributed-no-ack` operations. That is, the presence of many no-ack operations can cause ack operation to take a long time to complete.

Response:

For information on alleviating this problem, see *Tuning to Reduce Slow distributed-ack Messages* on page 150.

Slow system Performance

Slow system performance is sometimes caused by a buffer size that is too small for the objects being distributed.

Response:

If you are experiencing slow performance and are sending large objects (multiple megabytes), try increasing the socket buffer size settings in your system. For more information, see *Tuning Socket Communication* on page 151.

Can’t Get Windows Performance Data

Attempting to run performance measurements for GemFire Enterprise on Windows can produce this error message:

```
Can't get Windows performance data. RegQueryValueEx returned 5
```

This error can occur because incorrect information is returned when a Win32 application calls the ANSI version of `RegQueryValueEx` Win32 API with `HKEY_PERFORMANCE_DATA`. This error is described in Microsoft KB article ID 226371 at http://support.microsoft.com/kb/226371/en-us.

Response:

To successfully acquire Windows performance data, you need to verify that you have the proper registry key access permissions in the system registry. In particular, make sure that `Perflib` in the following registry path is readable (`KEY_READ` access) by the GemFire process:

```
HKEY_LOCAL_MACHINE\
    SOFTWARE\
    Microsoft\
    Windows NT\
    CurrentVersion\
    Perflib
```

An example of reasonable security on the performance data would be to grant administrators KEY_ALL_ACCESS access and interactive users KEY_READ access. This particular configuration would prevent non-administrator remote users from querying performance data.

See http://support.microsoft.com/kb/310426 and http://support.microsoft.com/kb/146906 for instructions about how to ensure that GemFire processes have access to the registry keys associated with performance.

11.3 System Failure and Recovery

If a system member withdraws from the distributed system involuntarily because the member, host, or network fails, the other members automatically adapt to the loss and continue to operate. The distributed system does not experience any disturbance such as timeouts.

In planning a strategy for data recovery, consider these factors:

- Whether the region is configured for data redundancy—partitioned regions only
- The region's role-loss policy configuration, which controls how the region behaves after a crash or system failure—distributed regions only
- Whether the region is configured for persistence to disk.

  If your processes persist data to disk, before restarting anything be sure you understand the information in *Option for System Member Shutdown Behavior* on page 125.

- The extent of the failure, whether multiple members or a network outage is involved
- Your application's specific needs, such as the difficulty of replacing the data and the risk of running with inconsistent data for your application
- When an alert is generated due to network partition or slow response, indicating that certain processes may, or will, fail

The rest of this chapter provides recovery instructions for various kinds system failures.

Network Partitioning, Slow Response, and Member Removal Alerts

When a network partition detection or slow responses occur, these alerts are generated:

- Network Partitioning is Detected (page 211)
- Member is Taking Too Long to Respond (page 211)
- Warning Notifications Before Removal (page 215)
- Member is Forced Out (page 215)

For information on configuring system members to help avoid a network partition configuration condition in the presence of a network failure or when members lose the ability to communicate to each other, refer to *Handling Network Outages* on page 129.

Network Partitioning is Detected

Alert:

```
Correlated loss of lead member ent(13221):45312/47731 with loss of
coordinator/potential coordinator bilbo(3942:admin):20133/22314
```

Description:

This alert is issued when network partitioning occurs, followed by this alert:

Alert:

```
Membership service failure: Channel closed:
com.gemstone.gemfire.ForcedDisconnectException: Exiting due to possible
network partition event due to loss of member ent(13221):45312/47731
```

Response:

The operator should examine the process to see if it is healthy. For both alerts, the process ID of the slow responder is `13221` on the machine named *ent*. The ports of the slow responder in the first alert are `20133/22314`, and the ports of the slow responder in the second alert are `45312/47331`. For the first alert, look for the string, `Starting distribution manager ent:20133/22314`, and examine the process owning the log file containing this string. For the second alert, look for the string, `Starting distribution manager ent:45312/47731`, and examine the process owning the log file containing this string.

Member is Taking Too Long to Respond

Alert:

```
15 sec have elapsed while waiting for replies: <ReplyProcessor21 6
waiting for 1 replies from [ent(27130):60333/36743]> on
ent(27134):60330/45855 whose current membership list is:
[[ent(27134):60330/45855, ent(27130):60333/36743]]
```

Description:

Member `ent(27130):60333/36743` is in danger of being forced out of the distributed system because of a suspect-verification failure. This alert is issued at the warning level, after the `ack-wait-threshold` is reached.

Response**:**

The operator should examine the process to see if it is healthy. The process ID of the slow responder is `27130` on the machine named `ent`. The ports of the slow responder are `60333/36743`. Look for the string, `Starting distribution manager ent:60333/36743`, and examine the process owning the log file containing this string.

Alert:

```
30 sec have elapsed while waiting for replies: <ReplyProcessor21 6
waiting for 1 replies from [ent(27130):60333/36743]> on
ent(27134):60330/45855 whose current membership list is:
[[ent(27134):60330/45855, ent(27130):60333/36743]]
```

Description:

Member `ent(27134)` is in danger of being forced out of the distributed system because of a suspect-verification failure. This alert is issued at the severe level, after the `ack-wait-threshold` is reached and after `ack-severe-alert-threshold` seconds have elapsed.

Response**:**

The operator should examine the process to see if it is healthy. The process ID of the slow responder is `27134` on the machine named `ent`. The ports of the slow responder are `60333/36743`. Look for the string, `Starting distribution manager ent:60333/36743`, and examine the process owning the log file containing this string.

Alert:

```
15 sec have elapsed while waiting for replies: <DLockRequestProcessor
33636 waiting for 1 replies from [ent(4592):33593/35174]> on
ent(4592):33593/35174 whose current membership list is:
[[ent(4598):33610/37013, ent(4611):33599/60008, ent(4592):33593/35174,
ent(4600):33612/33183, ent(4593):33601/53393, ent(4605):33605/41831]]
```

Description:

This alert is issued by partitioned regions and regions with `global` scope at the warning level, when the lock grantor has not responded to a lock request within the `ack-wait-threshold` and the `ack-severe-alert-threshold`.

Response:

None.

Alert:

```
30 sec have elapsed while waiting for replies: <DLockRequestProcessor
23604 waiting for 1 replies from [ent(4592):33593/35174]> on
ent(4598):33610/37013 whose current membership list is:
[[ent(4598):33610/37013, ent(4611):33599/60008, ent(4592):33593/35174,
ent(4600):33612/33183, ent(4593):33601/53393, ent(4605):33605/41831]]
```

Description:

This alert is issued by partitioned regions and regions with `global` scope at the severe level, when the lock grantor has not responded to a lock request within the `ack-wait-threshold` and the `ack-severe-alert-threshold`.

Response:

None.

Alert:

```
30 sec have elapsed waiting for global region entry lock held by
ent(4600):33612/33183
```

Description

This alert is issued by regions with `global` scope at the severe level, when the lock holder has held the desired lock for `ack-wait-threshold` + `ack-severe-alert-threshold` seconds and may be unresponsive.

Response:

None.

Alert:

```
30 sec have elapsed waiting for partitioned region lock held by
ent(4600):33612/33183
```

Description:

This alert is issued by partitioned regions at the severe level, when the lock holder has held the desired lock for `ack-wait-threshold` + `ack-severe-alert-threshold` seconds and may be unresponsive.

Response:

None.

No Locators Can Be Found

It is likely that all processes using the locators will exit with the same message.

Alert:

```
Membership service failure: Channel closed:
com.gemstone.gemfire.ForcedDisconnectException: There are no processes
eligible to be group membership coordinator (last coordinator left view)
```

Description:

Network partition detection is enabled (`enable-network-partition-detection` is set to `true`), and there are locator problems.

Response**:**

The operator should examine the locator processes and logs, and restart the locators.

Alert:

```
Membership service failure: Channel closed:
com.gemstone.gemfire.ForcedDisconnectException: There are no processes
eligible to be group membership coordinator (all eligible coordinators
are suspect)
```

Description:

Network partition detection is enabled (`enable-network-partition-detection` is set to `true`), and there are locator problems.

Response:

The operator should examine the locator processes and logs, and restart the locators.

Alert:

```
Membership service failure: Channel closed:
com.gemstone.gemfire.ForcedDisconnectException: Unable to contact any
locators and network partition detection is enabled
```

Description:

Network partition detection is enabled (`enable-network-partition-detection` is set to `true`), and there are locator problems.

Response:

The operator should examine the locator processes and logs, and restart the locators.

Alert:

```
Membership service failure: Channel closed:
com.gemstone.gemfire.ForcedDisconnectException: Disconnected as a slow-
receiver
```

Description:

The member was not able to process messages fast enough and was forcibly disconnected by another process.

Response:

The operator should examine and restart the disconnected process.

Warning Notifications Before Removal

Alert:

```
Membership: requesting removal of ent(10344):21344/24922 Disconnected as
a slow-receiver
```

Description:

This alert is generated only if the `slow-receiver` functionality is being used.

Response:

The operator should examine the locator processes and logs.

Alert:

```
Network partition detection is enabled and both membership coordinator
and lead member are on the same machine
```

Description:

This alert is issued if both the membership coordinator and the lead member are on the same machine.

Response:

The operator can turn this off by setting the system property `gemfire.disable-same-machine-warnings` to true. However, it is best to run locator processes, which act as membership coordinators when network partition detection is enabled, on separate machines from cache processes.

Member is Forced Out

Alert:

```
Membership service failure: Channel closed:
com.gemstone.gemfire.ForcedDisconnectException: This member has been
forced out of the Distributed System. Please consult GemFire logs to find
the reason.
```

Description:

The process discovered that it was not in the distributed system and cannot determine why it was removed. The membership coordinator removed the member after it failed to respond to an internal `are you alive` message.

Response:

The operator should examine the locator processes and logs.

11.4 Recovering From Application or Cache Server Crashes

When the application or cache server crashes, its local cache is lost, and any resources it owned (for example, distributed locks) are released. The member must recreate its local cache upon recovery.

This section covers the following:

Recovery in a Peer-to-Peer Configuration

When a member crashes, the other system members are told that it has left unexpectedly. The remaining members continue operation as though the missing application or cache server had never existed. If any remaining system member is waiting for a response (ACK), the ACK still succeeds and returns, because every member that is still alive has responded. (Configuring membership roles can change this behavior—for details, see *Managing Member Relationships* on page 453 of the *GemFire Enterprise Developer's Guide*.) If the lost member had ownership of a `GLOBAL` entry, then the next attempt to obtain that ownership acts as if no owner exists.

Recovery depends on how the member has its cache configured. This section covers the following:

To tell whether the regions are partitioned, distributed, or local, check the `cache.xml` file. If the file contains a local scope setting, the region has no connection to any other member:

```
<region-attributes scope="local">
```

If the file contains any other scope setting, it's configuring a distributed region. For example:

```
<region-attributes scope="distributed-no-ack">
```

If the file includes either of the following lines, it's configuring a partitioned region.

```
<partition-attributes...

<region-attributes data-policy="partition"/>

<region-attributes data-policy="persistent-partition"/>
```

The reassigned clients continue operating smoothly, as in the failover case. A successful rebalancing operation does not create any data loss.

If rebalancing fails, the client fails over to an active server with the normal failover behavior.

Recovery for Partitioned Regions

When an application or cache server crashes, any data in local memory is lost, including any entries in a local partitioned region data host.

Recovery with Data Redundancy

If the partitioned region is configured for redundancy and a member crashes, the system continues to operate smoothly with the remaining copies of the data. You do not need to take immediate action for recovery, as long as you still have at least two functioning members for each partitioned region.

By default, GemFire does not make new copies of the data until a new member is brought online to replace the member that crashed. You can control this behavior using the recovery delay attributes. For more information, see *Rebalancing Partitions* on page 189 of the *GemFire Enterprise Developer's Guide*.

To recover, start a replacement member. The new member regenerates the lost copies and returns them to the configured redundancy level. See *Adding an Extra Partitioned Region Data Host at Run Time* on page 137.

> *Make sure the replacement member has at least as much local memory as the old one—local-max-memory must be the same or larger. Otherwise, you can get into a situation where some entries have all their redundant copies but others don't.*

Even with high availability, you can lose data if too many applications and cache servers fail at the same time. Any lost data is replaced with new data created by the application as it returns to active work.

The number of members that can fail without losing data is equal to the number of redundant copies configured for the region. So if `redundant-copies=1`, then at any given time one member can be down without data loss. For more information, see Planning for Enough Members to Support Redundancy *on page 191 of the* GemFire Enterprise Developer's Guide.

You can also lose access to all copies of your data through network failure. See *Recovering From Network Outages* on page 225.

Recovery without Data Redundancy

If a member crashes and there are no redundant copies, the data in that member is lost. The other members in the partitioned region continue operation. Although the data is lost, the partitioned region does not experience any disturbance such as timeouts.

To recover, restart the member. The application returns to active work and automatically begins to create new data.

Maintaining and Recovering Partitioned Region Redundancy

The following alert [ALERT-1] (warning) is generated when redundancy for a partitioned region drops:

Alert:

```
[warning 2008/08/26 17:57:01.679 PDT dataStoregemfire5_jade1d_6424
<PartitionedRegion Message Processor2> tid=0x5c] Redundancy has dropped
below 3 configured copies to 2 actual copies for /partitionedRegion

[warning 2008/08/26 18:13:09.059 PDT dataStoregemfire5_jade1d_6424 <DM-
MemberEventInvoker> tid=0x1d5] Redundancy has dropped below 3 configured
copies to 1 actual copy for /partitionedRegion
```

The following alert [ALERT-2] (warning) is generated when, after creation of a partitioned region bucket, the program is unable to find enough members to host the configured redundant copies:

Alert:

```
[warning 2008/08/27 17:39:28.876 PDT gemfire_2_4 <RMI TCP Connection(67)-
10.80.250.201> tid=0x1786] Unable to find sufficient members to host a
bucket in the partitioned region. Region name = /partitionedregion
Current number of available data stores: 1 number successfully allocated
= 1 number needed = 2 Data stores available: [pippin(21944):41927/42712]
Data stores successfully allocated: [pippin(21944):41927/42712] Consider
starting another member
```

The following alert [EXCEPTION-1] (warning) and exception is generated when, after the creation of a partitioned region bucket, the program is unable to find any members to host the primary copy:

Alert:

```
[warning 2008/08/27 17:39:23.628 PDT gemfire_2_4 <RMI TCP Connection(66)-
10.80.250.201> tid=0x1888] Unable to find any members to host a bucket in
the partitioned region. Region name = /partitionedregion Current number
of available data stores: 0 number successfully allocated = 0 number
needed = 2 Data stores available: [] Data stores successfully allocated:
[] Consider starting another member
```

Exception:

```
com.gemstone.gemfire.cache.PartitionedRegionStorageException: Unable to
find any members to host a bucket in the partitioned region.
```

- Region name = `/partitionedregion`
- Current number of available data stores: 0
- Number successfully allocated = 0; Number needed = 2
- Data stores available: `[]`
- Data stores successfully allocated: `[]`

Response:

- Add additional members configured as data hosts for the partitioned region.
- Consider starting another member.

Recovery for Distributed Regions

For distributed regions, GemFire Enterprise has a facility for maintaining data in memory in another member. Cache servers' regions are usually, but not always, replicas, which store all of the data associated with a region. Other members may store only selected data. When members crash or exit, any replicas left running still have the full data set. When you restart an application, it can retrieve all its data automatically through the distributed system, as long as all its regions are replicas. For more information, see Chapter 8, *Replicate and Other Distributed Regions*, on page 197.

Regions can also persist data to disk. If the region is configured for persistence, the data remains in the region's disk directories after a crash. Then if the same region is created again with comparable attributes and there is no replicate to use for recovery, the persisted data on disk is used to initialize the region.

Restart the process. The system member recreates its cache automatically. If replication is set up, data is automatically loaded from the replicas, creating an up-to-date cache in sync with the rest of the system. If you have persisted data but no replicas, data is automatically loaded from the disk store files. Otherwise, the lost data is replaced with new data created by the application as it returns to active work.

Recovery for Regions of Local Scope

Regions of local scope have no memory backup, but may have data persisted to disk. If the region is configured for persistence, the data remains in the region's disk directories after a crash. The data on disk will be used to initialize the region when you restart.

Recovering Data From Disk

When you persist a region, the entry data on disk outlives the region in memory. If the member exits or crashes, the data remains in the region's disk directories—see *GemFire Output Files* on page 37. If the same region is created again, this saved disk data can be used to initialize the region.

Some general considerations for disk data recovery:

- Region persistence causes only entry keys and values to be stored to disk. Statistics and user attributes are not stored.
- If the application was writing to the disk asynchronously, the chances of data loss are greater. The choice is made at the region level, with the `disk-synchronous` attribute. The default is false, making writes asynchronous.
- When a region is initialized from disk, all recovered data is considered new at the time it is loaded from disk. The statistics for last modified time and last accessed time are set to the time of recovery. For information on how this might affect the region data, see *Keeping Your Data Current With Expiration* on page 176 of the *GemFire Enterprise Developer's Guide*.

For more information, see Chapter 6, *Managing Disk Stores*, on page 95.

Disk Recovery for Disk Writing—Synchronous Mode and Asynchronous Mode

Synchronous Mode of Disk Writing

Alert 1:

```
DiskAccessException has occured while writing to the disk for region
<Region_Name>. Attempt will be made to destroy the region locally.;
```

Alert 2:

```
Encountered Exception in destroying the region locally
```

Description:

These are error log-level alerts. Alert 2 is generated only if there was an error in destroying the region. If Alert 2 is not generated, then the region was destroyed successfully. The message indicating the successful destruction of a region is logged at the information level.

Alert 3:

```
Problem in stopping Cache Servers. Failover of clients is suspect
```

Description:

This is an error log-level alert that is generated only if servers were supposed to stop but encountered an exception that prevented them from stopping.

Response:

The user must be aware that the region may no longer exist on the node. The cache servers and gateway hubs may also have been stopped. The user needs to recreate the region and restart the cache servers.

Asynchronous Mode of Disk Writing

Alert 1:

```
Problem in Asynch writer thread for region <Region_name>. It will
terminate.
```

Alert 2:

```
Encountered Exception in destroying the region locally
```

Description:

These are error log-level alerts. Alert 2 is generated only if there was an error in destroying the region. If Alert 2 is not generated, then the region was destroyed successfully. The message indicating the successful destruction of a region is logged at the information level.

Alert 3:

```
Problem in stopping Cache Servers. Failover of clients is suspect
```

Description:

This is an error log-level alert that is generated only if servers were supposed to stop but encountered an exception that prevented them from stopping.

Response:

The user must be aware that the region may no longer exist on the node. The cache servers and gateway hubs may also have been stopped. The user needs to recreate the region and restart the cache servers.

Recovery in a Client/Server Configuration

For the client/server configuration, recovery from application or cache server crashes depends on the available servers and on client configuration. Normally, the servers are made highly available by running enough servers spread out on enough machines to ensure a minimum of coverage in case of network, machine, or server crashes. The clients are usually configured to connect to a primary and some number of secondary, or redundant, servers. The secondaries act as hot backups to the primary. For information on this, see *Highly Available Subscription Delivery* on page 241 of the *GemFire Enterprise Developer's Guide*. To cover in case of a client crash, the clients may have durable connections to their servers. If this is the case, some or all of their data and data events remain in server memory and are automatically recovered, providing that you restart the clients within a configured timeout. See *Durable Subscription Queues* on page 246 of the *GemFire Enterprise Developer's Guide*.

Recovering From Server Failure

Recovery from server failure has two parts: the server recovers as a member of a distributed system and then its clients recover its services.

When servers fail, their own recovery is carried out as for any member of a distributed system as described in Recovery in a Peer-to-Peer Configuration (page 216).

From the client's perspective, if the system is configured for high availability, server failure goes undetected unless enough servers fail that the server-to-client ratio drops below a workable level. In any case, your first course of action is to get the servers back up as quickly as possible.

To recover from server failure:

1. Recover the server and its data as described in Recovery in a Peer-to-Peer Configuration (page 216).
2. Once the server is available again, the locators (or client pools if you are using a static server list) automatically detect its presence and add it to the list of viable servers. It might take awhile for the clients to start using the recovered server. The time depends in part on how the clients are configured and how they are programmed. For information on client configuration, see *Client/Server Architecture and Configuration Basics* on page 201 of the *GemFire Enterprise Developer's Guide*.

If You Need to Start a Server at a New Host/Port Location

This section is only for systems where the clients' server pool configurations use static server lists. If the server pools are configured with locator lists, starting a server at a new address requires no special action because the new server is automatically detected by the locators. You can determine whether your clients use locator lists or server lists by looking at the client `cache.xml` files. Systems configured with static server lists have `<server>` elements listed inside the `<pool>` elements. Those using locator lists have `<locator>` elements instead. If there are no pools declared in the XML files, the servers or locators will be defined in the application code. Look for the API `PoolFactory` methods `addServer` or `addLocator`.

If the pools are configured with static server lists, the clients only connect to servers at the specific addresses provided in the lists. To move a server or add a server at a new location, you must modify the `<server>` specifications in the clients' `cache.xml` file. This change will only affect newly-started clients. To start using the new server information, either restart clients or wait for new clients to start, depending on your system characteristics and how quickly you need the changes to take effect.

Recovering From Client Failure

When a client crashes, restart it as quickly as possible in the usual way. The client recovers its data from its servers through normal operation. Some of the data may be recovered immediately, and some may be recovered lazily as the client requests it. Additionally, the server may be configured to replay events for some data and for some client queries. These are the different configurations that affect client recovery:

- **Entries immediately sent to the client**—Entries are immediately sent to the client for entries the client registers interest in, if those entries are present in the server cache.
- **Entries sent lazily to the client**—Entries are sent lazily to the client for entries that the client registers interest in that are not initially available in the server cache.
- **Events sent immediately to the client**—If the server has been saving events for the client, these are immediately replayed when the client reconnects. There are two types of events saved, cache modification events for entries in which the client has registered durable interest and query modification events for continuous queries that the client has created as durable queries.

If you need to sort out which update processes apply to your client, go to the *Developer's Guide* for details. You have to check a few configuration parameters, which can be set in various ways, on both the client and the server. For interest registration, see *Server-Initiated Data Flow* on page 219 of the

GemFire Enterprise Developer's Guide. For durable clients, see *Durable Subscription Queues* on page 246 of the *GemFire Enterprise Developer's Guide*.

If you have a durable client configured to connect to multiple servers, keep in mind that GemFire does not maintain server redundancy while the client is disconnected. If you lose all of its primary and secondary servers, you lose the client's queued messages. Even if the servers fail one at a time, so that running clients have time to fail over and pick new secondary servers, an off-line durable client cannot do that and thus loses its queued messages.

11.5 Recovering From Machine Crashes

When a machine crashes because of a shutdown, power loss, hardware failure, or operating system failure, all of its applications and cache servers and their local caches are lost. System members on other machines are notified that this machine's members have left the distributed system unexpectedly.

To recover:

1. Determine which processes run on this machine.
2. Reboot the machine.
3. If a GemFire locator runs here, start it first.

   At least one locator must be running before you start any applications or cache servers.

4. Start the applications and cache servers in the usual order.

   If you have disk store files, before restarting anything be sure you understand the information in *Option for System Member Shutdown Behavior* on page 125.

If you have to move a locator process to a different machine, the locator isn't useful until you update the locators list in the `gemfire.properties` file and restart all the applications and cache servers in the distributed system. If other locators are running, however, you don't have to restart the system immediately. For a list of the locators in use, check the `locators` property in one of the application `gemfire.properties` files.

Data Recovery for Partitioned Regions

The partitioned region initializes itself correctly regardless of the order in which the data hosts rejoin. The applications and cache servers recreate their data automatically as they return to active work.

If the partitioned region is configured for data redundancy, GemFire Enterprise may be able to handle a machine crash automatically with no data loss, depending on how many redundant copies there are and how many members have to be restarted. To estimate the number of concurrent member failures your installation can handle, see *Planning for Enough Members to Support Redundancy* on page 191 of the *GemFire Enterprise Developer's Guide*. See also *Recovery with Data Redundancy* on page 217.

If the partitioned region does not have redundant copies, the system members recreate the data through normal operation. If the member that crashed was an application, check whether it was designed to write its data to an external data source. If so, decide whether data recovery is possible and preferable to starting with new data generated through the GemFire distributed system.

Data Recovery for Distributed Regions

The applications and cache servers recreate their data automatically. Recovery happens through replicas, disk store files, or newly generated data, as explained in *Recovery for Distributed Regions* on page 219.

If the recovery is from disk stores, you may not get all of the latest data. Persistence depends on the operating system to write data to the disk, so when the machine or operating system fails unexpectedly, the last changes can be lost.

For maximum data protection, you can set up duplicate replicate regions on the network, with each one configured to back up its data to disk. Assuming the proper restart sequence, this architecture significantly increases your chances of recovering every update.

Data Recovery in a Client/Server Configuration

If the machine that crashed hosted a server, how the server recovers its data depends on whether the regions are partitioned or distributed. See *Data Recovery for Partitioned Regions* on page 223 or *Data Recovery for Distributed Regions* on page 223, as appropriate.

The impact of a server crash on its clients depends on whether the installation is configured for highly available servers. If you have to move a server to a different machine, you need to change its address on all of the related clients. For information on both issues, see *Recovery in a Client/Server Configuration* on page 220.

If the machine that crashed hosted a client, restart the client as quickly as possible and let it recover its data automatically from the server. For details, see *Recovering From Client Failure* on page 221.

11.6 Recovering From Network Outages

When the network connecting members of a distributed system goes down, system members treat this like a machine crash. Members on each side of the network failure respond by removing the members on the other side from the membership list. If network partitioning detection is enabled, one partition containing the lead member and a locator will continue to run, and other data hosts will shut down; otherwise, the system will behave is explained below.

Effect of Network Failure on Partitioned Regions

Both sides of the distributed system continue to run as though the members on the other side were not running. If the members that participate in a partitioned region are on both sides of the network failure, both sides of the partitioned region also continue to run as though the data hosts on the other side did not exist. In effect, you now have two partitioned regions.

When the network recovers, the members may be able to see each other again, but they are not able to merge back together into a single distributed system and combine their buckets back into a single partitioned region. You can be sure that the data is in an inconsistent state. Whether you are configured for data redundancy or not, you don't really know what data was lost and what wasn't. Even if you have redundant copies and they survived, different copies of an entry may have different values reflecting the interrupted workflow and inaccessible data.

Effect of Network Failure on Distributed Regions

By default, both sides of the distributed system continue to run as though the members on the other side were not running. For distributed regions, however, the regions's reliability policy configuration can change this default behavior. For details, see *Managing Member Relationships* on page 453 of the *GemFire Enterprise Developer's Guide*.

When the network recovers, the members may be able to see each other again, but they are not able to merge back together into a single distributed system.

Effect of Network Failure on Client/Server Installations

If a client loses contact with all of its servers, the effect is the same as if it had crashed. You need to restart the client. See *Recovering From Client Failure* on page 221. If a client loses contact with some servers, but not all of them, the effect on the client is the same as if the unreachable servers had crashed. See *Recovering From Server Failure* on page 221.

Servers, like applications, are members of a distributed system, so the effect of network failure on a server is the same as for an application. Exactly what happens depends on the configuration of your site. For recovery strategies, see the *Recovery*.

Recovery

The safest response is to restart all the processes and bring up a fresh data set. However, if you know the architecture of your system well, and you are sure you won't be resurrecting old data, you can do a selective restart. At the very least, you must restart all the members on one side of the network failure, because a network outage causes separate distributed systems that can't rejoin automatically.

To recover:

1. Decide which applications and cache servers to restart, based on the architecture of the distributed system.

   Assume that any process other than a data source is bad and needs restarting. For example, if an outside data feed is coming in to one member, which then redistributes to all the others, you can leave that process running and restart the other members.

2. Shut down all the processes that need restarting.

3. Restart them in the usual order.

   > *If you have disk stores, before restarting anything be sure you understand the information in* *Option for System Member Shutdown Behavior* on page 125.

   For partitioned regions, the startup order depends on the architecture and requirements of the applications and cache servers, not the partitioned region. The partitioned region can initialize itself correctly, regardless of the order in which the host members rejoin.

The members recreate the data as they return to active work. For details, see *Recovering From Application or Cache Server Crashes* on page 216.

Appendix

A The gemfire Command-line Utility

This appendix provides syntax and other reference information for the `gemfire` command-line utility, which allows you to view product version and licensing information, merge log files, print information from statistic files, and manage GemFire locators from an operating system command prompt.

The `gemfire` utility allows you to perform basic administration tasks from a script. However, all GemFire Enterprise® administrative operations must be executed on the same machine as the GemFire process, and only apply to a single gemfire process at a time.

A.1 Usage

At an operating system prompt, enter this command line:

```
gemfire [-debug] [-h[elp]] [-q] [-J<vmOpt>]* command|help ...
```

On Windows, you can display a command-line prompt from the Start menu by pointing to Programs, pointing to Accessories, then clicking Command Prompt.

This table describes the `gemfire` command-line options.

Table A.1 gemfire Command-line Options

Option	Description
`-debug`	Causes `gemfire` to log extra information when it fails.
`-h or -help`	Prints out help information for a command. For example, to display help information about `gemfire config`, issue the following command: `gemfire -h config`
`-q`	Provides quiet operation by suppressing extra messages.
`-J<vmOpt>`	JVM option for the command.

`command`	Specifies the operation to perform:
	`encrypt-password`
	`info-locator`
	`license`
	`merge-logs`
	`start-locator`
	`stats`
	`status-locator`
	`stop-locator`
	`tail-locator-log`
	`version`
	
	`encrypt-password`
	`help`
	`info-locator`
	`license`
	`merge-logs`
	`start-locator`
	`stats`
	`status-locator`
	`stop-locator`
	`tail-locator-log`
	`version`
	
	`revoke-missing-disk-store`
	`list-missing-disk-stores`
	`validate-disk-store`
	`compact-disk-store`
	`compact-all-disk-stores`
	`modify-disk-store`
	`shut-down-all`
	`backup`

A.2 Commands

```
gemfire [-debug] [-h[elp]] [-q] command ...
```

The `gemfire` command requires one of the *`command`* strings listed in the table below.

In the command descriptions, the following syntax is used:

`courier` designates literal text.

[ ] designate an optional item.

( ) are used to group items.

`italics` designate non-literal text—used to designate logical items.

* suffix means zero or more of the previous item.

| indicates one of several mutually-exclusive options.

Table A.2 gemfire Commands

Command	Description
`backup`	`gemfire backup <target directory>` Connects to a running system and asks all its members that have persistent data to back it up to the specified directory. The directory specified must exist on all members, but it can be a local directory on each machine. This command uses the `gemfire.properties` file for the distributed system specification. For more information, see Chapter 6, *Managing Disk Stores*, on page 95.
`compact-all_disk-stores`	`gemfire compact-all-disk-stores` Connects to a running system and tells its members to compact their disk stores. This command uses the compaction threshold that each member has configured for its disk stores. A disk store must have `allow-force-compaction` set to `true` to be compacted through this command. For more information, see Chapter 6, *Managing Disk Stores*, on page 95.
`compact-disk-store`	`gemfire compact-disk-store <diskStoreName> <directory>+ [-maxOplogSize=<int>]` Compacts an offline disk store. Compaction removes all unneeded records from the persistent files. Provide the disk store name and all its directories. `-maxOplogSize=<long>` causes the oplogs created by compaction to be no larger than the specified size in megabytes. For more information, see Chapter 6, *Managing Disk Stores*, on page 95.
`encrypt-password`	`gemfire encrypt-password passwordString` Encrypts the password provided and prints the encrypted password to standard output. This encrypted password is used in data source connections for transactions. For more information, see *Encrypting Connection Passwords* on page 64 of the *GemFire Enterprise Developer's Guide*.
`help`	`gemfire help [all | overview | commands | options | usage | configuration]` Prints information on how to use this tool. If you specify an optional help topic, then more detailed help is printed.

Command	Description
`info-locator`	`gemfire info-locator [-dir=locatorDir]` Prints information on a locator, including the locator's process ID. The `-dir` option specifies the directory of the locator whose information is desired. For details about locators, see *Configuring Member Discovery and Communication* on page 59.
`license`	`gemfire license [-file=licenseFile]` The `-file` option specifies a specific license file to print. Without this parameter, the utility uses the default license file and prints its location and contents. Displays details about the GemFire license for the machine on which the command is run. For details, see *GemFire Licenses* on page 24.
`list-missing-disk-stores`	`gemfire list-missing-disk-stores` Prints out a description of the disk stores that are currently missing from a distributed system. This command uses the `gemfire.properties` file for the distributed system specification. For more information, see Chapter 6, *Managing Disk Stores*, on page 95.
`merge-logs`	`gemfire merge-logs <logFile>+ [-out=outFile]` Merges the specified logs into a single log. The `-out` option specifies the output file for the merged log. By default, the merged file is sent to standard output.
`modify-disk-store`	`gemfire modify-disk-store <diskStoreName> <directory>+ [-region=<regionName> [-remove|(-lru=<none|lru-entry-count|lru-heap-percentage|lru-memory-size>|-lruAction=<none|overflow-to-disk|local-destroy>|-lruLimit=<int>|-concurrencyLevel=<int>|-initialCapacity=<int>|-loadFactor=<float>|-statisticsEnabled=<boolean>)*]] <diskStoreName> <directory>+ [-maxOplogSize=<int>]` Modifies an offline disk store. Use to remove a region from a disk store or to modify its load and memory control attributes. Provide the disk store name and all its directories. Provide the region name that you want to change. Then specify either `-remove` to take the region out of the disk store, or one or more of the region attribute switches to change attribute settings. For more information, see Chapter 6, *Managing Disk Stores*, on page 95.
`revoke-missing-disk-store`	`gemfire revoke-missing-disk-store <address> <directory>` Connects to a running system and tells its members to stop waiting for the specified disk store to be available. Only revoke a disk store if its files are lost. Once a disk store is revoked its files can no longer be loaded, so be careful. Use the `list-missing-disk-stores` command to get address and directory of the missing disk store, to pass to this `revoke` command. If the disk store was spread across multiple directories, specify the first directory in the list. This command uses the `gemfire.properties` file for the distributed system specification. For more information, see Chapter 6, *Managing Disk Stores*, on page 95.

Command	Description
`shut-down-all`	`gemfire shut-down-all` Connects to a running system and tells its members to shut down in an orderly fastion. Persistent partitioned regions bring themselves in sync before shutting down, which speeds startup the next time. This command uses the `gemfire.properties` file for the distributed system specification. For more information on the effects on startup of this kind of shutdown, see Chapter 6, *Managing Disk Stores*, on page 95.
`start-locator`	`gemfire start-locator [-port=port] [-address=ipAddr] [-dir=locatorDir] [-peer=<true|false>] [-server=<true|false>] [-hostname-for-clients=<ipAddr>] [-properties=gemfire.properties.file] [-DsystemPropertyName=value]* [-Xoption=value]*` Starts a locator. For details about locators, see *Configuring Member Discovery and Communication* on page 59. The `-port` option specifies the port on which the locator listens (by default, `10334`). Valid values are in the range `0..65535`. The `-address` option specifies the IP address on which the locator listens. By default, the locator listens on the default card for the machine. (Also see the discussion of the `bind-address` (page 49) configuration attribute.) The `-dir` option specifies the directory in which the locator runs. The `-peer` option indicates whether the locator acts as a peer locator service for members of its own distributed system. The default is `true`. The `-server` option indicates whether the locator acts as a server locator service for clients to its distributed system. The default is `true`. The `-hostname-for-clients` option specifies a host name or IP address that is sent to clients for connecting to the locator. The default is the address on which the locator is listening. The `-properties` option specifies the `gemfire.properties` file to use for configuring the locator's distributed system. The file's path should be absolute, or relative to the locator's directory, specified with the `-dir` option. The `-D` option allows you to provide the locator with a java system property from the command line. Any number of system properties may be specified. The `-X` option allows you to set a vendor-specific VM option. It is usually used to increase the size of the locator VM when using multicast. Any number of vendor-specific options can be specified.

Command	Description

`stats`	`gemfire stats ([instanceId][:typeId][.statId])* -archive=statFile [-details] [-nofilter|-persec|-persample] [-prunezeros] [-starttime=time] [-endtime=time]`
	Prints statistic values from a statistic archive.
	By default all statistics are printed. The `statSpec` arguments can be used to print individual resources or a specific statistic.
	The format of a `statSpec` is (in order): an optional combine operator, an optional *`instanceId`*, an optional *`typeId`*, an optional *`statId`*.
	The combine operator can be a plus (`+`) to combine all matches in the same file or double plus (`++`) to combine all matches across all files.
	The *`instanceId`* must be the name or id of a resource.
	The *`typeId`* is a colon (`:`) followed by the name of a resource type.
	The *`statId`* is a period (`.`) followed by the name of a statistic.
	A *`typeId`* or *`instanceId`* with no *`statId`* prints out all the matching resources and all their statistics.
	A *`typeId`* or *`instanceId`* with a *`statId`* prints out just the named statistic on the matching resources.
	A *`statId`* with no *`typeId`* or *`instanceId`* matches all statistics with that name.
	The `-archive` option specifies the archive file to use.
	The `-details` option causes statistic descriptions to also be printed.
	The `-nofilter` option, in conjunction with `-archive`, causes all printed statistics to be raw, unfiltered, values.
	The `-persec` option, in conjunction with `-archive`, causes the printed statistics to be the rate of change, per second, of the raw values.
	The `-persample` option, in conjunction with `-archive`, causes the printed statistics to be the rate of change, per sample, of the raw values.
	The `-prunezeros` option, in conjunction with `-archive`, suppresses the printing of statistics whose values are all zero.
	The `-starttime` option, in conjunction with `-archive`, causes statistics samples taken before the specified time to be ignored. The argument format must match this string:
	`yyyy/MM/dd HH:mm:ss.SSS z`
	where `z` is the time zone.
	The `-endtime` option, in conjunction with `-archive`, causes statistics samples taken after the specified time to be ignored. The argument format must match this string:
	`yyyy/MM/dd HH:mm:ss.SSS z`

`status-locator`	`gemfire status-locator [-dir=locatorDir]`
	Prints the status of a locator. The status string is one of the following:
	`stopped`
	`stopping`
	`killed`
	`starting`
	`running`
	The `-dir` option specifies the directory of the locator whose status you want to obtain.
	For details about locators, see *Selecting a Network Adapter Through a Bind Address* on page 69.

Command	Description
`stop-locator`	`gemfire stop-locator [-port=port] [-address=ipAddr] [-dir=locatorDir]` Stops a locator. For details about locators, see *Selecting a Network Adapter Through a Bind Address* on page 69. The `-port` option specifies the port that the locator is listening on (by default, `10334`). The `-addr` option specifies the IP address on which the locator is listening. By default, the locator listens on the default card for the machine. The `-dir` option specifies the directory in which the locator is running.
`tail-locator-log`	`gemfire tail-locator-log [-dir=locatorDir]` Prints out the tail end of the locator's log. The `-dir` option specifies the directory in which the locator is running.
`validate-disk-store`	`gemfire validate-disk-store <diskStoreName> <directory>+` Checks to make sure files of an offline disk store are valid. The name of the disk store and the directories its files are stored in are required arguments. For more information, see Chapter 6, *Managing Disk Stores*, on page 95.
`version`	`gemfire version` Prints GemFire product version information.

Appendix

B System Statistics

This appendix provides information on the GemFire Enterprise installation standard statistics for caching and distribution activities. The API for creating application-defined statistics and the `Region` and `Entry` statistics available from the cache are discussed in *Statistics* on page 464 of the *GemFire Enterprise Developer's Guide*.

In this appendix:

B.1 Configuring Statistics

When Java applications and cache servers join a distributed system, they indicate whether to enable statistics sampling and whether to archive the statistics that are gathered.

> GemFire statistics use the Java `System.nanoTimer` for nanosecond timing. This method provides nanosecond precision, but not necessarily nanosecond accuracy. For more information, see the online Java documentation for `System.nanoTimer` for the JRE you are using with GemFire.

This table lists all of the statistics-related properties. For performance reasons, all statistics sampling is disabled by default. For detailed information about setting these properties see *System Properties in the gemfire.properties File* on page 48.

Table B.1 Statistics Configuration Properties

Property	Purpose
statistic-sampling-enabled (page 58)	Enables statistics gathering. None of the other properties matter if this is not `true`.
enable-time-statistics (page 51)	Enables time-based statistics, which are disabled by default for better performance.
statistic-archive-file (page 57)	Enables archiving of statistics. The name of the archive file.
archive-file-size-limit (page 48)	Limit on the size of a single archive file.
archive-disk-space-limit (page 48)	Limit on the total size of the archive files.
statistic-sample-rate (page 58)	How often to take samples.

To enable time-based statistics, your gemfire.properties file must at least have these lines:

```
statistic-sampling-enabled=true
enable-time-statistics=true
```

If you want to archive the statistics, the file must also have:

```
statistic-archive-file=<your file name>
```

Examining Archived Statistics

When sampling and archiving are enabled, you can study statistics in archive files through VSD or by using the GemFire `stats` command. You can use VSD to examine archived historical data and to help diagnose performance problems. The VSD tool reads the sampled statistics and produces graphical displays for analysis.

Controlling the Size of Archive Files

You can specify limits on the archive files for statistics. These are the areas of control:

Archive File Growth Rate

- The `statistic-sample-rate` (page 58) controls the speed at which the archive file grows.
- If the designated `statistic-archive-file` (page 57) has the `.gz` suffix, it is compressed, thereby taking up less disk space.

Maximum Size of a Single Archive File

If the value of `archive-file-size-limit` (page 48) is greater than zero, a new archive is started when the size of the current archive exceeds the `archive-file-size-limit`. Only one archive can be active at a time.

If you modify the value of `archive-file-size-limit` while the distributed system is running, the new value does not take effect until the current archive becomes inactive (that is, when a new archive is started).

Maximum Size of All Archive Files

The `archive-disk-space-limit` (page 48) controls the maximum size of all inactive archive files combined. By default, `archive-disk-space-limit` is `0`, meaning that archive space is unlimited.

Whenever an archive becomes inactive or when the archive file is renamed, the combined size of the inactive files is calculated. If the size exceeds the `archive-disk-space-limit`, the inactive archive with the oldest modification time is deleted. This continues until the combined size is less than `archive-disk-space-limit`.

If `archive-disk-space-limit` is less than or equal to `archive-file-size-limit`, when the active archive is made inactive due to its size, it is immediately deleted.

If you modify the value of `archive-disk-space-limit` while the distributed system is running, the new value does not take effect until the current archive becomes inactive.

B.2 GemFire Enterprise System Statistics

GemFire Enterprise provides these standard statistics for your system.

When sampling and archiving are enabled, these statistics are saved in archive files that you can study through the `gemfire stats` command or by using VSD. For information on these tools, see:

- **gemfire stats**—*Commands* on page 229
- **VSD**—*Visual Statistics Display* tool, which can be acquired by contacting GemStone Technical Support

You can also see the statistics through the API, as explained in *Statistics* on page 464 of the *GemFire Enterprise Developer's Guide*.

> *For performance reasons, time-based statistics are disabled by default. To enable, set the gemfire property,* `enable-time-statistics` (page 51)*, to* `true`*. Note that sampling and archiving must also be enabled for this to take effect.*

System Performance Statistics

This section discusses the statistics of primary importance for system performance. Statistics are collected for each Java application or cache server that connects to a distributed system. For pure Java applications, `ProcessStats` and `SystemStats` are not collected.

Cache Client Notifier Statistics

Statistics regarding cache server operations sent to clients. For the statistics that are most useful in detecting slow receivers, see *VM Statistics* on page 278. The primary statistics are:.

`events`	Number of events (operations) processed by the cache client notifier.
`eventProcessingTime`	Total time, in nanoseconds, spent by the cache client notifier processing events.
`clientRegistrations`	Number of clients (operations) that have registered for updates.
`clientRegistrationTime`	Total time, in nanoseconds, spent doing client registrations.
`clientHealthMonitorRegister`	Number of clients that register.
`clientHealthMonitorUnRegister`	Number of clients that unregister.
`durableReconnectionCount`	Number of times the same durable client connects to the server.
`queueDroppedCount`	Number of times the client subscription queue for a particular durable client is dropped.
`eventsEnqueuedWhileClientAway Count`	Number of events enqueued for a durable client.
`cqProcessingTime`	Total time, in nanoseconds, spent by the cache client notifier processing CQs.

Cache Client Proxy Statistics

Statistics regarding cache server operations and cache server client notifications sent to a single client. The primary statistics are:

`messagesReceived`	Number of client operations messages received.
`messagesQueued`	Number of client operations messages added to the subscription queue.
`messagesFailedQueued`	Number of client operations messages attempted but failed to be added to the subscription queue.

`messagesNotQueuedOriginator`	Number of client operations messages received but not added to the subscription queue, because the receiving proxy represents the client originating the message.
`messagesNotQueuedNotInterested`	Number of client operations messages received but not added to the subscription queue because the client represented by the receiving proxy was not interested in the message's key.
`messagesNotQueuedConflated`	Number of client operations messages received but not added to the subscription queue because the queue already contains a message with the message's key.
`messageQueueSize`	Size of the operations subscription queue.
`messagesProcessed`	Number of client operations messages removed from the subscription queue and sent.
`messageProcessingTime`	Total time, in nanoseconds, spent sending messages to clients.
`cqCount`	Number of CQs operations on the client.

Cache Client Updater Statistics

Statistics in a client and pertain to server-to-client data pushed from the server over a queue to the client (they are the client side of the server's `CacheClientNotifierStatistics`). The primary statistics are:

`receivedBytes`	Total number of bytes received from the server.
`messagesBeingReceived`	Current number of message being received off the network or being processed after reception.
`messageBytesBeingReceived`	Current number of bytes consumed by messages being received or processed.

Cache Performance Statistics

Statistics on the GemFire Enterprise cache (available if the member creates a cache). These can be used to determine the type and number of cache operations being performed and how much time they consume. If you are running transactions in your distributed system, additional statistics are available; see *Partitioned Region Statistics* on page 283. The primary statistics are:

`loadsInProgress`	Current number of threads in this cache doing a cache load.
`loadsCompleted`	Total number of times a load on this cache has completed as a result of either a local `get()` or a remote netload.
`loadTime`	Total time spent invoking loaders on this cache.
`netloadsInProgress`	Current number of threads doing a network load initiated by a `get()` in this cache.
`netloadsCompleted`	Total number of times a network load initiated on this cache has completed.
`netloadTime`	Total time spent doing network loads on this cache.
`netsearchesInProgress`	Current number of threads doing a network search initiated by a `get()` in this cache.
`netsearchesCompleted`	Total number of times network searches initiated by this cache have completed.

netsearchTimeDesc	Total time spent doing network searches for cache values.
cacheWriterCallsInProgress	Current number of threads doing a cache writer call.
cacheWriterCallsCompleted	Total number of times a cache writer call has completed.
cacheWriterCallTime	Total time spent doing cache writer calls.
cacheListenerCallsInProgress	Current number of threads doing a cache listener call.
cacheListenerCallsCompleted	Total number of times a cache listener call has completed.
cacheListenerCallTime	Total time spent doing cache listener calls.
getInitialImagesInProgressDesc	Current number of `getInitialImage` operations currently in progress.
getInitialImagesCompleted	Total number of times `getInitialImages` initiated by this cache have completed.
getInitialImageTime	Total time spent doing `getInitialImages` for region creation.
getInitialImageKeysReceived	Total number of keys received while doing `getInitialImage` operations.
regions	The current number of `regions` in the cache.
partitionedRegions	The current number of partitioned regions in the cache.
destroys	The total number of times a cache object entry has been destroyed in this cache.
updates	The total number of `updates` originating remotely that have been applied to this cache.
updateTime	Total time spent performing an update.
invalidates	The total number of times an existing cache object entry value in this cache has been invalidated.
getsDesc	The total number of times a successful `get` has been done on this cache.
creates	The total number of times an entry is added to this cache.
puts	The total number of times an entry is added or replaced in this cache as a result of a local operation (`put()`, `create()`, or `get()` which results in load, netsearch, or netloading a value). Note, this only counts `puts` done explicitly on this cache; it does not count updates pushed from other caches.
putTime	Total time spent adding or replacing an entry in this cache as a result of a local operation. This includes synchronizing on the map, invoking cache callbacks, sending messages to other caches, and waiting for responses (if required).
putAlls	The total number of times a map is added or replaced in this cache as a result of a local operation. Note, this only counts `putAlls` done explicitly on this cache; it does not count updates pushed from other caches.
putallTime	Total time spent replacing a map in this cache as a result of a local operation. This includes synchronizing on the map, invoking cache callbacks, sending messages to other caches and waiting for responses (if required).
getTime	Total time spent doing get operations from this cache (including netsearch and netload).

`eventQueueSize`	The number of cache events waiting to be processed.
`eventQueueThrottleTime`	The total amount of time, in nanoseconds, spent delayed by the event queue throttle.
`eventQueueThrottleCount`	The total number of times a thread was delayed in adding an event to the event queue.
`eventThreads`	The number of threads currently processing events.
`misses`	Total number of times a get on the cache did not find a value already in local memory. The number of hits (that is, gets that did not miss) can be calculated by subtracting `misses` from gets.
`queryExecutions`	Total number of times some query has been executed.
`queryExecutionTime`	Total time spent executing queries.
`reliableQueuedOps`	Current number of cache operations queued for distribution to required roles.
`reliableQueueSize`	Current size in megabytes of disk used to queue for distribution to required roles.
`reliableQueueMax`	Maximum size in megabytes allotted for disk usage to queue for distribution to required roles.
`reliableRegions`	Current number of regions configured for reliability.
`reliableRegionsMissing`	Current number regions configured for reliability that are missing required roles.
`reliableRegionsQueuing`	Current number regions configured for reliability that are queuing for required roles.
`reliableRegionsMissingFullAccess`	Current number of regions configured for reliability that are missing require roles with full access.
`reliableRegionsMissingLimitedAccess`	Current number of regions configured for reliability that are missing required roles with limited access.
`reliableRegionsMissingNoAccess`	Current number of regions configured for reliability that are missing required roles with no access.

Cache Server Statistics

Statistics used for cache servers and for gateway receivers and are recorded in `CacheServerStats` in a cache server. The primary statistics are:

`getRequests`	Number of cache client operations `get` requests.
`readGetRequestTime`	Total time, in nanoseconds, spent in reading `get` requests.
`processGetTime`	Total time, in nanoseconds, spent in processing a cache client `get` request, including the time to get an object from the cache.
`getResponses`	Number of `getResponses` written to the cache client.
`writeGetResponseTime`	Total time, in nanoseconds, spent in writing `get` responses.
`putRequests`	Number of cache client operations `put` requests.
`readPutRequestTime`	Total time, in nanoseconds, spent in reading `put` requests.

processPutTime	Total time, in nanoseconds, spent in processing a cache client `put` request, including the time to put an object into the cache.
putResponses	Number of `putResponses` written to the cache client.
writePutResponseTime	Total time, in nanoseconds, spent in writing `put` responses.
putAllRequests	Number of cache client operations `putAll` requests.
readPutAllRequestTime	Total time, in nanoseconds, spent in reading `putAll` requests.
processPutAllTime	Total time, in nanoseconds, spent in processing a cache client `putAll` request, including the time to `put` all objects into the cache.
putAllResponses	Number of `putAllResponses` written to the cache client.
writePutAllResponseTime	Total time, in nanoseconds, spent in writing `putAll` responses.
destroyRequests	Number of cache client operations `destroy` requests.
readDestroyRequestTime	Total time, in nanoseconds, spent in reading `destroy` requests.
processDestroyTime	Total time, in nanoseconds, spent in processing a cache client `destroy` request, including the time to destroy an object from the cache.
destroyResponses	Number of `destroy` responses written to the cache client.
writeDestroyResponseTime	Total time, in nanoseconds, spent in writing `destroy` responses.
queryRequests	Number of cache client operations `query` requests.
readQueryRequestTime	Total time, in nanoseconds, spent in reading `query` requests.
processQueryTime	Total time, in nanoseconds, spent in processing a cache client `query` request, including the time to destroy an object from the cache.
queryResponses	Number of `query` responses written to the cache client.
writeQueryResponseTime	Total time, in nanoseconds, spent in writing `query` responses.
destroyRegionRequests	Number of cache client operations `destroyRegion` requests.
readDestroyRegionRequestTime	Total time, in nanoseconds, spent in reading `destroyRegion` requests.
processDestroyRegionTime	Total time, in nanoseconds, spent in processing a cache client `destroyRegion` request, including the time to destroy the region from the cache.
destroyRegionResponses	Number of `destroyRegion` responses written to the cache client.
writeDestroyRegionResponseTime	Total time, in nanoseconds, spent in writing `destroyRegion` responses.
containsKeyRequests	Number of cache client operations `containsKey` requests.
readContainsKeyRequestTime	Total time, in nanoseconds, spent reading `containsKey` requests.

processContainsKeyTime	Total time spent, in nanoseconds, processing a containsKey request.
containsKeyResponses	Number of containsKey responses written to the cache client.
writeContainsKeyResponseTime	Total time, in nanoseconds, spent writing containsKey responses.
processBatchRequests	Number of cache client operations processBatch requests.
readProcessBatchRequestTime	Total time, in nanoseconds, spent in reading processBatch requests.
processBatchTime	Total time, in nanoseconds, spent in processing a cache client processBatch request.
processBatchResponses	Number of processBatch responses written to the cache client.
writeProcessBatchResponseTime	Total time, in nanoseconds, spent in writing processBatch responses.
batchSize	The size (in bytes) of the batches received.
clearRegionRequests	Number of cache client operations clearRegion requests.
readClearRegionRequestTime	Total time, in nanoseconds, spent in reading clearRegion requests.
processClearRegionTime	Total time, in nanoseconds, spent in processing a cache client clearRegion request, including the time to clear the region from the cache.
clearRegionResponses	Number of clearRegion responses written to the cache client.
writeClearRegionResponseTime	Total time, in nanoseconds, spent in writing clearRegion responses.
clientNotificationRequests	Number of cache client operations notification requests.
readClientNotificationRequestTime	Total time, in nanoseconds, spent in reading client notification requests.
processClientNotificationTime	Total time, in nanoseconds, spent in processing a cache client notification request.
updateClientNotificationRequests	Number of cache client notification update requests.
readUpdateClientNotificationRequestTime	Total time, in nanoseconds, spent in reading client notification update requests.
processUpdateClientNotificationTime	Total time, in nanoseconds, spent in processing a client notification update request.
clientReadyRequests	Number of cache client ready requests.
readClientReadyRequestTime	Total time, in nanoseconds, spent in reading cache client ready requests.
processClientReadyTime	Total time, in nanoseconds, spent in processing a cache client ready request, including the time to destroy an object from the cache.
clientReadyResponses	Number of client ready responses written to the cache client.

writeClientReadyResponseTime	Total time, in nanoseconds, spent in writing client ready responses.
closeConnectionRequests	Number of cache client close connection operations requests.
readCloseConnectionRequestTime	Total time, in nanoseconds, spent in reading close connection requests.
processCloseConnectionTime	Total time, in nanoseconds, spent in processing a cache client close connection request.
failedConnectionAttempts	Number of failed connection attempts.
currentClientConnections	Number of sockets accepted.
currentClients	Number of client virtual machines (clients) connected.
outOfOrderGatewayBatchIds	Number of Out of Order batch IDs (batches).
abandonedWriteRequests	Number of write operations (requests) abandoned by clients
abandonedReadRequests	Number of read operations (requests) abandoned by clients
receivedBytes	Total number of bytes received from clients.
sentBytes	Total number of bytes sent to clients.
messagesBeingReceived	Current number of messages being received off the network or being processed after reception.
messageBytesBeingReceived	Current number of bytes consumed by messages being received or processed.
connectionsTimedOut	Total number of connections that have been timed out by the server because of client inactivity.
threadQueueSize	Current number of connections waiting for a thread to start processing their message.
acceptsInProgress	Current number of server accepts that are attempting to do the initial handshake with the client.
acceptThreadStarts	Total number of threads created (starts) to deal with an accepted socket. Note, this is not the current number of threads.
connectionThreadStarts	Total number of threads created (starts) to deal with a client connection. Note, this is not the current number of threads.
connectionThreads	Current number of threads dealing with a client connection.
connectionLoad	The load from client to server connections as reported by the load probe installed in this server.
loadPerConnection	The estimate of how much load i.s added for each new connection as reported by the load probe installed in this server.
queueLoad	The load from subscription queues as reported by the load probe installed in this server
loadPerQueue	The estimate of how much load would be added for each new subscription connection as reported by the load probe installed in this server

Client Statistics

These statistics are in a client and they describe all the messages sent from the client to a specific server. The primary statistics are:

opsInProgress	Current number of ops being executed.
opSendsInProgress	Current number of opSends being executed.
opSends	Total number of opSends that have completed successfully.
opSendFailures	Total number of opSends that have failed.
ops	Total number of ops that have completed successfully.
opFailures	Total number of op attempts that have failed.
opTimeouts	Total number of op attempts that have timed out.
opSendTime	Total amount of time, in nanoseconds, spent doing opSends.
opTime	Total amount of time, in nanoseconds, spent doing ops.
getsInProgress	Current number of gets being executed.
getSendsInProgress	Current number of getSends being executed.
getSends	Total number of getSends that have completed successfully.
getSendFailures	Total number of getSends that have failed.
gets	Total number of gets that have completed successfully.
getFailures	Total number of get attempts that have failed.
getTimeouts	Total number of get attempts that have timed out.
getSendTime	Total amount of time, in nanoseconds, spent doing getSends.
getTime	Total amount of time, in nanoseconds, spent doing gets.
putsInProgress	Current number of puts being executed.
putSendsInProgress	Current number of putSends being executed.
putSends	Total number of putSends that have completed successfully.
putSendFailures	Total number of putSends that have failed.
puts	Total number of puts that have completed successfully.
putFailures	Total number of put attempts that have failed.
putTimeouts	Total number of put attempts that have timed out.
putSendTime	Total amount of time, in nanoseconds, spent doing putSends.
putTime	Total amount of time, in nanoseconds, spent doing puts.
destroysInProgress	Current number of destroys being executed.
destroySendsInProgress	Current number of destroySends being executed.
destroySends	Total number of destroySends that have completed successfully.
destroySendFailures	Total number of destroySends that have failed.
destroys	Total number of destroys that have completed successfully.
destroyFailures	Total number of destroy attempts that have failed.
destroyTimeouts	Total number of destroy attempts that have timed out.

destroySendTime	Total amount of time, in nanoseconds, spent doing destroySends.
destroyTime	Total amount of time, in nanoseconds, spent doing destroys.
destroyRegionsInProgress	Current number of destroyRegions being executed.
destroyRegionSendsInProgress	Current number of destroyRegionSends being executed.
destroyRegionSends	Total number of destroyRegionSends that have completed successfully.
destroyRegionSendFailures	Total number of destroyRegionSends that have failed.
destroyRegions	Total number of destroyRegions that have completed successfully.
destroyRegionFailures	Total number of destroyRegion attempts that have failed.
destroyRegionTimeouts	Total number of destroyRegion attempts that have timed out.
destroyRegionSendTime	Total amount of time, in nanoseconds, spent doing destroyRegionSends.
destroyRegionTime	Total amount of time, in nanoseconds, spent doing destroyRegions.
clearsInProgress	Current number of clears being executed.
clearSendsInProgress	Current number of clearSends being executed.
clearSends	Total number of clearSends that have completed successfully.
clearSendFailures	Total number of clearSends that have failed.
clears	Total number of clears completed successfully.
clearFailures	Total number of clear attempts that have failed.
clearTimeouts	Total number of clear attempts that have timed out.
clearSendTime	Total amount of time, in nanoseconds, spent doing clearSends.
clearTime	Total amount of time, in nanoseconds, spent doing clears.
containsKeysInProgress	Current number of containsKeys being executed.
containsKeySendsInProgress	Current number of containsKeySends being executed.
containsKeySends	Total number of containsKeySends that have completed successfully.
containsKeySendFailures	Total number of containsKeySends that have failed.
containsKeys	Total number of containsKeys that completed successfully.
containsKeyFailures	Total number of containsKey attempts that have failed.
containsKeyTimeouts	Total number of containsKey attempts that have timed out.
containsKeySendTime	Total amount of time, in nanoseconds, spent doing containsKeyends.
containsKeyTime	Total amount of time, in nanoseconds, spent doing containsKeys.
keySetsInProgress	Current number of keySets being executed.
keySetSendsInProgress	Current number of keySetSends being executed.

keySetSends	Total number of keySetSends that have completed successfully.
keySetSendFailures	Total number of keySetSends that have failed.
keySets	Total number of keySets that have completed successfully.
keySetFailures	Total number of keySet attempts that have failed.
keySetTimeouts	Total number of keySet attempts that have timed out.
keySetSendTime	Total amount of time, in nanoseconds, spent doing keySetSends.
keySetTime	Total amount of time, in nanoseconds, spent doing keySets.
registerInterestsInProgress	Current number of registerInterests being executed.
registerInterestSendsInProgress	Current number of registerInterestSends being executed.
registerInterestSends	Total number of registerInterestSends that have completed successfully.
registerInterestSendFailures	Total number of registerInterestSends that have failed.
registerInterests	Total number of registerInterests that have completed successfully.
registerInterestFailures	Total number of registerInterest attempts that have failed.
registerInterestTimeouts	Total number of registerInterest attempts that have timed out.
registerInterestSendTime	Total amount of time, in nanoseconds, spent doing registerInterestSends.
registerInterestTime	Total amount of time, in nanoseconds, spent doing registerInterests.
unregisterInterestsInProgress	Current number of unregisterInterests being executed.
unregisterInterestSendsInProgress	Current number of unregisterInterestSends being executed.
unregisterInterestSends	Total number of unregisterInterestSends that have completed successfully.
unregisterInterestSendFailures	Total number of unregisterInterestSends that have failed.
unregisterInterests	Total number of unregisterInterests that have completed successfully
unregisterInterestFailures	Total number of unregisterInterest attempts that have failed.
unregisterInterestTimeouts	Total number of unregisterInterest attempts that have timed out.
unregisterInterestSendTime	Total amount of time, in nanoseconds, spent doing unregisterInterestSends.
unregisterInterestTime	Total amount of time, in nanoseconds, spent doing unregisterInterests.
querysInProgress	Current number of querys being executed.
querySendsInProgress	Current number of querySends being executed.

querySends	Total number of querySends that have completed successfully.
querySendFailures	Total number of querySends that have failed.
querys	Total number of querys completed successfully.
queryFailures	Total number of query attempts that have failed.
queryTimeouts	Total number of query attempts that have timed out.
querySendTime	Total amount of time, in nanoseconds, spent doing querySends.
queryTime	Total amount of time, in nanoseconds. spent doing querys.
createCQsInProgress	Current number of createCQs being executed.
createCQSendsInProgress	Current number of createCQSends being executed.
createCQSends	Total number of createCQSends that have completed successfully.
createCQSendFailures	Total number of createCQSends that have failed.
createCQs	Total number of createCQs that have completed successfully.
createCQFailures	Total number of createCQ attempts that have failed.
createCQTimeouts	Total number of createCQ attempts that have timed out.
createCQSendTime	Total amount of time, in nanoseconds, spent doing createCQSends.
createCQTime	Total amount of time, in nanoseconds, spent doing createCQs.
stopCQsInProgress	Current number of stopCQs being executed.
stopCQSendsInProgress	Current number of stopCQSends being executed.
stopCQSends	Total number of stopCQSends that have completed successfully.
stopCQSendFailures	Total number of stopCQSends that have failed.
stopCQs	Total number of stopCQs that have completed successfully.
stopCQFailures	Total number of stopCQ attempts that have failed.
stopCQTimeouts	Total number of stopCQ attempts that have timed out.
stopCQSendTime	Total amount of time, in nanoseconds, spent doing stopCQSends.
stopCQTime	Total amount of time, in nanoseconds, spent doing stopCQs.
closeCQsInProgress	Current number of closeCQs being executed.
closeCQSendsInProgress	Current number of closeCQSends being executed.
closeCQSends	Total number of closeCQSends that have completed successfully.
closeCQSendFailures	Total number of closeCQSends that have failed.
closeCQs	Total number of closeCQs that have completed successfully.
closeCQFailures	Total number of closeCQ attempts that have failed.
closeCQTimeouts	Total number of closeCQ attempts that have timed out.
closeCQSendTime	Total amount of time, in nanoseconds, spent doing closeCQSends.

closeCQTime	Total amount of time, in nanoseconds, spent doing closeCQs.
gatewayBatchsInProgress	Current number of gatewayBatchs being executed.
gatewayBatchSendsInProgress	Current number of gatewayBatchSends being executed.
gatewayBatchSends	Total number of gatewayBatchSends that have completed successfully.
gatewayBatchSendFailures	Total number of gatewayBatchSends that have failed.
gatewayBatchs	Total number of gatewayBatchs completed successfully.
gatewayBatchFailures	Total number of gatewayBatch attempts that have failed.
gatewayBatchTimeouts	Total number of gatewayBatch attempts that have timed out.
gatewayBatchSendTime	Total amount of time, in nanoseconds, spent doing gatewayBatchSends.
gatewayBatchTime	Total amount of time, in nanoseconds, spent doing gatewayBatchs.
readyForEventsInProgress	Current number of readyForEventss being executed
readyForEventsSendsInProgress	Current number of readyForEventsSends being executed.
readyForEventsSends	Total number of readyForEventsSends that have completed successfully.
readyForEventsSendFailures	Total number of readyForEventsSends that have failed.
readyForEvents	Total number of readyForEventss that have completed successfully.
readyForEventsFailures	Total number of readyForEvents attempts that have failed.
readyForEventsTimeouts	Total number of readyForEvents attempts that have timed out.
readyForEventsSendTime	Total amount of time, in nanoseconds, spent doing readyForEventsSends.
readyForEventsTime	Total amount of time, in nanoseconds, spent doing readyForEvents.
makePrimarysInProgress	Current number of makePrimarys being executed.
makePrimarySendsInProgress	Current number of makePrimarySends being executed.
makePrimarySends	Total number of makePrimarySends that have completed successfully.
makePrimarySendFailures	Total number of makePrimarySends that have failed.
makePrimarys	Total number of makePrimarys that have completed successfully.
makePrimaryFailures	Total number of makePrimary attempts that have failed.
makePrimaryTimeouts	Total number of makePrimary attempts that have timed out.
makePrimarySendTime	Total amount of time, in nanoseconds, spent doing makePrimarySends.
makePrimaryTime	Total amount of time, in nanoseconds, spent doing makePrimarys.
closeConsInProgress	Current number of closeCons being executed.
closeConSendsInProgress	Current number of closeConSends being executed.

closeConSends	Total number of closeConSends that have completed successfully.
closeConSendFailures	Total number of closeConSends that have failed.
closeCons	Total number of closeCons that have completed successfully.
closeConFailures	Total number of closeCon attempts that have failed.
closeConTimeouts	Total number of closeCon attempts that have timed out.
closeConSendTime	Total amount of time, in nanoseconds, spent doing closeConSends.
closeConTime	Total amount of time, in nanoseconds, spent doing closeCons.
primaryAcksInProgress	Current number of primaryAcks being executed.
primaryAckSends	Total number of primaryAckSends that have completed successfully.
primaryAckSendFailures	Total number of primaryAckSends that have failed.
primaryAcks	Total number of primaryAcks that have completed successfully.
primaryAckFailures	Total number of primaryAck attempts that have failed.
primaryAckTimeouts	Total number of primaryAck attempts that have timed out.
primaryAckSendTime	Total amount of time, in nanoseconds, spent doing primaryAckSends.
primaryAckTime	Total amount of time, in nanoseconds, spent doing primaryAcks.
pingsInProgress	Current number of pings being executed.
pingSendsInProgress	Current number of pingSends being executed.
pingSends	Total number of pingSends that have completed successfully.
pingSendFailures	Total number of pingSends that have failed.
pings	Total number of pings that have completed successfully.
pingFailures	Total number of ping attempts that have failed.
pingTimeouts	Total number of ping attempts that have timed out.
pingSendTime	Total amount of time, in nanoseconds, spent doing pingSends.
pingTime	Total amount of time, in nanoseconds, spent doing pings.
registerInstantiatorssInProgress	Current number of registerInstantiators being executed
registerInstantiatorsSendsInProgress	Current number of registerInstantiators sends being executed
registerInstantiatorsSends	Total number of registerInstantiators sends that have completed successfully
registerInstantiatorsSendFailures	Total number of registerInstantiators sends that have failed
registerInstantiators	Total number of registerInstantiators completed successfully

registerInstantiatorsFailures	Total number of registerInstantiators attempts that have failed.
registerInstantiatorsTimeouts	Total number of registerInstantiators attempts that have timed out.
registerInstantiatorsSendTime	Total amount of time, in nanoseconds, spent doing registerInstantiatorsSends.
registerInstantiatorsTime	Total amount of time, in nanoseconds, spent doing registerInstantiators.
connections	Current number of connections.
connects	Total number of times a connection has been created.
disconnects	Total number of times a connection has been destroyed.
putAllsInProgress	Current number of putAlls being executed.
putAllSendsInProgress	Current number of putAllSends being executed.
putAllSends	Total number of putAllSends that have completed successfully.
putAllSendFailures	Total number of putAllSends that have failed.
putAlls	Total number of putAlls that have completed successfully.
putAllFailures	Total number of putAll attempts that have failed.
putAllTimeouts	Total number of putAll attempts that have timed out.
putAllSendTime	Total amount of time, in nanoseconds, spent doing putAllSends.
putAllTime	Total amount of time, in nanoseconds, spent doing putAlls.
getAllsInProgress	Current number of getAlls being executed.
getAllSendsInProgress	Current number of getAllSends being executed.
getAllSends	Total number of getAllSends that have completed successfully.
getAllSendFailures	Total number of getAllSends that have failed.
getAlls	Total number of getAlls that have completed successfully.
getAllFailures	Total number of getAll attempts that have failed.
getAllTimeouts	Total number of getAll attempts that have timed out.
getAllSendTime	Total amount of time, in nanoseconds, spent doing getAllSends.
getAllTime	Total amount of time, in nanoseconds, spent doing getAlls.
receivedBytes	Total number of bytes received from the server.
sentBytes	Total number of bytes sent to the server.
messagesBeingReceived	Current number of messages being received off the network or being processed after reception.
messageBytesBeingReceived	Current number of bytes consumed by messages being received or processed.

CQ Statistics

These statistics are for continuous querying information for a single CQ and for the query service's management of CQs as a whole. The primary statistics are:

CQS_CREATED	Number of CQ operations created.
CQS_ACTIVE	Number of CQ operations actively executing.
CQS_STOPPED	Number of CQ operations stopped.
CQS_CLOSED	Number of CQ operations closed.
CQS_ON_CLIENT	Number of CQ operations on the client.
CLIENTS_WITH_CQS	Number of Clients with CQ operations.
CQ_QUERY_EXECUTION_TIME	Time taken, in nanoseconds, for CQ query execution.
CQ_QUERY_EXECUTIONS_COMPLETED	Number of CQ query executions operations.
CQ_QUERY_EXECUTION_IN_ PROGRESS	CQ Query execution operations in progress.
UNIQUE_CQ_QUERY	Number of unique CQ queries.

Delta Propagation Statistics

These statistics are for delta propagation between members:

processedDeltaMessages	The number of distribution messages containing delta that this GemFire system has processed.
processedDeltaMessagesTime	The amount of time this distribution manager has spent in applying delta on its existing value.
preparedDeltaMessages	The number of distribution messages containing delta that this GemFire system has prepared for distribution.
preparedDeltaMessagesTime	The total amount of time this distribution manager has spent preparing delta parts of messages.
deltaMessageFailures	The number of distribution messages containing delta that could not be processed at receiving side.
fullValueDeltaMessagesSent	The number of distribution messages sent in response to full value requests by a remote GemFire System as a result of failure in applying delta.
fullValueDeltaMessagesRequested	The number of distribution messages containing full value requested by this GemFire system after failing to apply received delta.
partitionMessagesWithDeltaSent	Number of PartitionMessages containing delta sent.
partitionMessagesWithDeltaSentTime	Total time spent extractng deltas.
partitionMessagesWithDeltaProcessed	Number of PartitionMessages containing delta processed.
partitionMessagesWithDeltaProcessedTime	Total time spent applying deltas.
partitionMessagesWithDeltaFailures	Number of failures while processing PartitionMessages containing delta.

partitionMessagesWithFullValueDeltaSent	Number of PartitionMessages containing full delta value sent.
partitionMessagesWithFullValueDeltaRequested	Number of requests for PartitionMessages containing full delta value as a result of failure in applying delta.
processedDeltaPuts	Number of cache client put requests containing delta received from a client and processed successfuly.
processedDeltaPutsTime	Total time spent in applying delta received from a client on existing value in this server's region.
deltaPutFailures	Number of failures encountered while processing delta received from a client on this server.
fullDeltaRequests	Number of full value requests made by this server to the sender client after failing to apply delta.
deltaFullValueRequests	Number of full value requests received from a client after failing to apply delta and processed successfully by this server.
deltaPuts	Total number of puts containing delta.
deltaPutsTime	Total amount of time, in nanoseconds, spent constructing delta part of puts.
deltaFullValuePuts	Total number of full value puts processed successfully in response to failed delta puts.
processedDeltaMessages	Current number of delta messages received off network and processed after reception.
deltaMessageFailures	Current number of delta messages received but could not be processed after reception.
processedDeltaMessagesTime	Total time spent applying received delta parts on existing messages at clients.
fullDeltaMessages	Current number of full value delta messages received off network and processed after reception.
preparedDeltaMessages	Number of client messages being prepared for dispatch, which have delta part in them.

DiskDirStatistics

These statistics pertain to the disk usage for a region’s disk directory. The primary statistics are:

diskSpace	The total number of bytes current being used on disk in this directory.

Disk Region Statistics

Statistics regarding operations performed on a disk region for persistence/overflow. The primary statistics are:

commits	Total number of commits.
commitTime	Total amount of time, in nanoseconds, spent doing commits.
writes	Total number of region entries that have been written to disk. A write is done every time an entry is created on disk or every time its value is modified on the disk.

writeTime	Total amount of time, in nanoseconds, spent writing to the disk.
writtenBytes	Total number of bytes that have been written to the disk.
flushes	Total number of times the async write buffer has been flushed.
flushTime	Total amount of time, in nanoseconds, spent doing a buffer flush.
flushedBytes	Total number of bytes flushed out of the async write buffer to the disk.
reads	Total number of region entries that have been read from the disk.
readTime	Total amount of time, in nanoseconds, spent reading from the disk.
readBytes	Total number of bytes that have been read from the disk.
recoveryTime	Total amount of time, in nanoseconds, spent doing a recovery.
recoveredBytes	Total number of bytes that have been read from disk during a recovery.
removes	Total number of region entries that have been removed from the disk.
removeTime	Total amount of time, in nanoseconds, spent removing from the disk.
bufferSize	Current number of bytes buffered to be written to the disk.
entriesOnDisk	Current number of entries whose value is on the disk and is not in memory. This is true of overflowed entries. It is also true of recovered entries that have not yet been faulted in.
entriesInVM	Current number of entries whose value resides in the VM. The value may also have been written to the disk.

Distribution Statistics

Statistics on the GemFire distribution layer. These can be used to tell how much message traffic there is between this member and other distributed system members. For the statistics that are most useful in detecting slow receivers, see *Distribution Statistics Related to Slow Receivers* on page 263. The primary statistics are:

sentMessagesDesc	The number of distribution messages that the GemFire system has sent, which includes broadcastMessages.
sentCommitMessagesDesc	The number of transaction commit messages that the GemFire system has created to be sent. Note, it is possible for a commit to only create one message even though it will end up being sent to multiple recipients.
commitWaitsDesc	The number of transaction commits that had to wait for a response before they could complete.
sentMessagesTimeDesc	The total amount of time this distribution manager has spent sending messages, which includes broadcastMessagesTime.

sentMessagesMaxTimeDesc	The highest amount of time this distribution manager has spent distributing a single message to the network.
broadcastMessagesDesc	The number of distribution messages that the GemFire system has broadcast. A broadcast message is one sent to every other manager in the group.
broadcastMessagesTimeDesc	The total amount of time this distribution manager has spent broadcasting messages. A broadcast message is one sent to every other manager in the group.
receivedMessagesDesc	The number of distribution messages that the GemFire system has received.
receivedBytesDesc	The number of distribution message bytes that the GemFire system has received.
sentBytesDesc	The number of distribution message bytes that the GemFire system has sent.
processedMessagesDesc	The number of distribution messages that the GemFire system has processed.
processedMessagesTimeDesc	The amount of time this distribution manager has spent in `message.process()`.
messageProcessingScheduleTimeDesc	The amount of time this distribution manager has spent dispatching a message to processor threads.
overflowQueueSizeDesc	The number of normal distribution messages currently waiting to be processed.
waitingQueueSizeDesc	The number of distribution messages currently waiting for some other resource before they can be processed.
overflowQueueThrottleTimeDesc	The total amount of time, in nanoseconds, spent delayed by the overflow queue throttle.
overflowQueueThrottleCountDesc	The total number of times a thread was delayed in adding a normal message to the overflow queue.
highPriorityQueueSizeDesc	The number of high priority distribution messages currently waiting to be processed.
highPriorityQueueThrottleTimeDesc	The total amount of time, in nanoseconds, spent delayed by the high priority queue throttle.
highPriorityQueueThrottleCounDesc	The total number of times a thread was delayed in adding a normal message to the high priority queue.
serialQueueSizeDesc	The number of serial distribution messages currently waiting to be processed.
serialQueueBytesDesc	The approximate number of bytes consumed by serial distribution messages currently waiting to be processed.
serialPooledThreadDesc	The number of threads created in the SerialQueuedExecutorPool.
serialQueueThrottleTimeDesc	The total amount of time, in nanoseconds, spent delayed by the serial queue throttle.
serialQueueThrottleCountDesc	The total number of times a thread was delayed in adding a ordered message to the serial queue.
serialThreadsDesc	The number of threads currently processing serial/ordered messages.

processingThreadsDesc	The number of threads currently processing normal messages.
highPriorityThreadsDesc	The number of threads currently processing high priority messages.
partitionedRegionThreadsDesc	The number of threads currently processing partitioned region messages.
waitingThreadsDesc	The number of threads currently processing messages that had to wait for a resource.
messageChannelTimeDesc	The total amount of time received messages spent in the distribution channel.
replyMessageTimeDesc	The amount of time spent processing reply messages; `final String distributeMessageTimeDesc` = The amount of time it takes to prepare a message and send it on the network. This includes `sentMessagesTime`.
nodesDesc	The current number of nodes in this distributed system.
replyWaitsInProgressDesc	Current number of threads waiting for a reply.
replyWaitsCompletedDesc	Total number of times `waits` for a reply have completed.
replyWaitTimeDesc	Total time spent waiting for a reply to a message.
replyWaitMaxTimeDesc	Maximum time spent transmitting and then waiting for a reply to a message. See `sentMessagesMaxTime` for related information.
replyTimeoutsDesc	Total number of message replies that have timed out.
receiverConnectionsDesc	Current number of sockets dedicated to receiving messages.
failedAcceptsDesc	Total number of times an `accept` (receiver creation) of a connect from some other member has failed.
failedConnectsDesc	Total number of times a connect (sender creation) to some other member has failed.
reconnectAttemptsDesc	Total number of times an established connection was lost and a reconnect was attempted.
lostConnectionLeaseDesc	Total number of times an unshared sender socket has remained idle long enough that its lease expired.
sharedOrderedSenderConnectionsDesc	Current number of shared sockets dedicated to sending ordered messages.
sharedUnorderedSenderConnectionsDesc	Current number of shared sockets dedicated to sending unordered messages.
threadOrderedSenderConnectionsDesc	Current number of thread sockets dedicated to sending ordered messages.
threadUnorderedSenderConnectionsDesc	Current number of thread sockets dedicated to sending unordered messages.
asyncQueuesDesc	Current number of queues for asynchronous messaging.
asyncQueueFlushesInProgressDesc	Current number of asynchronous queues being flushed.
asyncQueueFlushesCompletedDesc	Total number of asynchronous queue flushes completed.
asyncQueueFlushTimeDesc	Total time spent flushing asynchronous queues.

asyncQueueTimeoutExceededDesc	Total number of asynchronous queues that have timed out by being blocked for more than async-queue-timeout milliseconds.
asyncQueueSizeExceededDesc	Total number of asynchronous queues that have exceeded the maximum size.
asyncDistributionTimeoutExceededDesc	Total number of times the async-distribution-timeout has been exceeded during a socket write.
asyncQueueSizeDesc	Current size in bytes used for asynchronous queues.
asyncQueuedMsgsDesc	The total number of queued messages used for asynchronous queues.
asyncDequeuedMsgsDesc	The total number of queued messages that have been removed from the queue and successfully sent.
asyncConflatedMsgsDesc	The total number of queued conflated messages used for asynchronous queues.
asyncThreadsDesc	Total number of asynchronous message queue threads.
asyncThreadInProgressDesc	Current iterations of work performed by asynchronous message queue threads.
asyncThreadCompletedDesc	Total number of iterations of work performed by asynchronous message queue threads.
asyncThreadTimeDesc	Total time spent by asynchronous message queue threads performing iterations.
receiverDirectBufferSizeDesc	Current number of bytes allocated from direct memory as buffers for incoming messages.
receiverHeapBufferSizeDesc	Current number of bytes allocated from Java heap memory as buffers for incoming messages.S
senderDirectBufferSizeDesc	Current number of bytes allocated from direct memory as buffers for outgoing messages.
senderHeapBufferSizeDesc	Current number of bytes allocated from Java heap memory as buffers for outgoing messages.
replyHandoffTimeDesc	Total number of seconds to switch thread contexts from processing thread to application thread.
partitionedRegionThreadJobsDesc	The number of messages currently being processed by partitioned region threads.
viewThreadsDesc	The number of threads currently processing view messages.
serialThreadJobsDesc	The number of messages currently being processed by serial threads.
viewThreadJobsDesc	The number of messages currently being processed by view threads.
serialPooledThreadJobsDesc	The number of messages currently being processed by pooled serial processor threads.
processingThreadJobsDesc	The number of messages currently being processed by pooled message processor threads.
highPriorityThreadJobsDesc	The number of messages currently being processed by high priority processor threads.
waitingThreadJobsDesc	The number of messages currently being processed by waiting pooly processor threads.

`syncSocketWritesInProgress`	Current number of synchronous/blocking socket write calls in progress.
`syncSocketWriteTime`	Total amount of time, in nanoseconds, spent in synchronous/blocking socket write calls.
`syncSocketWrites`	Total number of completed synchronous/blocking socket write calls.
`syncSocketWriteBytes`	Total number of bytes sent out in synchronous/blocking mode on sockets.
`ucastReads`	Total number of unicast datagrams received.
`ucastReadBytes`	Total number of bytes received in unicast datagrams.
`ucastWriteTime`	Total amount of time, in nanoseconds, spent in unicast datagram socket write calls.
`ucastWrites`	Total number of unicast datagram socket write calls.
`ucastWriteBytes`	Total number of bytes sent out on unicast datagram sockets.
`ucastRetransmits`	Total number of unicast datagram socket retransmissions.
`mcastReads`	Total number of multicast datagrams received.
`mcastReadBytes`	Total number of bytes received in multicast datagrams.
`mcastWriteTime`	Total amount of time, in nanoseconds, spent in multicast datagram socket write calls.
`mcastWrites`	Total number of multicast datagram socket write calls.
`mcastWriteBytes`	Total number of bytes sent out on multicast datagram sockets.
`mcastRetransmits`	Total number of multicast datagram socket retransmissions.
`mcastRetransmitRequests`	Total number of multicast datagram socket retransmission requests sent to other processes.
`serializationTime`	Total amount of time, in nanoseconds, spent serializing objects.
`serializations`	Total number of object serialization calls.
`serializedBytes`	Total number of bytes produced by object serialization.
`deserializationTime`	Total amount of time, in nanoseconds, spent deserializing objects.
`deserializations`	Total number of object deserialization calls.
`deserializedBytes`	Total number of bytes consumed by object deserialization.
`msgSerializationTime`	Total amount of time, in nanoseconds, spent serializing messages.
`msgDeserializationTime`	Total amount of time, in nanoseconds, spent deserializing messages.
`batchSendTime`	Total amount of time, in nanoseconds, spent queueing and flushing message batches.
`batchWaitTime`	Reserved for future use
`batchCopyTime`	Total amount of time, in nanoseconds, spent copying messages for batched transmission.
`batchFlushTime`	Total amount of time, in nanoseconds, spent flushing batched messages to the network.

ucastFlushes	Total number of flushes of the unicast datagram protocol, prior to sending a multicast message.
ucastFlushTime	Total amount of time, in nanoseconds, spent waiting for acknowledgements for outstanding unicast datagram messages.
flowControlRequests	Total number of flow control credit requests sent to other processes.
flowControlResponses	Total number of flow control credit responses sent to a requestor.
flowControlWaitsInProgress	Number of threads blocked waiting for flow-control recharges from other processes.
flowControlWaitTime	Total amount of time, in nanoseconds, spent waiting for other processes to recharge the flow of the control meter.
flowControlThrottleWaitsInProgress	Number of threads blocked waiting due to flow-control throttle requests from other members.
jgNAKACKreceivedMessages	Number of received messages awaiting stability in NAKACK.
jgNAKACKsentMessages	Number of sent messages awaiting stability in NAKACK.
jgUNICASTreceivedMessages	Number of received messages awaiting receipt of prior messages.
jgUNICASTsentMessages	Number of un-acked normal priority messages.
jgUNICASTsentHighPriorityMessages	Number of un-acked high priority messages
jgUNICASTdataReceivedTime	Amount of time spent in JGroups UNICAST send.
jgSTABLEsuspendTime	Amount of time JGroups STABLE is suspended.
jgSTABLEmessages	Number of STABLE messages received by JGroups.
jgSTABLEmessagesSent	Number of STABLE messages sent by JGroups.
jgSTABILITYmessages	Number of STABILITY messages received by JGroups.
jgUDPupTime	Time, in nanosecnds, spent in JGroups UDP processing up events.
jgUDPdownTime	Time, in nanoseconds, spent in JGroups UDP processing down events.
jgNAKACKupTime	Time, in nanoseconds, spent in JGroups NAKACK processing up events.
jgNAKACKdownTime	Time, in nanoseconds, spent in JGroups NAKACK processing down events.
jgUNICASTupTime	Time, in nanoseconds, spent in JGroups UNICAST processing up events.
jgUNICASTdownTime	Time, in nanoseconds, spent in JGroups UNICAST processing down events.
jgSTABLEupTime	Time, in nanoseconds, spent in JGroups STABLE processing up events.
jgSTABLEdownTime	Time, in nanoseconds, spent in JGroups STABLE processing down events.

jgFRAG2upTime	Time, in nanoseconds, spent in JGroups FRAG2 processing up events.
jgFRAG2downTime	Time, in nanoseconds, spent in JGroups FRAG2 processing down events.
jgGMSupTime	Time, in nanoseconds, spent in JGroups GMS processing up events.
jgGMSdownTime	Time, in nanoseconds, spent in JGroups GMS processing down events.
jgFCupTime	Time, in nanoseconds, spent in JGroups FC processing up events.
jgFCdownTime	Time, in nanoseconds, spent in JGroups FC processing down events.
jgDirAckupTime	Time, in nanoseconds, spent in JGroups DirAck processing up events.
jgDirAckdownTime	Time, in nanoseconds, spent in JGroups DirAck processing down events.
jgVIEWSYNCdownTime	Time, in nanoseconds, spent in JGroups VIEWSYNC processing down events.
jgVIEWSYNCupTime	Time, in nanoseconds, spent in JGroups VIEWSYNC processing up events.
jgFDdownTime	Time, in nanoseconds, spent in JGroups FD processing down events.
jgFDupTime	Time, in nanoseconds, spent in JGroups FD processing up events.
jgTCPGOSSIPdownTime	Time, in nanoseconds, spent in JGroups TCPGOSSIP processing down events.
jgTCPGOSSIPupTime	Time, in nanoseconds, spent in JGroups TCPGOSSIP processing up events.
jgDISCOVERYdownTime	Time, in nanoseconds, spent in JGroups DISCOVERY processing down events.
jgDISCOVERYupTime	Time, in nanoseconds, spent in JGroups DISCOVERY processing up events.
jgDownTime	Down Time spent in JGroups stacks.
jgUpTime	Up Time spent in JGroups stacks.
jChannelUpTime	Up Time spent in JChannel including jgroup stack.
jgFCsendBlocks	Number of times JGroups FC halted send events due to backpressure.
jgFCautoRequests	Number of times JGroups FC automatically sent replenishment requests.
jgFCreplenish	Number of times JGroups FC received replenishments messages from receivers.
jgFCresumes	Number of times JGroups FC resumed sends events due to backpressure.
jgFCsentCredits	Number of times JGroups FC sent credits events to a sender.

jgFCsentThrottleRequests	Number of times JGroups FC sent throttle events requests to a sender.
asyncSocketWritesInProgress	Current number of non-blocking socket write calls in progress.
asyncSocketWrites	Total number of non-blocking socket write calls completed.
asyncSocketWriteRetries	Total number of retries needed to write a single block of data using non-blocking socket write calls.
asyncSocketWriteTime	Total amount of time, in nanoseconds, spent in non-blocking socket write calls.
asyncSocketWriteBytes	Total number of bytes sent out on non-blocking sockets.
asyncQueueAddTime	Total amount of time, in nanoseconds, spent in adding messages to async queue.
asyncQueueRemoveTime	Total amount of time, in nanoseconds, spent in removing messages from async queue.
jgDirAcksReceived	Number of DirAck acks received.
jgFragmentationsPerformed	Number of message fragmentation operations performed.
jgFragmentsCreated	Number of message fragments created.
socketLocks	Total number of times a socket has been locked.
socketLockTime	Total amount of time, in nanoseconds, spent locking a socket.
bufferAcquiresInProgress	Current number of threads waiting to acquire a buffer.
bufferAcquires	Total number of times a buffer has been acquired.
bufferAcquireTime	Total amount of time, in nanoseconds, spent acquiring a socket.
messagesBeingReceived	Current number of messages being received off the network or being processed after reception.
messageBytesBeingReceived	Current number of bytes consumed by messages being received or processed.
serialThreadStarts	Total number of times a thread has been created for the serial message executor.
viewThreadStarts	Total number of times a thread has been created for the view message executor.
processingThreadStarts	Total number of times a thread has been created for the pool processing normal messages.
highPriorityThreadStarts	Total number of times a thread has been created for the pool handling high priority messages.
waitingThreadStarts	Total number of times a thread has been created for the waiting pool.
partitionedRegionThreadStarts	Total number of times a thread has been created for the pool handling partitioned region messages.
serialPooledThreadStarts	Total number of times a thread has been created for the serial pool(s).
TOSentMsgs	Total number of messages sent on thread owned senders.

Distribution Statistics Related to Slow Receivers

The distribution statistics provide statistics pertaining to slow receivers. The primary statistics are:

`asyncSocketWrite*`	Used anytime a producer is distributing to one or more consumers with a non-zero distribution timeout. These statistics also reflect the writes done by the threads that service asynchronous queues.
`asyncQueue*`	Provide information about queues the producer is managing for its consumers. There are no statistics maintained for individual consumers. The following are the primary statistics of this type.
`asyncQueues`	Indicates the number of queues currently in the producer.
`asyncQueueTimeoutExceeded`	Incremented every time a queue flushing has exceeded `async-queue-timeout` and the receiver has been sent a disconnect message.
`asyncQueueSizeExceeded`	Incremented every time a queue has exceeded `async-max-queue-size` and the receiver has been sent a disconnect message.
`asyncDistributionTimeoutExceeded`	Incremented every time an `asyncSocketWrite` has exceeded `async-distribution-timeout` and an async queue has been created.

DLock Statistics

These statistics are for distributed lock services. The primary statistics are:

`grantorsDesc`	The current number of lock grantors hosted by this system member.
`servicesDesc`	The current number of lock services used by this system member.
`tokensDesc`	The current number of lock tokens used by this system member.
`requestQueuesDesc`	The current number of lock request queues used by this system member.
`serialQueueSizeDesc`	The number of serial distribution messages currently waiting to be processed.
`serialThreadsDesc`	The number of threads currently processing serial/ordered messages.
`waitingQueueSizeDesc`	The number of distribution messages currently waiting for some other resource before they can be processed.
`waitingThreadsDesc`	The number of threads currently processing messages that had to wait for a resource.
`lockWaitsInProgressDesc`	Current number of threads waiting for a distributed lock.
`lockWaitsCompletedDesc`	Total number of times distributed lock wait has completed by successfully obtaining the lock.
`lockWaitTimeDesc`	Total time spent waiting for a distributed lock that was obtained.

lockWaitsFailedDesc	Total time spent waiting for a distributed lock that failed to be obtained.
lockWaitFailedTimeDesc	Total number of times distributed lock wait has completed by failing to obtain the lock.
grantWaitsInProgressDesc	Current number of distributed lock requests being granted.
grantWaitsCompletedDesc	Total number of times granting of a lock request has completed by successfully granting the lock.
grantWaitTimeDesc	Total time spent attempting to grant a distributed lock.
grantWaitsNotGrantorDesc	Total number of times granting of lock request failed because not grantor.
grantWaitNotGrantorTimeDesc	Total time spent granting of lock requests that failed because not grantor.
grantWaitsTimeoutDesc	Total number of times granting of lock request failed because of a timeout.
grantWaitTimeoutTimeDesc	Total time spent granting of lock requests that failed because of a timeout.
grantWaitsNotHolderDesc	Total number of times granting of lock request failed because reentrant was not holder.
grantWaitNotHolderTimeDesc	Total time spent granting of lock requests that failed because reentrant was not holder.
grantWaitsFailedDesc	Total number of times granting of lock request failed because try locks failed.
grantWaitFailedTimeDesc	Total time spent granting of lock requests that failed because try locks failed.
grantWaitsSuspendedDesc	Total number of times granting of lock request failed because lock service was suspended.
grantWaitSuspendedTimeDesc	Total time spent granting of lock requests that failed because lock service was suspended.
grantWaitsDestroyedDesc	Total number of times granting of lock request failed because lock service was destroyed.
grantWaitDestroyedTimeDesc	Total time spent granting of lock requests that failed because lock service was destroyed.
createGrantorsInProgressDesc	Current number of initial grantors being created in this process.
createGrantorsCompletedDesc	Total number of initial grantors created in this process.
String createGrantorTimeDesc	Total time spent waiting create the initial grantor for lock services.
serviceCreatesInProgressDesc	Current number of lock services being created in this process.
serviceCreatesCompletedDesc	Total number of lock services created in this process.
serviceCreateLatchTimeDesc	Total time spent creating lock services before releasing create latches.
serviceInitLatchTimeDesc	Total time spent creating lock services before releasing init latches.
grantorWaitsInProgressDesc	Current number of threads waiting for grantor latch to open.

grantorWaitsCompletedDesc	Total number of times waiting threads completed waiting for the grantor latch to open.
grantorWaitTimeDesc	Total time spent waiting for the grantor latch which resulted in success.
grantorWaitsFailedDesc	Total number of times waiting threads failed to finish waiting for the grantor latch to open.
grantorWaitFailedTimeDesc	Total time spent waiting for the grantor latch which resulted in failure.
grantorThreadsInProgressDesc	Current iterations of work performed by grantor thread.
grantorThreadsCompletedDesc	Total number of iterations of work performed by grantor thread(s).
grantorThreadExpireAndGrantLocksTimeDesc	Total time spent by grantor thread(s) performing `expireAndGrantLocks` tasks.
grantorThreadHandleRequestTimeoutsTimeDesc	Total time spent by grantor thread(s) performing `handleRequestTimeouts` tasks.";
grantorThreadRemoveUnusedTokensTimeDesc	Total time spent by grantor thread(s) performing `removeUnusedTokens` tasks.
grantorThreadTimeDesc	Total time spent by grantor thread(s) performing all grantor tasks.
pendingRequestsDesc	The current number of pending lock requests queued by grantors in this process.
destroyReadWaitsInProgress Desc	Current number of threads waiting for a `DLockService` destroy read lock.
destroyReadWaitsCompletedDesc	Total number of times a `DLockService` destroy read lock wait has completed successfully.
destroyReadWaitTimeDesc	Total time spent waiting for a `DLockService` destroy read lock that was obtained.
destroyReadWaitsFailedDesc	Total number of times a `DLockService` destroy read lock wait has completed unsuccessfully.
destroyReadWaitFailedTimeDesc	Total time spent waiting for a `DLockService` destroy read lock that was not obtained.
destroyWriteWaitsInProgressDesc	Current number of writes waiting for a `DLockService` destroy write lock.
destroyWriteWaitsCompletedDesc	Total number of times a `DLockService` destroy write lock wait has completed successfully.
destroyWriteWaitTimeDesc	Total time spent waiting for a `DLockService` destroy write lock that was obtained.
destroyWriteWaitsFailedDesc	Total number of times a `DLockService` destroy write lock wait has completed unsuccessfully.
destroyWriteWaitFailedTimeDesc	Total time spent waiting for a `DLockService` destroy write lock that was not obtained.
destroyReadsDesc	The current number of `DLockService` destroy read locks held by this process.
destroyWritesDesc	The current number of `DLockService` destroy write locks held by this process.
lockReleasesInProgressDesc	Current number of threads releasing a distributed lock.

lockReleasesCompletedDesc	Total number of times distributed lock release has completed.
lockReleaseTimeDesc	Total time spent releasing a distributed lock.
becomeGrantorRequestsDesc	Total number of times this member has explicitly requested to become lock grantor.

Function Service Statistics

The following are the aggregate Function Execution statistics for all function executions.:

functionExecutionsCompleted	Total number of completed function.execute() calls.
functionExecutionsCompletedProcessingTime	Total time consumed for all completed function invocations.
functionExecutionsRunning	Number of function invocations that are currently running.
resultsSentToResultCollector	Total number of results sent to the ResultCollector.
resultsReceived	Total number of results received and passed to the ResultCollector.
functionExecutionsHasResultCompletedProcessingTime	Total time consumed for all completed execute() calls where hasResult() returns true.
functionExecutionsHasResultRunning	A gauge indicating the number of currently active execute() calls for functions where hasResult() returns true.
functionExecutionsExceptions	Total number of Exceptions Occured while executing functions.

Function Statistics

These are the statistics for each execution of the function.

functionExecutionsCompleted	Total number of completed function.execute() calls for given function.
functionExecutionsCompletedProcessingTime	Total time consumed for all completed invocations of the given function.
functionExecutionsRunning	number of currently running invocations of the given function.
resultsSentToResultCollector	Total number of results sent to the ResultCollector.
functionExecutionCalls	Total number of FunctionService.execute() calls for given function.
functionExecutionsHasResultCompletedProcessingTime	Total time consumed for all completed given function.execute() calls where hasResult() returns true.
functionExecutionsHasResultRunning	A gauge indicating the number of currently active execute() calls for functions where hasResult() returns true.
resultsReceived	Total number of results received and passed to the ResultCollector.
functionExecutionsExceptions	Total number of Exceptions Occurred while executing function.

Gateway Statistics

These statistics are for an outgoing gateway queue and connection. The primary statistics are:

`eventsQueued`	Number of events operations added to the event queue.
`eventsNotQueuedConflated`	Number of events operations received but not added to the event queue because the queue already contains an event with the event's key.
`eventQueueTime`	Total time, in nanoseconds, spent queueing events.
`eventQueueSize`	Size of the event operations queue.
`eventsDistributed`	Number of events operations removed from the event queue and sent.
`batchDistributionTime`	Total time, in nanoseconds, spent distributing batches of events to other gateways.
`batchesDistributed`	Number of batches of events operations removed from the event queue and sent.
`batchesRedistributed`	Number of batches of events operations removed from the event queue and resent.
`unprocessedTokensAddedByPrimary`	Number of tokens added through a listener to the secondary's unprocessed token map by the primary.
`unprocessedEventsAddedBySecondary`	Number of events added to the secondary's unprocessed event map by the secondary.
`unprocessedEventsRemovedByPrimary`	Number of events removed through a listener from the secondary's unprocessed event map by the primary.
`unprocessedTokensRemovedBySecondary`	Number of tokens removed from the secondary's unprocessed token map by the secondary.
`unprocessedEventsRemovedByTimeout`	Number of events removed from the secondary's unprocessed event map by a timeout.
`unprocessedTokensRemovedByTimeout`	Number of tokens removed from the secondary's unprocessed token map by a timeout.
`unprocessedEventMapSize`	Current number of events entries in the secondary's unprocessed event map.
`unprocessedTokenMapSize`	Current number of tokens entries in the secondary's unprocessed token map.

Gateway Hub Statistics

These statistics are for the WAN gateway hub. The primary statistics are:

`eventsReceived`	Number of events operations received by this hub.
`eventsQueued`	Number of events operations added to the event queue by this hub.
`eventQueueTime`	Total time, in nanoseconds, spent queueing events
`eventQueueSize`	Size of the event operations queue.
`eventsProcessed`	Number of events operations removed from the event queue and processed by this hub.
`numberOfGateways`	Number of gateways operations known to this hub.

Locator Statistics

These statistics are on the GemFire locator. The primary statistics are:

KNOWN_LOCATORS	Number of locators known to this locator.
REQUESTS_TO_LOCATOR	Number of requests this locator has received from clients.
RESPONSES_FROM_LOCATOR	Number of responses this locator has sent to clients.
ENDPOINTS_KNOWN	Number of servers this locator knows about.
REQUESTS_IN_PROGRESS	The number of location requests currently being processed by the thread pool.
REQUEST_TIME	Time, measured in nanoseconds, spent processing server location requests.
RESPONSE_TIME	Time, measured in nanoseconds, spent sending location responses to clients.
SERVER_LOAD_UPDATES	Total number of times a server load update has been received.

LRU Statistics – Count-based

The entry-count least recently used (LRU) eviction mechanism records these LRUStatistics. The primary statistics are:

entriesAllowed	Number of entries allowed in this region.
entryCount	Number of entries in this region.
lruEvictions	Number of total entry evictions triggered by an LRU.
lruDestroys	Number of entry destroys triggered by an LRU.
lruDestroysLimit	Maximum number of entry destroys triggered by an LRU before a scan occurs.
lruEvaluations	Number of entries evaluated during LRU operations
lruGreedyReturns	Number of non-LRU entries evicted during LRU operations.

LRU Statistics – Size-based

The least recently used (LRU) mechanism that keeps the size of a region under a given set point records these MemLRUStatistics. The primary statistics are:

bytesAllowed	Total number of bytes allowed in this region.
byteCount	Number of bytes in region
lruEvictions	Total number of entry evictions triggered by LRU.
lruDestroys	Number of entry destroys triggered by LRU.
lruDestroysLimit	Maximum number of entry destroys triggered by LRU before a scan occurs.
lruEvaluations	Number of entries evaluated during LRU operations.
lruGreedyReturns	Number of non-LRU entries evicted during LRU operations.

Pool Statistics

These statistics are in a client and they describe one of the client's connection pools. The primary statistics are:

INITIAL_CONTACTS	Number of contacts initially made the user.
KNOWN_LOCATORS	Current number of locators discovered.
ENDPOINTS_KNOWN	Current number of servers discovered.
QUEUE_SERVERS	Number of servers hosting this client's subscription queue.
REQUESTS_TO_LOCATOR	Number of requests from this connection pool to a locator.
RESPONSES_FROM_LOCATOR	Number of responses from the locator to this connection pool.
connections	Current number of connections.
connects	Total number of times a connection has been created.
disconnects	Total number of times a connection has been destroyed.
minPoolSizeConnects	Total number of connects done to maintain minimum pool size.
lifetimeConnects	Total number of connects done due to lifetime expiration.
idleDisconnects	Total number of disconnects done due to idle expiration.
lifetimeDisconnects	Total number of disconnects done due to lifetime expiration.
idleChecks	Total number of checks done for idle expiration.
lifetimeChecks	Total number of checks done for lifetime expiration.
lifetimeExtensions	Total number of times a connection's lifetime has been extended because the servers are still balanced.
connectionWaitsInProgress	Current number of threads waiting for a connection.
connectionWaits	Total number of times a thread completed waiting for a connection (either by timing out or by getting a connection).
connectionWaitTime	Total time, in nanoseconds, spent waiting for a connection.

Process Statistics – Linux

Operating system statistics on the VM's process. These can be used to determine the member's CPU, memory, and disk usage. Operating system statistics are not available in *pure Java* mode, where GemFire Enterprise runs without the use of the GemFire native library.

These are the equivalent of SolarisProcessStats when we're running on Linux. The primary statistics are:

imageSize	Size, in megabytes, of the process's image.
rssSIze	Size, in megabytes, of the process's resident size.

Process Statistics – Solaris

perating system statistics on the VM's process. These can be used to determine the member's CPU, memory, and disk usage. Operating system statistics are not available in *pure Java* mode, where GemFire Enterprise runs without the use of the GemFire native library.

For the Solaris operating system, when not using pure-java mode, these statistics are gathered for every process. The primary statistics are:

`allOtherSleepTime`	The number of milliseconds the process has been sleeping for some reason not tracked by any other stat. Note, all light-weight processes (lwps) contribute to this stat's value, so check `lwpCurCount` to understand large values.
`characterIo`	The number of characters read and written.
`dataFaultSleepTime`	The number of milliseconds the process has been faulting in data pages.
`heapSize`	The size, in megabytes, of the process's heap.
`imageSize`	The size, in megabytes, of the process's image.
`involContextSwitches`	The number of times the process operation was forced to do a context switch.
`kernelFaultSleepTime`	The number of milliseconds the process has been faulting in kernel pages.
`lockWaitSleepTime`	The number of milliseconds the process has been waiting for a user lock. Note, all lwp's contribute to this stat's value, so check `lwpCurCount` to understand large values.
`lwpCurCount`	The current number of lightweight process threads that exist in the process.
`lwpTotalCount`	The total number of lightweight process threads that have ever contributed to the process's statistics.
`majorFaults`	The number of times the process operation has had a page fault that needed disk access.
`messagesRecv`	The number of messages received by the process.
`messagesSent`	The number of messages sent by the process.
`minorFaults`	The number of times the process operation has had a page fault that did not need disk access.
`rssSize`	The size, in megabytes of the process's resident set size.
`signalsReceived`	The total number of operating system signals this process has received.
`systemCalls`	The total number system call operations done by this process.
`stackSize`	The size, in megabytes, of the process's stack.
`stoppedTime`	The amount of time, in milliseconds, the process has been stopped.
`systemTime`	The amount it time, in milliseconds, the process has been using the CPU to execute system calls.
`textFaultSleepTime`	The amount of time, in milliseconds, the process has been faulting in text pages.
`trapTime`	The amount of time, in milliseconds, the process has been in system traps.
`userTime`	The amount of time, in milliseconds, the process has been using the CPU to execute user code.
`volContextSwitches`	The number of voluntary context switch operations done by the process.

`waitCpuTime`	The amount of time, in milliseconds, the process has been waiting for a CPU due to latency.
`activeTime`	The amount of time, in milliseconds, the process has been using the CPU to execute user or system code.
`cpuUsed`	The percentage of recent CPU time used by the process.
`memoryUsed`	The percentage of real memory used by the process.

Process Statistics – Windows

Operating system statistics on the VM's process. These can be used to determine the member's CPU, memory, and disk usage. Operating system statistics are not available in *pure Java* mode, where GemFire Enterprise runs without the use of the GemFire native library.

These are the equivalent of SolarisProcessStats when running on Windows. The primary statistics are:

`handles`	The total number of handle items currently open by this process. This number is the sum of the handles currently open by each thread in this process.
`priorityBase`	The current base priority of the process. Threads within a process can raise and lower their own base priority relative to the process's base priority.
`threads`	Number of threads currently active in this process. An instruction is the basic unit of execution in a processor, and a thread is the object that executes instructions. Every running process has at least one thread.
`activeTime`	The elapsed time, in milliseconds, that all of the threads of this process used the processor to execute instructions. An instruction is the basic unit of execution in a computer, a thread is the object that executes instructions, and a process is the object created when a program is run. Code executed to handle some hardware interrupts and trap conditions are included in this count.
`pageFaults`	The total number of page fault operations by the threads executing in this process. A page fault occurs when a thread refers to a virtual memory page that is not in its working set in main memory. This will not cause the page to be fetched from disk if it is on the standby list and hence already in main memory, or if it is in use by another process with whom the page is shared.
`pageFileSize`	The current number of bytes this process has used in the paging file(s). Paging files are used to store pages of memory used by the process that are not contained in other files. Paging files are shared by all processes, and lack of space in paging files can prevent other processes from allocating memory.
`pageFileSizePeak`	The maximum number of bytes this process has used in the paging file(s). Paging files are used to store pages of memory used by the process that are not contained in other files. Paging files are shared by all processes, and lack of space in paging files can prevent other processes from allocating memory.
`privateSize`	The current number of bytes this process has allocated that cannot be shared with other processes.

`systemTime`	The elapsed time, in milliseconds, that the threads of the process have spent executing code in privileged mode. When a Windows system service is called, the service will often run in Privileged Mode to gain access to system-private data. Such data is protected from access by threads executing in user mode. Calls to the system can be explicit or implicit, such as page faults or interrupts. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. These subsystem processes provide additional protection. Therefore, some work done by Windows on behalf of your application might appear in other subsystem processes in addition to the privileged time in your process.
`userTime`	The elapsed time, in milliseconds, that this process's threads have spent executing code in user mode. Applications, environment subsystems, and integral subsystems execute in user mode. Code executing in User Mode cannot damage the integrity of the Windows Executive, Kernel, and device drivers. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. These subsystem processes provide additional protection. Therefore, some work done by Windows on behalf of your application might appear in other subsystem processes in addition to the privileged time in your process.
`virtualSize`	Virtual Bytes is the current size in bytes of the virtual address space the process is using. Use of virtual address space does not necessarily imply corresponding use of either disk or main memory pages. Virtual space is finite, and by using too much, the process can limit its ability to load libraries.
`virtualSizePeak`	The maximum number of bytes of virtual address space the process has used at any one time. Use of virtual address space does not necessarily imply corresponding use of either disk or main memory pages. Virtual space is however finite, and by using too much, the process might limit its ability to load libraries.

`workingSetSize`	The current number of bytes in the Working Set of this process. The Working Set is the set of memory pages touched recently by the threads in the process. If free memory in the computer is above a threshold, pages are left in the Working Set of a process even if they are not in use. When free memory falls below a threshold, pages are trimmed from Working Sets. If they are needed they will then be soft-faulted back into the Working Set before they are paged out to disk.
`workingSetSizePeak`	The maximum number of bytes in the Working Set of this process at any point in time. The Working Set is the set of memory pages touched recently by the threads in the process. If free memory in the computer is above a threshold, pages are left in the Working Set of a process even if they are not in use. When free memory falls below a threshold, pages are trimmed from Working Sets. If they are needed they will then be soft-faulted back into the Working Set before they leave main memory.

Resource Manager Statistics

Statistics related to the GemFire resource manager. Use these to help analyze and tune your VM memory settings and the GemFire resource-manager settings.

`criticalThreshold`	The cache `resource-manager` setting `critical-heap-percentage`..
`heapCriticalEvents`	Number of times incoming cache activities were blocked due to heap use going over the critical threshold.
`heapSafeEvents`	Number of times incoming cache activities were unblocked due to heap use going under the critical threshold.
`evictionThreshold`	The cache `resource-manager` setting `eviction-heap-percentage`..
`evictionStartEvents`	Number of times eviction activities were started due to the heap use going over the eviction threshold.
`evictionStopEvents`	Number of times eviction activities were stopped due to the heap use going below the eviction threshold.
`tenuredHeapUsed`	Percentage of tenured heap currently in use.

StatSampler

These statistics show how much time is spent collecting statistics.

`sampleCount`	Total number of samples taken by this sampler.
`sampleTime`	Total amount of time spent taking samples.

System Statistics – Linux

Operating system statistics on the member's machine. These can be used to determine total cpu, memory, and disk usage on the machine. Operating system statistics are not available in pure Java mode.

These are the equivalent of `SolarisSystemStats` when running on Linux. The primary statistics are:

`allocatedSwap`	Number of megabytes of swap space that have actually been written to. Swap space must be reserved before it can be allocated.
`bufferMemory`	Number of megabytes of memory allocated to buffers.
`contextSwitches`	Total number of context switches from one thread to another on the computer. Thread switches can occur either inside of a single process or across processes. A thread switch may be caused either by one thread asking another for information, or by a thread being preempted by another, higher priority thread becoming ready to run.
`cpuActive`	Percentage of the total available time that has been used in a non-idle state.
`cpuIdle`	Percentage of the total available time that has been spent sleeping.
`cpuNice`	Percentage of the total available time that has been used to execute user code in processes with low priority.
`cpuSystem`	Percentage of the total available time that has been used to execute system (that is, kernel) code.
`cpuUser`	Percentage of the total available time that has been used to execute user code.
`cpus`	Number of online CPUs (items) on the local machine.
`freeMemory`	Number of megabytes of unused memory on the machine.
`pagesPagedIn`	Total number of pages that have been brought into memory from disk by the operating system's memory manager.
`pagesPagedOut`	Total number of pages that have been flushed from memory to disk by the operating system's memory manager.
`pagesSwappedIn`	Total number of swap pages that have been read in from disk by the operating system's memory manager.
`pagesSwappedOut`	Total number of swap pages that have been written out to disk by the operating system's memory manager.
`physicalMemory`	Actual amount of total physical memory on the machine.
`processCreates`	The total number of times a process (operation) has been created.
`processes`	Number of processes in the computer at the time of data collection. Notice that this is an instantaneous count, not an average over the time interval. Each process represents the running of a program.
`sharedMemory`	Number of megabytes of shared memory on the machine.
`unallocatedSwap`	Number of megabytes of swap space that have not been allocated.
`loopbackPackets`	Number of network packets sent (or received) on the loopback interface.
`loopbackBytes`	Number of network bytes sent (or received) on the loopback interface.

`recvPackets`	Total number of network packets received (excluding loop-back).
`recvBytes`	Total number of network bytes received (excluding loop-back).
`recvErrors`	Total number of network receive errors.
`recvDrops`	Total number network receives (packets) dropped.
`xmitPackets`	Total number of network packets transmitted (excluding loopback.
`xmitBytes`	Total number of network bytes transmitted (excluding loop-back).
`xmitErrors`	Total number of network transmit errors.
`xmitDrops`	Total number of network transmits (packets) dropped.
`xmitCollisions`	Total number of network transmit collisions.
`loadAverage1`	Average number of threads in the run queue or waiting for disk I/O over the last minute.
`loadAverage15`	Average number of threads in the run queue or waiting for disk I/O over the last fifteen minutes.
`loadAverage5`	Average number of threads in the run queue or waiting for disk I/O over the last five minutes.

System Statistics – Solaris

Operating system statistics on the member's machine. These can be used to determine total cpu, memory, and disk usage on the machine. Operating system statistics are not available in pure Java mode.

These statistics are recorded for the machine on which the program is running when not using pure Java and running on Solaris. The primary statistics are:

`allocatedSwap`	The number of megabytes of swap space that have actually been written to. Swap space must be reserved before it can be allocated.
`cpuActive`	The percentage of the total available time that has been used to execute user or system code.
`cpuIdle`	The percentage of the total available time that has been spent sleeping.
`cpuIoWait`	The percentage of the total available time that has been spent waiting for disk IO to complete.
`cpuSwapWait`	The percentage of the total available time that has been spent waiting for paging and swapping to complete.
`cpuSystem`	The percentage of the total available time that has been used to execute system (that is, kernel) code.
`cpuUser`	The percentage of the total available time that has been used to execute user code.
`cpuWaiting`	The percentage of the total available time that has been spent waiting for IO, paging, or swapping.
`cpus`	The number of online CPUs on the local machine.
`freeMemory`	The number of megabytes of unused memory on the machine.

physicalMemory	The actual amount of total physical memory on the machine.
processes	The number of processes in the computer at the time of data collection. Notice, this is an instantaneous count, not an average over the time interval. Each process represents the running of a program
reservedSwap	The number of megabytes of swap space reserved for allocation by a particular process.
schedulerRunCount	The total number of times the system scheduler has put a thread in its run queue.
schedulerSwapCount	The total number of times the system scheduler has swapped out an idle process.
schedulerWaitCount	The total number of times the system scheduler has removed a thread from the run queue because it was waiting for a resource.
unreservedSwap	The number of megabytes of swap space that are free. If this value goes to zero new processes can no longer be created.
unallocatedSwap	The number of megabytes of swap space that have not been allocated.
anonymousPagesFreed	The total number pages that contain heap, stack, or other changeable data that have been removed from memory and added to the free list.
anonymousPagesPagedIn	The total number pages that contain heap, stack, or other changeable data that have been allocated in memory and possibly copied from disk.
anonymousPagesPagedOut	The total number pages that contain heap, stack, or other changeable data that have been removed from memory and copied to disk.
contextSwitches	The total number of context switches from one thread to another on the computer. Thread switches can occur either inside of a single process or across processes. A thread switch may be caused either by one thread asking another for information, or by a thread being preempted by another, higher priority thread becoming ready to run.
execPagesFreed	The total number read only pages that contain code or data that have been removed from memory and returned to the free list.
execPagesPagedIn	The total number read only pages that contain code or data that have been copied from disk to memory.
execPagesPagedOut	The total number read only pages that contain code or data that have been removed from memory and will need to be paged in when used again.
failedMutexEnters	The total number of times a thread entering a mutex had to wait for the mutex to be unlocked.
failedReaderLocks	The total number of times readers failed to obtain a readers/writer locks on their first try. When this happens the reader must wait for the current writer to release the lock.

failedWriterLocks	The total number of times writers failed to obtain a readers/writer locks on their first try. When this happens the writer must wait for all the current readers or the single writer to release the lock.
fileSystemPagesFreed	The total number of pages, that contained the contents of a file due to the file being read from a file system, that have been removed from memory and put on the free list.
fileSystemPagesPagedIn	The total number of pages that contain the contents of a file due to the file being read from a file system.
fileSystemPagesPagedOut	The total number of pages, that contained the contents of a file due to the file being read from a file system, that have been removed from memory and copied to disk.
hatMinorFaults	The total number of hat faults. You only get these on systems with software memory management units.
interrupts	The total number of interrupts that have occurred on the computer.
involContextSwitches	The total number of times a thread was forced to give up the CPU even though it was still ready to run.
majorPageFaults	The total number of times a page fault required disk IO to get the page.
messageCount	The total number of `msgrcv` and `msgsnd` system calls.
pageDaemonCycles	The total number of revolutions of the page daemon's scan "clock hand".
pageIns	The total number of times pages have been brought into memory from disk by the operating system's memory manager.
pageOuts	The total number of times pages have been flushed from memory to disk by the operating system's memory manager.
pagerRuns	The total number of times the pager daemon has been scheduled to run.
pagesPagedIn	The total number of pages that have been brought into memory from disk by the operating system's memory manager.
pagesPagedOut	The total number of pages that have been flushed from memory to disk by the operating system's memory manager.
pagesScanned	The total number pages examined by the pageout daemon. When the amount of free memory gets below a certain size, the daemon starts to look for inactive memory pages to steal from processes. A high scan rate is a good indication of needing more memory.
procsInIoWait	The number of processes waiting for block I/O at this instant in time.
protectionFaults	The total number of times memory has been accessed in a way that was not allowed. This results in a segmentation violation and in most cases a core dump.
semphoreOps	The total number of semaphore operations.
softwareLockFaults	The total number of fault operations caused by software locks held on memory pages.

systemCalls	The total number of fault operations caused by software locks held on memory pages.
systemMinorFaults	The total number of minor page fault operations in kernel code. Minor page faults do not require disk access.
threadCreates	The total number of times a thread operation has been created.
traps	The total number of trap operations that have occurred on the computer.
userMinorFaults	The total number of minor page fault operations in non-kernel code. Minor page faults do not require disk access.
loopbackInputPackets	The total number of input packets received over the loopback network adaptor.
loopbackOutputPackets	The total number of output packets sent over the loopback network adaptor.
inputPackets	Packets received.
inputErrors	Input errors.
outputPackets	Solaris out packets.
outputErrors	Output errors.
collisions	Solaris collisions.
inputBytes	Octets received.
outputBytes	Octats transmitted.
multicastInputPackets	Multicast packets received.
multicastOutputPackets	Multicast packets requested to be sent.
broadcastInputPackets	Broadcast packets received.
broadcastOutputPackets	Broadcast packets requested to be sent.
inputPacketsDiscarded	Number of received packets discarded.
outputPacketsDiscarded	Packets that could not be sent up because the queue was flow controlled.
loadAverage1	The average number of threads ready to run over the last minute.
loadAverage15	The average number of threads ready to run over the last 15 minutes.
loadAverage5	The average number of threads ready to run over the last five minute.

VM Statistics

Show the VM's Java usage and can be used to detect possible problems with memory consumption. These statistics are recorded from `java.lang.Runtime` under `VMStats`. The primary statistics are:

cpus	Number of CPUs available to the Java VM on its machine.
daemonThreads	Current number of live daemon threads in this VM.
fdLimit	Maximum number of file descriptors.
fdsOpen	Current number of open file descriptors.

`freeMemory`	An approximation for the total amount of memory, measured in bytes, currently available for future allocated objects.
`loadedClasses`	Total number of classes loaded since the VM started.
`maxMemory`	The maximum amount of memory, measured in bytes, that the VM will attempt to use.
`peakThreads`	High water mark of live threads in this VM.
`pendingFinalization`	Number of objects that are pending finalization in the java VM.
`processCpuTime`	CPU time, measured in nanoseconds, used by the process.
`threads`	Current number of live threads (both daemon and non-daemon) in this VM.
`threadStarts`	Total number of times a thread has been started since this VM started.
`totalMemory`	The total amount of memory, measure in bytes, currently available for current and future objects.
`unloadedClasses`	Total number of classes unloaded since the VM started.

VMGC Statistics

These statistics show how much time used by different VM garbage collection and are available on JDK 1.5 and later VMs. The primary statistics are:

`collections`	Total number of collections this garbage collector has done.
`collectionTime`	Approximate elapsed time spent doing collections by this garbage collector.

VM Memory Usage Statistics

Show details on how the Java heap memory is being used. This statistic is available on JDK 1.5 and later VMs. The primary statistics are:

`committedMemory`	The amount of committed memory, measured in bytes, for this area.
`initMemory`	Initial memory the VM requested from the operating system for this area.
`maxMemory`	The maximum amount of memory, measured in bytes, this area can have.
`usedMemory`	The amount of used memory, measured in bytes, for this area.

VM Memory Pool Statistics

These statistics describe memory usage in difference garbage collector memory pools. The primary statistics are:

`collectionUsageExceeded`	Total number of times the garbage collector detected that memory usage in this pool exceeded the collectionUsageThreshold.
`collectionUsageThreshold`	The collection usage threshold, measured in bytes, for this pool.

`collectionUsedMemory`	The estimated amount of used memory, measured in bytes, after that last garbage collection of this pool.
`currentCommittedMemory`	The amount of committed memory, measured in bytes, for this pool.
`currentInitMemory`	Initial memory the VM requested from the operating system for this pool.
`currentMaxMemory`	The maximum amount of memory, measured in bytes, this pool can have.
`currentUsedMemory`	The estimated amount of used memory, measured in bytes, currently in use for this pool.
`usageExceeded`	Total number of times that memory usage in this pool exceeded the usageThreshold.
`usageThreshold`	The usage threshold, measured in bytes, for this pool.

Cache Performance Statistics Related to Transactions

During the operation of GemFire cache transactions, if enabled, the following statistics are compiled and stored as properties in the `CachePerfStats` statistic resource. Because the transaction's data scope is the cache, these statistics are collected on a per-cache basis.

> *For performance reasons, time-based statistics are disabled by default. To enable, set the gemfire property,* `enable-time-statistics` (page 51)*, to* `true`*. Note that sampling and archiving must also be enabled for this to take effect.*

`txCommits`	Total number of times a transaction commit has succeeded.
`txFailures`	Total number of times a transaction commit has failed.
`txRollbacks`	Total number of times a transaction has been explicitly rolled back.
`txSuccessLifeTime`	The total amount of time, in nanoseconds, spent in a transaction before a successful commit. The time measured starts at transaction begin and ends when commit is called.
`txFailedLifeTime`	The total amount of time, in nanoseconds, spent in a transaction before a failed commit. The time measured starts at transaction begin and ends when commit is called.
`txRollbackLifeTime`	The total amount of time, in nanoseconds, spent in a transaction before an explicit rollback. The time measured starts at transaction begin and ends when rollback is called.
`txCommitTime`	The total amount of time, in nanoseconds, spent doing successful transaction commits.
`txFailureTime`	The total amount of time, in nanoseconds, spent doing failed transaction commits.
`txRollbackTime`	The total amount of time, in nanoseconds, spent doing explicit transaction rollbacks.
`txCommitChanges`	Total number of changes made by committed transactions.
`txFailureChanges`	Total number of changes lost by failed transactions.
`txRollbackChanges`	Total number of changes lost by explicit transaction rollbacks.
`txConflictCheck Time`	The total amount of time, in nanoseconds, spent doing conflict checks during transaction commit.

Event Queue Statistics From Server-to-Client Communication

The following statistics track event messages queued on the server to be sent to the client. The statistics are gathered for each client subscription queue and are incremental for the lifetime of the queue. The event messages are referred to as events in these statistics.

`eventsQueued`	Number of events placed in the subscription queue.
`eventsConflated`	Number of events conflated. If this is high, the server's dispatcher may be running slowly. This could be caused by one or more slow client's causing blocking in their subscription queues.
`eventsRemoved`	Number of events removed from the subscription queue.
`eventsTaken`	Number of events taken from the subscription queue.

`eventsExpired`	Number of events that have expired while in the subscription queue. If this is high on a secondary server, it might be that the `MessageSyncInterval` on the primary is set too high, causing the secondary to fall behind in event cleanup.
`eventsRemovedByQrm`	Number of events removed based on a message sent from the primary. Only incremented while the subscription queue is in a secondary server.
`numVoidRemovals`	Number of events which were supposed to be destroyed from the subscription queue through `remove` but were removed by some other operation like conflation or expiration.
`numSequenceViolated`	Number of events that had sequence ID less than or equal to the last sequence ID. The system assumes these events are duplicates and does not add them to the subscription queue. A non-zero value may indicate message loss.
`threadIdentifiers`	Number of `ThreadIdentifier` objects (units) in the subscription queue.

Partitioned Region Statistics

Whenever you run members that contain partitioned region buckets, GemFire gathers statistics specific to the operation of partitioned regions. Statistics are gathered for each member. Partitioned region statistics provide run-time and historical data on these areas:

- Region operations executed on the member
- Message traffic to and from this member requesting operations on the partitioned region
- Data entry distribution among the buckets hosted by this member

Partitioned region statistics are gathered for each member. Time-based partitioned region statistics, like all other time-based GemFire statistics, are disabled by default. For details on turning on time-based statistics, see *GemFire Enterprise System Statistics* on page 238. Otherwise, partitioned region statistics are always enabled.

Unlike the transient GemFire region statistics, partitioned region statistics can be archived and charted. You can view them using VSD.

Partitioned region statistics are archived in the `statArchive.gfs` file. To view them through VSD, in the top pane select the statistics category containing the partitioned region name in this format:

`PartitionedRegion`*partitioned_region_name*`Statistics.gfs`

For example, every member that participates in the partitioned region named `STOCK` generates a category of statistics named `PartitionedRegionSTOCKStatistics`.

Statistics on Region Operations

These statistics track the standard region operations executed in the member. Operations can originate locally or in a request from a remote member.

Unsuccessful operations are not counted in these statistics.

`containsKeyCompleted`	Number of successful `containsKey` operations in this member.
`containsKeyOpsRetried`	Number of `containsKey` or `containsValueForKey` operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`containsKeyRetries`	Total number of times `containsKey` or `containsValueForKey` operations were retried. If multiple retries are required on a single operation, this stat counts them all.
`containsKeyTime`	Total time, in nanoseconds, the member spent doing `containsKey` operations in this member.
`containsValueForKeyCompleted`	Number of successful `containsValueForKey` operations in this member.
`containsValueForKeyTime`	Total time, in nanoseconds, the member spent doing `containsValueForKey` operations in this member.
`createOpsRetried`	Number of `create` operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`createRetries`	Total number of times `create` operations were retried. If multiple retries are required on a single operation, this stat counts them all.

createsCompleted	Number of successful create operations in this member.
createTime	Total time, in nanoseconds, the member spent doing create operations in this member.
destroyOpsRetried	Number of destroy operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
destroyRetries	Total number of times destroy operations were retried. If multiple retries are required on a single operation, this stat counts them all.
destroysCompleted	Number of successful destroy operations in this member.
destroyTime	Total time, in nanoseconds, the member spent doing destroy operations in this member.
getOpsRetried	Number of get operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
getEntriesCompleted	Number of get entry operations completed.
getEntriesTime	Total time, in nanoseconds, spent performing get entry operations.
getRetries	Total number of times get operations were retried. If multiple retries are required on a single operation, this stat counts them all.
getsCompleted	Number of successful get operations in this member.
getTime	Total time, in nanoseconds, the member spent doing get operations in this member.
sentMessageMaxTime	Longest amount of time, in milliseconds, taken to write a message to the network before a forced disconnect occurs. This stat is always active regardless of the setting of the enable-time-statistics gemfire.properties setting.
replyWaitMaxTime	Longest amount of time, in milliseconds, taken to write a message and receive a reply before a forced disconnect occurs. This stat is always active regardless of the setting of the enable-time-statistics gemfire.properties setting.
invalidatesCompleted	Number of successful invalidate operations in this member.
invalidateOpsRetried	Number of invalidate operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
invalidateRetries	Total number of times invalidate operations were retried. If multiple retries are required on a single operation, this stat counts them all.
invalidateTime	Total time, in nanoseconds, the member spent doing invalidate operations in this member.
putOpsRetried	Number of put operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.

putRetries	Total number of times put operations were retried. If multiple retries are required on a single operation, this stat counts them all.
putsCompleted	Number of successful put operations in this member.
putTime	Total time, in nanoseconds, the member spent doing put operations in this member.

Statistics on Partition Messages

These statistics track the incoming and outgoing message traffic generated by requests for operations on this partitioned region.

> *Unsuccessful operations and local operations —those that originated in this member—are not counted in these statistics.*

partitionMessagesProcessed	Number of region operations executed in this member at the request of other data hosts for the region.
partitionMessagesProcessingTime	Total time, in nanoseconds, the member spent executing region operations in this member at the request of remote members.
partitionMessagesReceived	Number of remote requests this member received for any region operation in this member.
partitionMessagesSent	Number of requests this member sent for any region operation on a remote member.
prMetaDataSentCount	Number of times meta data refresh sent on client's request. Used with pr-single-hop functionality.

Statistics on Data Entry Caching

These statistics track the pattern of data entry distribution among the buckets in this member.

avgBucketSize	Average number of entries for each of the primary buckets in this member.
bucketCount	Total number of buckets in this member.
bucketCreationsCompleted	Number of logical bucket creation operations requests completed after which the bucket was created.
bucketCreationsTime	Total time, in nanoseconds, spent waiting for bucket creation requests to complete after which the bucket was created.
bucketCreationsDiscoveryCompleted	Number of bucket creation operations requests completed after which it was discovered that the bucket was created by another VM.
bucketCreationsDiscoveryTime	Total time, in nanoseconds, spent waiting for bucket creation requests to complete after which it was discovered that the bucket was created by another VM.
dataStoreBytesInUse	The number of bytes stored in this cache for the named partitioned region.
dataStoreEntryCount	Total number of entries in all the buckets in this member.
maxBucketSize	Largest number of entries in the primary buckets in this member.

`minBucketSize`	Smallest number of entries in the primary buckets in this member.
`totalBucketSize`	Total number of entries in the primary buckets.

Statistics on Redundancy

These statistics track status on partitioned region data copies.

`configuredRedundantCopies`	This is equivalent to the `PartitionAttributes.getRedundantCopies()` configuration that was used to create this partitioned region. This value remains unchanged for a given partitioned region.

`actualRedundantCopies`

The least current redundant number of copies for any data in this partitioned region (there may be some data that is fully redundant, but some data will have only this number of copies).

- this value may drop, when a datastore is lost, or rise when a datastore is added.
- this value may drop temporarily during partitioned region creation or destruction and then rise again.
- if this value remains low, then partitioned region data is at risk and may be lost if another datastore is lost.
- a healthy partitioned region will maintain a value equal to `configuredRedundantCopies`.
- the user should add one or more datastores if the value remains low.
- high-availability may result in a brief fluctuation, but it should return to a value equal to `configuredRedundantCopies` if there are sufficient datastores present (that is, killing one datastore will cause the data hosted on it to failover to another datastore).

`lowRedundancyBucketCount`

The number of buckets in this partitioned region that currently have fewer copies than the `configuredRedundantCopies` for this partitioned region.

- this value may rise above zero when a datastore is lost and return to zero when one or more datastores are added.
- this value may rise temporarily during partitioned region creation or destruction and then return to zero.
- if this value remains above zero, then partitioned region data is at risk and may be lost if another datastore is lost.
- this value will be above zero whenever `actualRedundantCopies` is less than `configuredRedundantCopies`.
- a healthy partitioned region will maintain a value of zero.
- the user should add one or more datstores if this value remains above zero.
- high-availability may result in a brief fluctuation, but it should return to zero if there are sufficient datastores present (that is, killing one datastore will cause the data hosted on it to failover to another datastore).

Glossary

ACK wait threshold A time-to-wait for message acknowledgement between system members.

administrative event See entry key.

API **A**pplication **P**rogramming **I**nterface. GemFire provides APIs to cached data for Java applications.

application program A program designed to perform a specific function directly for the user or, in some cases, for another application program. GemFire applications use the GemFire application programming interfaces (APIs) to modify cached data.

attribute Querying: A named member of a data object. The public fields and methods of an object may be accessed as attributes in the context of a query.

Region: See region attributes.

attribute path A sequence of attributes separated by a dot (.), applied to objects where the value of each attribute is used to apply the next attribute.

blocking A behavior associated with synchronization functions. Blocking behavior is exhibited as waiting for a signal to proceed, regardless of how long it takes. See also timeout.

cache In-memory GemFire data storage created by an application or cache server for data storage, distribution, and management. This is the point of access for Java applications for all caching features, and the only view of the cache that is available to the application. Cache creation creates a connection to the distributed system. See also local and remote.

cache-local Residing or occurring in the local cache.

cache.xml Common name for the XML file that declares the initial configuration of a cache. This file is used to customize the behavior of the GemFire cache server process and can be used by any Java application. Applications can also configure the cache through the GemFire Java APIs. You can give this file any name.

cache event See entry key.

cache listener User-implemented plug-in for receiving and handling region entry events. A region's cache listener is called after an entry in the local cache is modified. See also cache writer.

cache loader User-implemented plug-in for loading data into a region. A region's cache loader is used to load data that is requested of the region but is not available in the distributed system. For a distributed region, the loader that is used can be in a different cache from the one where the data-request operation originated. See also netSearch and netLoad.cache misses, where a requested key is not present or has a null value in the local cache.

cache miss The situation where an key's value is requested from a cache and the requested key is not present or has a null value. GemFire responds to cache misses in various ways, depending on the region and system configuration. For example, a client region goes to its servers to satisfy cache misses. A region with local scope uses its data loader to load the value from an outside data source, if a loader is installed on the region.

cache server A long-lived, configurable GemFire distributed system member process.

cache transaction A native GemFire transaction, managed by GemFire and not by JTA. This type of transaction operates only on data available from the GemFire cache in the local member. See also JTA and global transaction.

cache writer User-implemented plug-in intended for synchronizing the cache with an outside data source. A region's cache writer is a synchronous listener to cache data events. The cache writer has the ability to abort a data modification. See also cache listener and netWrite.

client A GemFire application that is configured as a standalone distributed system member, with regions configured as client regions. Client configuration uses the `<client-cache>` `cache.xml` element and the `ClientCache` API.

client region A GemFire cache region that is configured to go to one or more GemFire servers, in a separate GemFire distributed system, for all data distribution activities. Among other things, client regions go to servers to satisfy cache misses, distribute data modifications, and to run single queries and continuous queries.

collection Used in the context of a query for a group of distinct objects of homogeneous type, referred to as elements. Valid collections include the `java.util.Collection` as well as `Set`, `Map`, `List`, and arrays. The elements in a collection can be iterated over. Iteration over a `Map` traverses its entries as instances of `Map.Entry`. A region can also be treated as a collection of its values. See also QRegion.

commit A transactional operation that merges a transaction's result into the cache. Changes are made in an "all or none" fashion. Other changes from outside the current transaction are kept separate from those being committed.

concurrency-level Region attribute that specifies an estimate of the number of threads ever expected to concurrently modify values in the region. The actual concurrency may vary; this value is used to optimize the allocation of system resources.

conflation Combining entries in a message queue for better performance. When an event is added to queue, if a similar event exists in the queue, there are two ways to conflate the events. One way is to remove the existing entry from wherever it resides in the queue, and add the new entry to the end of the queue. The other way is to replace the existing entry with the new entry, where it resides in the queue, and add nothing to the end of the queue. In GemFire, region entry update events, server events going to clients, and gateway events going to remote distributed systems can all be conflated.

connection The connection used by an application to access a GemFire Enterprise system. A Java application connects to its GemFire distributed system when it creates its cache. The

application must connect to a distributed system to gain access to the GemFire functionalities. A client connects to a running GemFire server to distribute data and events between itself and the server tier. These client connections are managed by server connection pools within the client applications. Gateways connect to remote site GemFire gateway hubs to distribute data events between sites.

consumer GemFire member process that receives data and/or events from other members. Peer consumers are often configured with replicated regions, so all changes in the distributed system arrive into the local cache. Client consumers can register subscriptions with their servers so that updates are automatically forwarded from the server tier. See producer.

coordinator The member of the distributed system that sends out membership views. This is typically the locator in GemFire.

data region A logical grouping of data within a cache. Regions usually contain data entries (see **entry**). Each region has a set of region attributes governing activities such as expiration, distribution, data loading, events, and capacity control. In addition, a region can have an application-defined user attribute.

data accessor In the context of a region, a member configured to use a region, but not store any data for it in the member's local cache. Common use cases for data accessors are thin clients, and thin producer and consumer applications. Accessors can put data into the region and receive events for the region from remote members or servers, but they store no data in the application. See also data host.

data entry See entry.

data host In the context of a region, a member configured to store data for the region. Members that do not store data are commonly referred to as data accessors. This is used mostly for partitioned regions, where data is spread across the distributed system among the data hosts.

data region See region.

data-policy Region attribute used to determine what events the region receives from remote caches, whether data is stored in the local cache, and whether the data is persisted to disk. For disk persistence, writes are performed according to the cache disk-store configuration.

deadlock A situation in which two or more processes are waiting indefinitely for events that will never occur.

destroy Distributed**:** To remove a cached object across the distributed cache.

Local**:** To remove a cached object from the local cache only.

disk region A persistent region.

disk-store Cache element specifying location and write behavior for disk storage. Used for persistence and overflow of data. The cache can have multiple disk stores, which are specified by name for region attributes, client subscription queues (for servers), and wan gateway queues.

distributed cache A collection of caches spread across multiple machines and multiple locations that functions as a single cache for the individual applications.

distributed system One or more GemFire system members that have been configured to communicate with each other, forming a single, logical system.

distributed-ack scope Data distribution setting that causes synchronous distribution operations, which wait for acknowledgement from other caches before continuing. Operations from multiple caches can arrive out of order. This scope is slower but more reliable than `distributed-no-ack`.

distributed-no-ack-scope Data distribution setting that causes asynchronous distribution operations, which return without waiting for a response from other caches. This scope produces the best performance, but is prone to race conditions.

entry A data object in a region consisting of a key and a value. The value is either null (invalid) or a Java object. A region entry knows what region it is in. An entry can have an application-defined user attribute. See also region data, entry key, and entry value.

entry key The unique identifier for an entry in a region.

entry value The data contained in an entry.

event An action recognized by the GemFire system members, which can respond by executing callback methods. The GemFire API produces two types of events: cache events for detail-level management of applications with data caches and administrative events for higher-level management of the distributed system and its components. An operation can produce administrative events, cache events, or both.

eviction-attributes Region attribute that causes the cache to limit the size of the region by removing old entries to make space for new ones.

expiration A cached object expires when its time-to-live or idle timeout counters are exhausted. A region has one set of expiration attributes for itself and one set for all of its entries.

expiration action The action to be taken when a cached object expires. The expiration action specifies whether the object is to be invalidated or destroyed and whether the action is to be performed only in the local cache or throughout the distributed system. A destroyed object is completely removed from the cache. A region is invalidated by invalidating all entries contained in the region. An entry is invalidated by having its value marked as invalid. `Region.getEntry.getValue` returns `null` for an invalid entry.

In GemFire, expiration attributes are set at the region level for the region and at the entry level for entries. See also idle timeout and time-to-live.

factory method An interface for creating an object which at creation time can let its subclasses decide which class to instantiate. The factory method helps instantiate the appropriate subclass by creating the correct object from a group of related classes.

forced disconnect Forcible removal of a member from membership without the member's consent.

gateway Configured inside a gateway-hub, a gateway defines a single remote distributed system site in a multi-site installation and manages communication with the remote site. The gateway might have multiple endpoints assigned to a single remote site. The gateway also specifies queue management parameters for its endpoints.

gateway-hub GemFire caching entity that represents its distributed system in a multi-site installation. The hub manages the gateways that send messages to other distributed system sites and listens on an incoming port for connections from remote gateways.

gemfire Command-line utility that allows you to perform various GemFire management tasks from the command line, including locator start and stop, online disk store management, log management, and license information listing.

gemfire.properties Common name for the file used for distributed system configuration, including system member connection and communication behavior, logging and statistics files and settings, and security settings. Applications can also configure the distributed system through the GemFire Java APIs. You can give this file any name.

global scope Data distribution setting that provides locking across the distributed system for load, create, put, invalidate, and destroy operations on the region and its entries. This scope is the slowest, but it guarantees consistency across the distributed system.

global transaction A JTA-controlled transaction in which multiple resources, such as the GemFire cache and a JDBC database connection, participate. JTA coordinates the completion of the transaction with each of the transaction's resources. See also JTA and cache transaction.

HTTP World Wide Web's **H**yper**t**ext **T**ransfer **P**rotocol. A standard protocol used to request and transmit information over the Internet or other computer network.

idle timeout The amount of time a region or region entry may remain in the cache without being accessed before being expired. Access to an entry includes any get operation and any operation that resets the entry's time-to-live counter. Region access includes any operation that resets an entry idle timeout and any operation that resets the region's time-to-live.

Idle timeout attributes are set at the region level for the region and at the entry level for entries. See also time-to-live and expiration action.

initial capacity Region attribute. The initial capacity of the map used for storing region entries.

invalid The state of an object when the cache holding it does not have the current value of the object.

invalidate Distributed**:** To mark an object as being invalid across the distributed cache.

Local**:** To mark an object as being invalid in the local cache only.

JDBC **J**ava **D**ata**B**ase **C**onnectivity. A programming interface that lets Java applications access a database via the SQL language.

JMX **J**ava **M**anagement e**X**tensions. A set of specifications for dynamic application and network management in the J2EE development and application environment.

JNDI **J**ava **N**aming and **D**irectory **I**nterface. An interface to naming and directory services for Java applications. Applications can use JNDI to locate data sources, such as databases to use in global transactions. GemFire allows its JNDI to be configured in a `cache.xml` configuration file.

JTA **J**ava **T**ransaction **A**PI. The local Java interfaces between a transaction manager (JTS) and the parties involved in a global transaction. GemFire can be a member of a JTA global transaction. See also global transaction.

key constraint Enforcing a specific entry key type. The `key-constraint` region attribute, when set, constrains the entries in the region to keys of the specified object type.

listener An event handler. The listener registers its interest in one or more events, such as region entry updates, and is notified when the events occur.

load factor Region attribute. The load factor of the map used for storing entries.

local Local cache: The part of the distributed cache that is resident in the current member's memory. This term is used to differentiate the cache where a specific operation is being performed from other caches in the same distributed system or in another distributed system. See also remote.

Region with local scope: A region whose scope is set to local. This type of region does not distribute anything with other member's in the distributed system.

Region shortcuts: In the `RegionShortcut` and settings, `LOCAL` means the scope is set to local. All client regions have local scope. In the `ClientRegionShortcut` settings, `LOCAL` means the region does not connect to the client's servers.

local scope Data distribution setting that keeps data private and visible only to threads running within the local member. A region with local scope is completely contained in the local cache. Client regions are automatically given local scope.

locator GemFire process that tracks system members and provides current membership information to joining members so they can establish communication. For server systems, the locator also tracks servers and server load and, when a client requests a server connection, the locator sends the client to one of the least loaded servers. See also gemfire.

LRU **L**east **R**ecently **U**sed. Used to refer to region entry or entries most eligible for eviction due to lack of interest by client applications. GemFire offers eviction controllers that use the LRU status of a region's entries to determine which to evict to free up space. Possible eviction actions are local destroy and overflow. See also resource manager.

member A process that has established a connection to a distributed system. This can be a Java or Native Client application with a distributed system connection and a cache or a GemFire process such as a locator or cacheserver. The minimal GemFire process configuration is a member that is connected to a distributed system.

message queue A first-in, first-out data structure in a GemFire Enterprise system member that stores messages for distribution in the same order that the original operations happened in the local member. Each thread has its own queue. Depending on the kind of queue, the messages could be going between two members of a distributed system, a client and server, or two members in different distributed systems. See also conflation.

mirroring See replicate.

multicast A form of UDP communications where a datagram is sent to multiple processes in one network operation.

named region attributes Region attributes that are stored in the member memory and can be retrieved through their region attributes `refid` setting. GemFire provides standard predefined named region attributes, that are stored using region shortcut `refid`s. You can use any stored attributes that you wish, setting an `id` when you create them and using the id setting in the `refid` you want to use to retrieve them.

netLoad The method used by GemFire Enterprise to load an entry value into a distributed region. The `netLoad` operation invokes all remote cache loaders defined for the region until either the entry value is successfully loaded or all loaders have been tried.

netSearch The method used by GemFire Enterprise to search remote caches for a data entry that is not found in the member's local cache region. This method operates only on distributed regions.

netWrite The method used by GemFire Enterprise to invoke a cache writer for region and region entry events. This method operates only on distributed regions. For each event, if any cache writer is defined for the region, the `netWrite` operation invokes exactly one of them.

network partitioning A situation that arises from a communications partition that causes processes to become unaware of one another.

OQL **O**bject **Q**uery **L**anguage, SQL-92 extended for querying object data. GemFire supports a subset of OQL.

overflow Eviction option for eviction controllers. This causes the values of LRU entries to be moved to disk when the region reaches capacity. Writes are performed according to the cache disk-store configuration.

oplog / operation log The files in a disk-store used for the cache operations.

partitioned region A region that manages large volumes of data by partitioning it into manageable chunks and distributing it across multiple machines. Defining partition attributes or setting the region attribute data-policy to `partition` makes the region a partitioned region.

peer A GemFire member application that is not configured as a client. Peer configuration uses the `<cache>` `cache.xml` element and the `Cache` API. Peers can also be configured as servers to client applications and as gateway-hubs to remote distributed systems.

persistent region A region with the attribute data-policy set to persistent-replicate.

persistent-partition A region attribute setting identifying a region as a partitioned region whose data is persisted to disk. With persistence, all region entry keys and values are stored in an operation log on disk as well as being stored in memory. Also referred to as disk region. Writes are performed according to the cache disk-store configuration.

persistent-replicate A region attribute setting identifying a region as a replicate whose data is persisted to disk. With persistence, all region entry keys and values are stored in an operation log on disk as well as being stored in memory. Also referred to as disk region. Writes are performed according to the cache disk-store configuration.

producer A GemFire member process that puts data into the cache for consumption by other members. Producers may be configured with empty regions, where the data they put into the cache is not stored locally, but causes cache update events to be sent to other members. This is a common configuration in peer members and for client processes. See consumer.

pull model Data distribution model where each process receives update only for the data in which the process has explicitly expressed interest. In a GemFire peer member, this is accomplished using a distributed, non-replicated region and creating the data entries that are of interest in the local region. When updates happen for the region in remote caches, the only updates that are forwarded to the local cache are those for entries that are already defined in the local cache. In a GemFire client, you get pull behavior by specifically subscribing to the entries of interest. See push model.

pure Java mode Running GemFire Enterprise without the use of the GemFire native library. GemFire Enterprise can run in this mode with limited capabilities.

push model Data distribution model where each process receives updates for everything in the data set. In a GemFire peer member, this is accomplished using a replicated region. All data modifications, creations, and deletes in remote caches are pushed to the replicated region. In a GemFire client, you get push behavior by registering interest in all keys in the region. See pull model.

QRegion The region object representation in a GemFire query. A `QRegion` extends `com.gemstone.gemfire.cache.Region` and `java.util.Collection` so that the single region specification can provide access both to region attributes and to region data collections such as keys and entry values. See also collection.

query string A fully-formed SQL statement that can be passed to a query engine and executed against a data set. A query string may or may not contain a `SELECT` statement.

race condition Anomalous behavior caused by the unexpected dependence on the relative timing of events. Race conditions often result from incorrect assumptions about possible ordering of events.

range-index An XPath index optimized for range-queries with the added index maintenance expense of sorting the set of values. A range index allows faster retrieval of the set of nodes with values in a certain range. See also structure-index and value-index.

region A logical grouping of data within a cache. Regions usually contain data entries (see **entry**). Each region has a set of region attributes governing activities such as expiration, distribution, data loading, events, and capacity control. In addition, a region can have an application-defined user attribute.

region attributes The class of attributes governing the creation, distribution, and management of a region and its entries.

region data All of the entries directly contained in the region.

region entry See entry.

region shortcut Enums `RegionShortcut` and `ClientRegionShortcut` defining the main region types in GemFire for peers/servers and clients, respectively. Region shortcuts are predefined named region attributes.

remote Resident or running in a cache other than the current member's cache, but connected to the current member's cache through `GemFire`. For example, if a member does not have a data entry in the region in its local cache, it can do a `netSearch` in an attempt to retrieve the entry from the region in a remote cache within the same distributed system. Or, if the member is a client, it can send a request to a server in an attempt to retrieve the entry from the region in a remote server cache in the server's distributed system. In multi-site installations, a gateway sends events from the local cache to remote caches in other distributed systems. See also local.

replicated region A region with data-policy set to replicate or persistent-replicate.

replicate Region data-policy specification indicating to copy all distributed region data into the local cache at region creation time and to keep the local cache consistent with the distributed region data.

resource manager GemFire process that works with your VM's tenured garbage collection (GC) to control heap use and protect your VM from hangs and crashes due to memory overload. The manager prevents the cache from consuming too much memory by evicting old data and,

	if the collector is unable to keep up, by refusing additions to the cache until the collector has freed an adequate amount of memory. Eviction is done for regions configured for LRU eviction based on heap percentage. See also LRU and eviction-attributes.
role	The purpose a member fills in a distributed system, or how a member relates to other members. These optional membership roles specify the circumstances under which a member continues operation after incidents such as network failures. Members can fill one or more roles. Any number of members can be configured to satisfy the same role, and a member can be configured to play any number of roles.
rollback	A transactional operation that excludes a transaction's changes from the cache, leaving the cache undisturbed.
scope	Region attribute: In non-partitioned regions, a distribution property for data identifying whether it is distributed and, if so, whether distribution acknowledgements are required and whether distributed synchronization is required. A distributed region's cache loader and cache writer (defined in the local cache) can be invoked for operations originating in remote caches. A region that is not distributed has a local scope. See also replicate and data-policy. Querying: The data context for the part of the query currently under evaluation. The expressions in a `SELECT` statement's `FROM` clause can add to the data that is in scope in the query.
SELECT statement	A statement of the form `SELECT` *`projection_list`* `FROM` *`expressions`* `WHERE` *`expressions`* that can be passed to the query engine, parsed, and executed against data in the local cache.
serialization	The process of converting an object or object graph to a stream of bytes.
server	A GemFire member application that is configured as a peer in its own system and as a server to connecting GemFire client applications.
server group	An optional logical grouping of servers in a server distributed system. There is always the default server group made up of all available server in the server distributed system. Clients can specify the server group in their server pool configuration. Then the pool only connects to those servers. If no group is specified, the default is used.
server connection pool	The cache entity that manages client connections to servers.
socket	The application interface for TCP/IP communications. UDP provides unicast and multicast datagram sockets, while TCP provides server and connection sockets. TCP server sockets are used by server processes to create connection sockets between the server and a client.
SQL	**S**tructured **Q**uery **L**anguage.
SSL	**S**ecure **S**ocket **L**ayer. A protocol for secure communication between Java VMs.
statistics enabled	Region attribute. Specifies whether to collect statistics for the region.
struct	A data type that has a fixed number of elements, each of which has a field name and can contain an object value.
structure-index	An XPath index that is basically a pre-computed query. Any legal XPath expression can be used. The index maintains lists of all nodes that match the expression used to create it. If a

query is performed that has the same expression as the index then the result is available without XPath evaluation. See also range-index and value-index.

system member See member.

TCP The **T**ransmission **C**ontrol **P**rotocol is a part of the internet protocol (IP) suite that provides unicast communications with guaranteed delivery. The TCP protocol is connection-based, meaning that a TCP socket can only be used to send messages between one pair of processes at a time. Compare to UDP.

timeout A behavior associated with synchronization functions. Timeout behavior is exhibited as refusal to wait longer than a specified time for a signal to proceed. See also blocking.

time-to-live The amount of time a region or region entry may remain in the cache without being modified before being expired. Entry modification includes creation, update, and removal. Region modification includes creation, update, or removal of the region or of any of its entries.

Time-to-live attributes are set at the region level for the region and at the entry level for entries. See also idle timeout and expiration action.

transaction See cache transaction and global transaction.

transaction listener User-implemented plug-in for receiving and handling transaction events. A transaction listener is called after a transaction commits. See also transaction writer.

transaction writer User-implemented plug-in intended for synchronizing the cache with an outside data source. A transaction writer is a synchronous listener to cache transactions. The transaction writer has the ability to veto a transaction. See also transaction listener.

transactional view The result of a history of transactional operations for a given open transaction.

transport layer The network used to connect the GemFire system members in a GemFire Enterprise system.

TTL See time-to-live.

UDP The **U**ser **D**atagram **P**rotocol is a part of the internet protocol (IP) suite that provides simple, unreliable transmission of datagram messages from one process to another. Reliability must be implemented by applications using UDP. The UDP protocol is connectionless, meaning that the same UDP socket can be used to send or receive messages to or from more than one process. Compare to TCP.

unicast A message sent from one process to another process (point-to-point communications). Both UDP and TCP provide unicast messaging.

URI **U**niform **R**esource **I**dentifier. A unique identifier for abstract or physical resources on the World Wide Web.

user attribute An optional object associated with a region or a data entry where an application can store data about the region or entry. The data is accessed by the application only. GemFire Enterprise does not use these attributes. Compare to region attributes, which are used by GemFire.

value constraint Enforcing a specific entry value type. The `value-constraint` region attribute, when set, constrains the entries in the region to values of the specified object type. Value constraints

	can be used to provide object typing for region querying and indexing. The value-constraint is only checked in the cache that does the entry put or create operation. When the entry is distributed to other caches, the value constraint is not checked.
value-index	An XPath index that operates much as a structure-index does, but that separates the nodes that match the XPath expression into sets mapped by each node's value. This allows further filtering of the nodes to be evaluated in a query by going directly to those with a specific value. See also structure-index and range-index.
view	A collection of member identifiers that defines the membership group in JGroups.
VM	**V**irtual **M**achine, also referred to as a Java VM.
XML	E**X**tensible **M**arkup **L**anguage. An open standard for describing data, XML is a markup language similar to HTML. Both are designed to describe and transform data, but where HTML uses predefined tags, XML allows tags to be defined inside the XML document itself. Thus, virtually any data item can be identified. The XML programmer creates and implements data-appropriate tags whose syntax is defined in a DTD file or an XML schema definition.
XML schema	The definition of the structure, content, and semantics used in an XML document. The XML schema is a superset of DTD. Unlike DTD, XML schemas are written in XML syntax, which, although more verbose than DTD, are more descriptive and can have stronger typing. Files containing XML schema definitions generally have the `xsd` extension.
XPath	A language that describes a way to locate and process items in Extensible Markup Language (XML) documents by using an addressing syntax based on a path through the document's logical structure or hierarchy.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W